A Design and Control Methodology for Human Exoskeletons

by

John Ryan Steger

B.S. (Rice University) 2001
M.S. (University of California, Berkeley) 2003

A dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Mechanical Engineering

in the

GRADUATE DIVISION

of the

UNIVERSITY OF CALIFORNIA, BERKELEY

Committee in charge:
Professor H. Kazerooni, Chair
Professor Paul K. Wright
Professor Sara Beckman

Spring 2006

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

A Design and Control Methodology for Human Exoskeletons

by

John Ryan Steger

Abstract

A Design and Control Methodology for

Human Exoskeletons

by

John Ryan Steger

Doctor of Philosophy in Mechanical Engineering

University of California, Berkeley

Professor H. Kazerooni, Chair

Carrying a payload directly on the body is an unavoidable aspect of human life. Human bipedal locomotion knows no equal: people travel on foot to virtually every corner of the globe. Despite the efficiency and convenience of wheeled apparatus, uneven terrain, enclosed environments and accessibility limits require virtually every transportation task to include a phase in which material goods must be physically carried by a person. As of today, no artificial intelligence or programmed behavior has been able to match a human's ability to balance and maneuver in unstructured real-world environments.

The Berkeley Lower Extremity Exoskeleton solves the problem of supporting and carrying heavy loads on the body and allows a person to navigate unencumbered by the weight of the payload they are carrying. The Berkeley Lower Extremity Exoskeleton is an anthropomorphic and energetically autonomous robotic device comprised of two legs, a backpack, a harness system and a control computer that provides a wearable load support platform.

This thesis presents a control scheme called Sensitivity Amplification Control that enables an exoskeleton to support a payload and shadow the movement of the wearer in an intuitive and unobtrusive manner. The control algorithm developed here increases the closed-loop system sensitivity to its wearer's forces and torques without any measurement from the wearer. This strategy requires an accurate dynamic model of the system but does not require direct measurements from the human. The trade-off between not having sensors to measure human action and the sacrificed robustness due to model parameter variation is described. A modification to the controller is also explored that partially circumvents this limitation.

Professor Hami Kazerooni

Dissertation Committee Chair

This thesis is dedicated to Rachel.

I could not have done this without your love

and support. I thank you with all of my heart.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to thank my advisor, Professor Hami Kazerooni, for his vision and determination in bringing the exoskeleton project to the Berkeley, for assembling such an incredible team of dedicated engineers, and for inspiring each of us to dig deep into engineering and tackle the hard problems. Your patience, support, and guidance over the past five years have made me a better engineer.

Thanks to all of the members of the Berkeley Robotics and Human Engineering Laboratory. Without your brilliance, dedication and teamwork none of this would have been possible. Special thanks to Jean-Louis Racine for laying the groundwork for almost every aspect of the exoskeleton project. Thanks to Andrew Chu and Adam Zoss for transforming the concept of a human exoskeleton into a physical reality. Thanks to Sung Hoon (Sunny) Kim, George Anwar, and Mamuda Abatcha for designing the electronics and communication that brought the exoskeleton to life. Thanks to Lihua Huang for her help with the control of the exoskeleton. And, special thanks to all of the other incredible people that have taken part in making this project a success.

Chapter 1

Introduction

1.1 Thesis

This thesis develops the control and design techniques needed to create a robotic device, called a human exoskeleton, which reduces or eliminates the burden of carrying load on the body. The human body and human bipedal motion provide an unsurpassed load support and transportation platform in terms of versatility and maneuverability in environments that are otherwise inaccessible to wheeled transportation. Supporting a payload on the body is a universal component of human life. For some soldiers, firefighters, and disaster recovery workers, payloads are a significant fraction of one's body weight and carrying them is both unavoidable and essential to their duties. Unfortunately, carrying heavy loads is not only fatiguing, but also a potentially hazardous distraction and injurious activity.

Fig. 1-1 Illustration of the Berkeley Lower Extremity Exoskeleton (BLEEX) from the 2004 *New York Times: Year in Ideas* [1]

A human exoskeleton is worn by a person and carries payload, such as a heavy backpack, that would have otherwise been supported by the person's own musculoskeletal system (Fig. 1-1). To accomplish this, the exoskeleton must act so that it does not impede the person's behavior (e.g. standing, sitting, walking, running, etc.) and yet also act to ensure that forces associated with the payload always travel through the exoskeleton structure and not through the human body. An exoskeleton has the benefit of reducing the fatigue associated with load carriage, allowing a person to carry greater loads over

longer time periods and distances. Additionally, a person wearing a load carrying exoskeleton is less prone to develop injuries from stress to nerves, muscles, bones, and joints that would be caused by carrying a heavy load unassisted [2]. These benefits come at the cost of having a physical device in close proximity to the body that could require a power source that needs replenishing, can physically malfunction and stop carrying load, or inhibit the person in some situations or maneuvers. The goals of this thesis are as follows:

1) Build a framework for controlling a lower extremity human exoskeleton that ensures that it always carries the payload and never impedes the motion of the human.

2) Demonstrate how this control strategy is applied to a real exoskeleton robot and verify its performance and limitations both mathematically and through experimentation with the real system.

3) Develop guidelines for designing the hardware and control of lower extremity exoskeletons that strike a balance between robustness and performance by incorporating control directly into the hardware design of the device.

In this thesis I develop and explore a control algorithm that assumes a preexisting robotic system configuration: an assembly of links connected together by joints, acted upon by force/torque actuators such as hydraulic cylinders or electric motors, and outfitted with sensors that can feed information about the kinematic and dynamic state of the robot back to a computer running the control algorithm. The control framework provides the process for interpreting the information from the sensors and determining the appropriate commands for actuators.

What is a human exoskeleton?

Fig. 1-2 Ant (*Formicidae*) SEM* image showing an insect exoskeleton [3].

In scientific terms, an exoskeleton refers to an insect or crustacean's hard outer structure (Fig. 1-2) that protects it from the environment and provides structural support for the organism [4]. We borrow from this biological definition and classify a robot that is worn externally by a person and that acts to carry a payload and/or provide structural support for the person, as a *human exoskeleton.* While the 1st and 2nd generations of exoskeletons developed in our research group and discussed in this thesis were not designed to provide armor or environmental protection for the wearer, future versions currently being designed are incorporating these insect exoskeleton benefits.

Unlike unrealistic fantasy-type concepts fueled by movie makers and science fiction writers, the lower-extremity exoskeleton conceived at Berkeley (Fig. 1-3) is a practical load-carrying platform that, in operation, should be transparent to the user. It is not intended to impart super-human strength or speed. The reasons for this are: 1) eliminating the burden of load carriage solves a real and current source of injury and fatigue in a wide variety of applications; 2) a device capable of augmenting human strength implies that the person wearing the exoskeleton could be put situations that would cause bodily harm if the exoskeleton were to fail and stop supporting the load; 3) an exoskeleton intended to enhance speed would necessarily force the human musculoskeletal system to move at speeds the body was not designed to handle.

* Scanning Electron Microscope (SEM)

Exoskeleton control framework

The control framework developed in this thesis is called Sensitivity Amplification Control (SAC) and is designed to ensure that the exoskeleton moves in response to forces and torques applied by the user. At the same time, the SAC scheme seeks to maintain a comfortably low interaction force between the user and the exoskeleton. This control framework does not rely on any force sensing at human-machine interface points. The result is that with the controller operating, the exoskeleton seems to "shadow" human movement without the user having to make a conscious effort to control it (no joystick, steering wheel, etc.). The SAC scheme is unique in that it relies on human's own closed loop motor control (muscles, nervous system, brain, and kinesthetic feedback) to stabilize the combined human-machine system.

Fig. 1-3 The Berkeley Lower Extremity Exoskeleton (BLEEX) and pilot Ryan Steger.

Application of control framework on exoskeleton hardware

This thesis demonstrates the application of the SAC scheme on exoskeleton hardware developed in the Berkeley Robotics and Human Engineering Laboratory between 2001 and 2005. The process of applying the SAC scheme to a multi-degree-of-freedom system is covered and experimental results are presented in which the SAC scheme allows a person wearing the exoskeleton to maneuver freely while it supports a heavy payload.

Incorporating controls knowledge into hardware design

Simplification and robustness became primary goals for the exoskeleton project once the overall concept of payload support via a powered robotic suit had been successfully demonstrated. To this end, the design goals for the exoskeleton shifted to favor fewer actuators, less power consumption, and fewer sensors and control electronics. By combining observations from the biomechanics of human motion and walking with the ability to experimentally program the actuators of the exoskeleton hardware to simulate alternate mechanical behaviors and impedances, it was possible to create new mechanical designs for the exoskeleton with the behavior of the SAC scheme implicitly embedded in the mechanical hardware. A case study of this process is presented in which a passive (un-powered) exoskeleton ankle is designed and experimentally tested.

1.2 Motivation

Many tasks require transporting a load in situations where a wheeled vehicle or external apparatus (e.g. a sled or cart) is cumbersome or simply not feasible. These situations sometimes require that a person's hands be free to carry other gear or to perform a task. These situations range from everyday tasks such as navigating inside buildings that include stairs and narrow passages, to more extreme environments such a those faced by a

soldier in combat on uneven difficult terrain. In each of these situations a person typically carries gear attached to the body with a harness in the case of a backpack, through a waist belt pack, or some other system that secures load to the body. Designing robotic systems that combine the flexibility of having a payload attached closely and unobtrusively on the body with the ability to reduce or eliminate the physical exertion and energy expenditure required to support the load is the focus of the human exoskeleton research in the Berkeley Robotics and Human Engineering Laboratory.

Since the late 1960's load-carrying, and in some cases strength-enhancing, human exoskeleton devices have been studied with limited success [5]. For each attempt, which will be discussed in greater detail in Chapter 3, technological limitations prevented the integration of sensing, control, and actuation into a practical, safe, and autonomous system. Advances in computing, sensing, and power supply technology lead our group to re-examine the idea of creating a human exoskeleton in late 2000. By mid 2003 the first generation prototype of the Berkeley Lower Extremity Exoskeleton (BLEEX) had been completed and included the necessary high performance computing, sensing, actuation, and energetically autonomous operation to make a practical human exoskeleton. Controlling this robot such that it moved in concert with the human, did not impede the human's motion, and still supported the payload was the focus of my research that led to the development of the Sensitivity Amplification Control scheme.

Classical control theory employs feedback of information about the state (current, past, and/or predicted output) of a given plant in order to choose control commands that cause the plant to behave in a specific manner (such as track a desired parameter like force, position, or temperature). In practice this translates to decreasing the sensitivity of the

output of a plant to external disturbances such a noise, friction, or un-modeled dynamics on the system. This classical theory has been successfully applied to commercial robotic systems worldwide because it allows them to perform repetitive tasks with precision and accuracy over long periods of time. This is a result of the controller's feedback mechanism making the robot insensitive to variations in its environment, mechanical performance or task at hand.

In contrast, a human exoskeleton has goals that are quite different from classical control. Because the Berkeley exoskeleton was designed without sensors attached directly to the human, the controller must make the exoskeleton very sensitive to disturbances caused by motion of the user. In other words, the exoskeleton controller needs to *amplify* sensitivity to the wearer's motion. Achieving this sensitivity amplification without causing controller instability required considering the role of both the feedback loop formed by the exoskeleton controller and the feedback loop through the human's own nervous system. While there are significant costs related to modeling and model parameter robustness associated with this approach (discussed in Chapter 4), the Sensitivity Amplification Control scheme represents a new approach to the control of human-machine systems.

Soldier load carriage

The lower extremity exoskeleton project at U.C. Berkeley was funded primarily through a grant from the Defense Advanced Research Projects Agency (DARPA), which is a research funding organization under the U.S. Department of Defense. A U.S. Army infantry soldier was the "customer" for the exoskeleton and the target as far as design parameters were concerned. Soldiers routinely carry heavy loads over long distances and numerous publications exist linking heavy load carriage directly to military losses, poor

performance and unnecessary deaths [6-9]. Army guidelines suggest a maximum rucksack payload for combat of 22kg and for an approach march of 33kg [10]. However, the real average loads measured on soldiers in the field were 40kg and 73kg, respectively [10]. Based on 50th percentile U.S. soldier body mass [8], these measured values represent carrying 55-100% of body weight as additional payload. As a reference point, accepted "safe" backpack loads are considered to be 33-40% of body weight [11]. The loads carried by soldiers are not only fatiguing, but represent a real threat to long term health. The need for a device or method that reduces the load on a soldier is overwhelmingly clear.

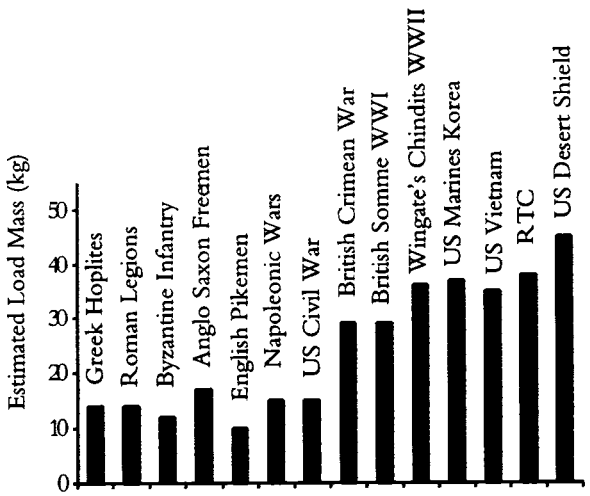


Fig. 1-4 Loads carried by various infantry units through history (adapted from [12]).

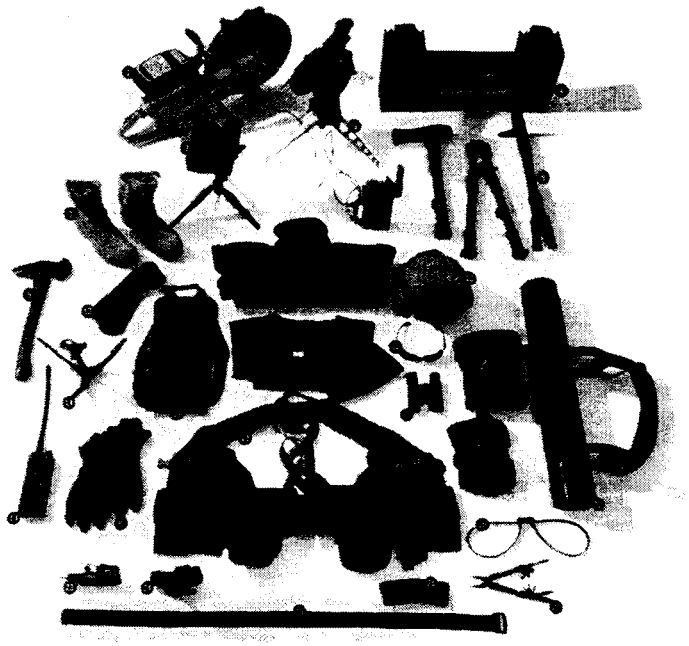


Fig. 1-5 Additional gear being *added* to U.S. soldier's basic combat gear for fighting in urban environments [13].

Fig. 1-4 shows a chronological history of loads carried by various infantry units throughout history. Despite significant advances in lightweight materials technology over the past 100 years, the total weight carried has continued to rise. Even as far back as World War I, soldiers were carrying over 85% of their body weight on their backs [9]. According to [13], recent changes in the nature of combat from large battles with mechanized rolling equipment to urban door-to-door combat is leading to even higher payloads. Fig. 1-5 depicts gear for urban combat that is currently being added to the U.S. soldier's basic combat load, indicating that the load carriage problem is worsening still.

Non-military applications

Exoskeletons have numerous applications that extend beyond soldier load carriage. It can provide disaster relief workers the ability to carry tools, food, and medical supplies into areas no longer accessible to vehicles. The enhanced load carrying ability could allow medical and aid workers to carry injured people from disaster sites, enabling them to be one-person ambulances. Similarly, an exoskeleton could allow firefighters to carry breathing apparatus and life-support equipment with less fatigue and for longer periods of time. Simplified versions could even one day offload the weight carried by recreational backpackers and hikers.

From an industrial perspective, an exoskeleton could be employed in almost any scenario where heavy parts, packages or containers need to be carried between workstations, loaded on and off vehicles or delivered to various locations. One could imagine mail carriers using an exoskeleton on routes that require hand delivery and door-to-door service. Commercial and residential movers could use exoskeletons to reduce the load and consequent injury in their spine, hips, and legs.

Movie and TV film crews could make use of an exoskeleton to provide a stable, yet mobile platform for heavy camera, video, and microphone equipment. As a consumer product, a simple exoskeleton might even be worn by parents to allow them to carry their children on their back without feeling the weight. Provided the exoskeleton is properly designed and does not encumber a person's normal actions, any scenario in which weight must be physically transported could potentially be made less injurious and less difficult through the use of a human exoskeleton.

Orthotic and prosthetic applications

At this point only applications of an exoskeleton to healthy individuals as a means of augmenting their ability has been explored. It is also conceivable that an exoskeleton could be used as an orthotic device. An orthotic is a device designed to support, correct, or align abnormal or weakened joints and limbs [14]. The exoskeleton's rigid structure could serve as a platform for adding orthotic support to the lower limbs or back. While it would require significant modification, the exoskeleton could also be used as a prosthetic device for individuals who are missing portions of a limb or have lost components of lower limb function. The changes that would need to take place to create a prosthetic exoskeleton would include altering the connection between the human and the exoskeleton such that, rather than support a payload, the exoskeleton actually supports a portion of the person's own body weight. Additionally, the control scheme would have to be altered because the SAC architecture assumes that the person has voluntary control of each joint. Possibilities for controlling a prosthetic exoskeleton include using feedback from an opposite sound limb ("echoing" or mirroring the behavior of the sound limb with the appropriate phase offset), measuring residual nerve activity, or using muscle activity of another portion of the body to control the exoskeleton leg. These control

techniques have been successfully applied in various research and commercial prosthetic devices [15-19].

1.3 Thesis contributions

The development of BLEEX and subsequent generations of exoskeletons in the Berkeley Robotics and Human Engineering Laboratory has been a large undertaking involving many stakeholders. Graduate students, my advisor Prof. Hami Kazerooni, staff engineers, undergraduate assistants, and outside consultants have all contributed to various aspects of the project. I became involved in an auxiliary role on the project at its beginning in 2001 while completing my Master of Science thesis project on the design and control of a haptic feedback device in the same lab. In 2003 I began this thesis work when I took over as lead of the control portion of the Berkeley Lower Extremity Exoskeleton project.

When I began work on control, the basic mechanical design of BLEEX as seen in Fig. 1-3 had been completed and implementation of an early version of the exoskeleton control and interface software (developed by a previous Ph.D. student, Jean-Louis Racine [20]) had begun.

My contribution to this portion of the project was to complete the testing and debugging of the software and bring the exoskeleton to the state that it could safely and reliably be coupled with a human operator. Once this was complete, BLEEX became the platform on which I developed and tested the Sensitivity Amplification Control scheme that, for the first time, enabled a person to wear the exoskeleton and walk freely while it supported the payload. Near the end of my graduate studies I began work as part of a team designing a significantly lighter, more capable, and less power intensive 2nd generation

BLEEX. As part of this work, I have applied my experience with BLEEX control as a design guideline and I am responsible for the passive foot-ankle-shank component of the current 2nd generation BLEEX.

1.4 Thesis contents (outline)

A short description of each chapter is given below:

- Chapter 2 gives an overview of human biomechanics. In particular, this chapter focuses on the kinematics and dynamics of human walking as they apply to exoskeleton design and the biomechanics of load carriage.

- Chapter 3 provides a history of exoskeleton research, related robotics projects, and the Berkeley Lower Extremity Exoskeleton project. A categorization that demonstrates how these projects fit into the larger field of robotics research is presented. Also, control strategies used for previous exoskeleton related research are discussed.

- Chapter 4 develops the Sensitivity Amplification Control scheme by examining a simple one-degree-of-freedom model of an exoskeleton leg. The role of the human's dynamics are then added to the analysis and the overall stability of the controller is discussed.

- Chapter 5 covers the implementation of the SAC scheme on the actual multi-degree-of-freedom BLEEX hardware. Results from experiments with the exoskeleton on a test-stand and in actual walking situations are discussed. A modification to the control is presented which circumvents some of the issues of parameter robustness issues described in chapter 4.

- Chapter 6 concludes with a discussion of the results and offers direction for future investigation.

Chapter 2

Human Biomechanics

Before discussing designing a robot that can walk with a person, it is important to understand how a human walks unassisted by an exoskeleton. How do the limbs move? How does we progress forward while walking? How do we transfer load between our feet? How much do we alter our behavior when carrying a heavy load? "Walking" implies a continuous motion, but questions of how we initiate motion, how we come to a stop, how we ascend and descend stairs and ramps, and perform other maneuvers is also important to take into account if an exoskeleton is to do these same things while connected to our body. Nonetheless, "walking" is still the baseline performance test we have used for designing exoskeletons and the terminology, biomechanics and energetics of walking will all apply to these other cases.

2.1 Human anatomy and physiology

Biomechanics literature uses a common terminology to describe the morphology of the body. All terms reference the body when it is in the *orthograde* or "*anatomical position*:" standing erect, arms at the sides, facing forward [21]. The body is divided into a set of three orthogonal planes: *frontal* (or *coronal*), *transverse*, and *sagittal* (Fig. 2-1). The *mid-sagittal* plane is also referred to as the median plane. *Superior* refers to the direction towards the head (up) and *inferior* refers to the direction towards the feet (down). *Anterior* refers to the front of the body and *posterior* refers to the back. *Medial* is the direction pointing toward the midline of the body (the intersection of the mid-sagittal and frontal planes), and *lateral* points away from the midline. *Proximal* indicates the region of a body part closest to the

trunk and *distal* indicates the opposite direction, away from the trunk.

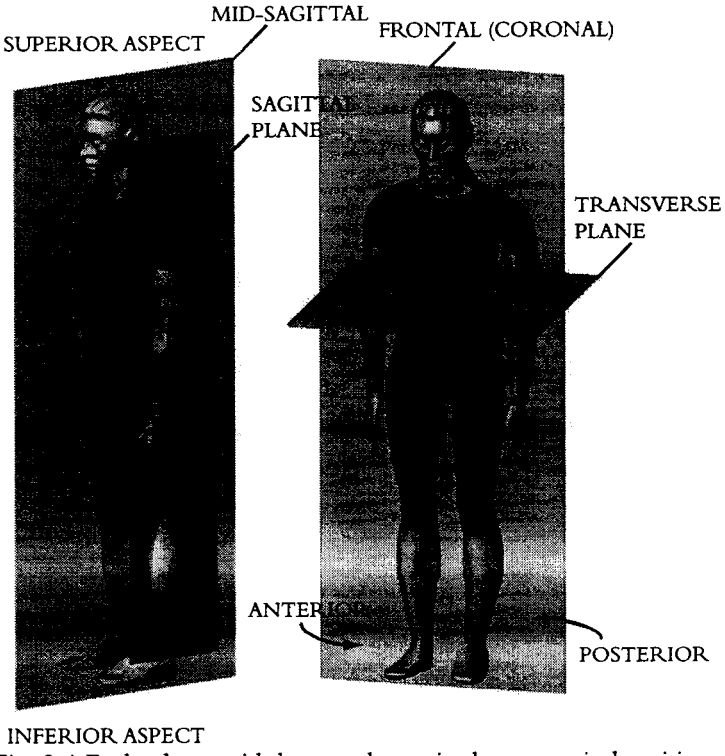


Fig. 2-1 Body planes with human shown in the anatomical position.

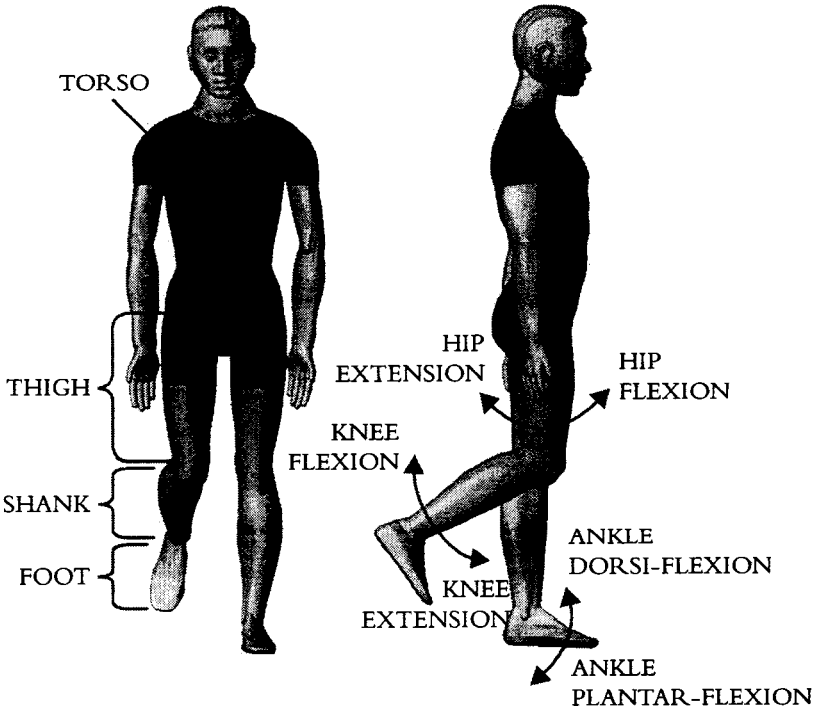


Fig. 2-2 Limb segment and joint motion terminology.

For the purpose of this thesis, limb segments, joints, and directions will be referred by the terminology in Fig. 2-2. Other joint motions not indicated in the figure include medial and lateral *rotation* of the foot, ankle, shank, thigh and hip in the transverse plane. Also, ankle and hip rotation in the frontal plane (about axes that intersects the flexion/extension axes) is called *abduction* when the foot or leg is swinging away from the mid-sagittal plane and *adduction* when the foot or leg is swinging toward the mid-sagittal plane.

The joint degrees of freedom shown in Fig. 2-2 represent a small subset of the actual degrees of freedom found in the human body. However, they are sufficient to understand the basic mechanics of human walking. When necessary, detail about specific additional degrees of freedom will be provided. Thorough reviews of human physiology and degrees-of freedom can be found in [22, 23] and [24].

HAT center of gravity and the body weight vector or "load-line"

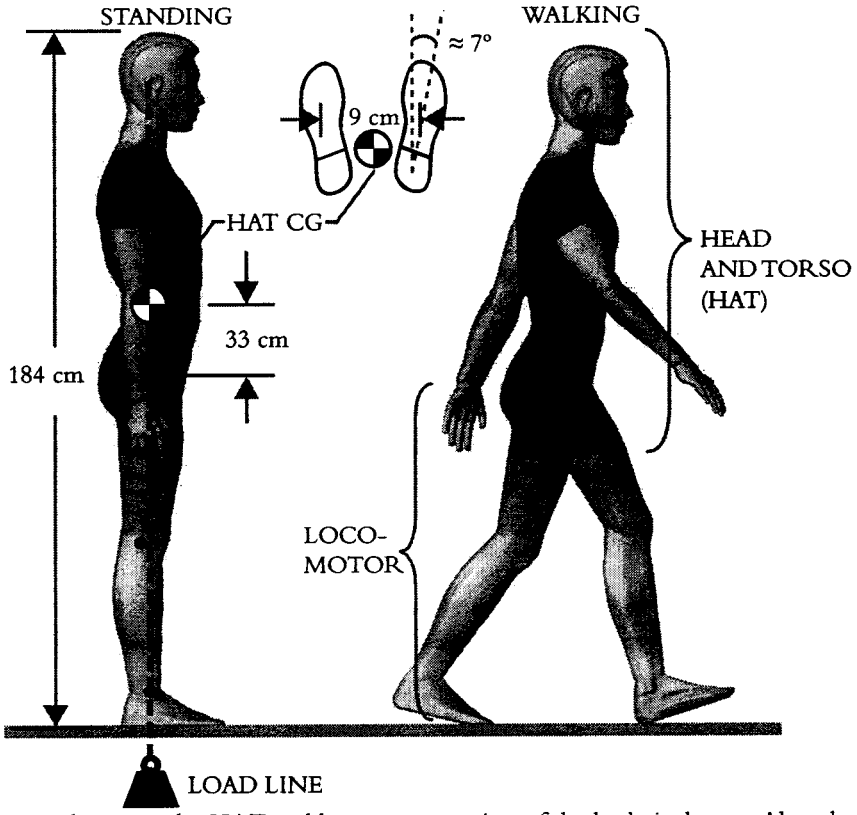


Fig. 2-3 The distinction between the HAT and locomotor portion of the body is shown. Also, the location of the load line for static standing is indicated. In the sagittal plane, the load line passes through the inner ear, HAT CG, slightly posterior to the hip, anterior to the knee, and anterior to the ankle. The dimensions given for the HAT CG position [25] and foot orientation [26] are based on a 50th percentile male.

Because this thesis deals primarily with the mechanics of walking, the convention from locomotion biomechanics of treating the upper body as a lumped mass is used [25]. The body is divided into two sections: the *locomotor*, which includes the legs and feet, and the *HAT*, which stands for Head And Torso. The HAT refers to this upper body lumped mass and also includes the neck, pelvis, arms, and hands. The HAT makes up approximately 70% of body mass [27]. When standing still, the sagittal plane location of the HAT center of gravity (CG) is in the superior half of the body, approximately 1/3rd of the distance between the hip joint and the top of the head [25].

An important concept illustrated in Fig. 2-3 is the body weight vector or *load line.*

The load line is the vector from the HAT CG down through the floor and in line with the equal and opposite vector of the ground's reaction force. The position of the load line in the sagittal plane relative to the body determines whether the person is balanced (the load line must pass through the ground contact patch), what magnitude of torques are necessary at each joint to support the body, and whether a joint is in a self-locking or "over-center" configuration. The over-center condition typically refers to the knee joint. If the load line passes anterior to the knee joint in the sagittal plane, the effect of the body weight is to straighten the leg into a locked position. This is advantageous in many scenarios because no muscle torque (i.e. energy) is needed to maintain knee angle and the knee is self-stabilizing. In the transverse plane for the static standing case, the feet are typically separated slightly and angled outwards (for a 50^{th} percentile male, the separation is approximately 9 cm and the outward angle or toe-out is 7^{0} [25].) The load line would pass halfway between the feet in the transverse plane.

2.2 The gait cycle

Walking, or more generally, locomotion, is composed of a repetitive sequence of limb movements. Forward movement is accomplished by shifting the weight onto one foot, swinging the opposite leg through the air and striking the ground ahead of the body, transferring the weight to the foot that was swinging, and then reversing the roles. This sequence is called a *gait cycle* (GC) and walking is composed of a smoothly connected series of gait cycles. Most biomechanics literature identifies the moment one foot strikes the ground as the "start" of the gait cycle, in large part because this impact is a readily measured data point in the laboratory [25, 28]. The impact is typically on the heel of the foot and this point in the gait cycle will be referred to as *heel strike* in this thesis. Some individuals strike the ground either flat footed or with the toe first. For this reason, some

biomechanics texts may refer to the beginning of the gait cycle by the more general term *initial contact* [25]. One full gait cycle refers to the interval between two successive heel-strikes on the same foot. The term stride is also used in some sources to indicate the period between two successive heel-strikes. A step refers to the transition from one foot heel-strike to the other foot heel-strike (there are two steps in each stride).

The gait cycle is divided into functionally distinct sections referred to as phases. The most basic division is between the stance phase when a leg is on the ground, and the swing phase when it is in the air. Because the gait cycle is a full step by both the right and left legs, there are two stance and two swing phases in each gait cycle. For normal individuals, the gait cycle is symmetric for the left and right legs. The gross normal distribution for timing of the phases is 60% stance and 40% swing [25].

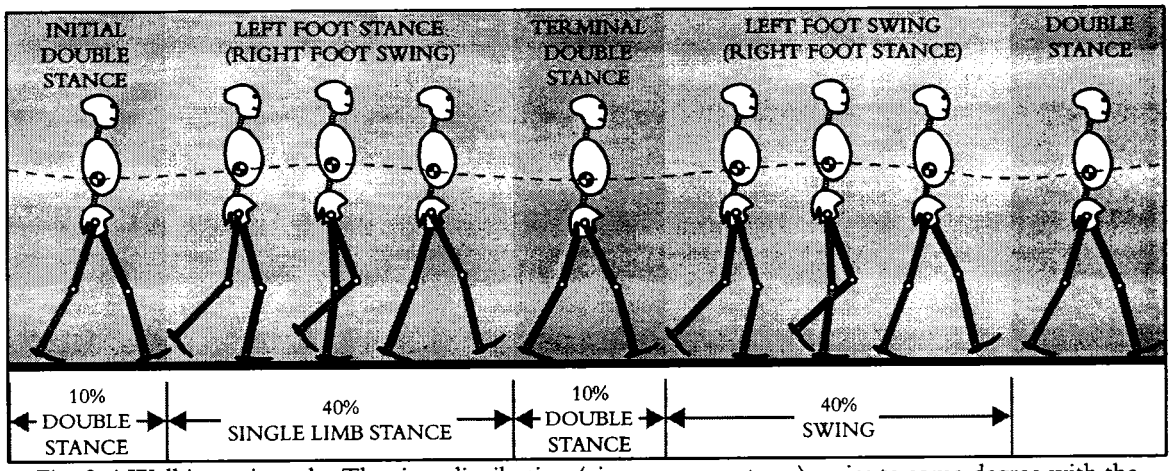


Fig. 2-4 Walking gait cycle. The time distribution (given as percentages) varies to some degree with the walking velocity. The percentages shown represent an average walking speed of 1.3 m/s [29]. Also indicated is the HAT CG path in the sagittal plane [28].

The stance phase of the gait cycle is further subdivided into two double stance phases (initial double stance and terminal double stance), and a single foot stance phase while the opposite leg is in swing. Double stance refers to the period when both feet are touching the ground. The timing distribution for the stance phases is 10% each for the two double

stance phases where both feet are on the ground and 40% for the single stance where only one foot is on the ground. During the double stance phases, the body weight is transferred from one leg to another. The terms initial and terminal refer only to the position in the overall gait cycle as each phase looks identical except for the change of foot.

Table 2-1

Gait cycle (GC) with phases and corresponding function (percent of GC from [25] is indicated for each phase.)

	PHASE	FUNCTIONAL OBJECTIVE
Initial Double Stance	2% Heel-strike	Shock absorbtion
	6% Loading Response	Weight transfer
	2% Toe-off	Forward propulsion
Single Stance	20% Mid-stance	Weight support, raise HAT CG over stance foot
	20% Terminal-stance	Weight support, advance body past stance foot
Terminal Double Stance	2% Heel-strike	Shock absorbtion
	6% Loading Response	Weight transfer
	2% Toe-off	Forward propulsion
Swing	13% Initial swing	Flexion of knee to clear ground and acceleration of swing leg
	13% Mid swing	Advancement of swing leg ahead of HAT CG and extension of knee
	13% Terminal Swing	Deceleration of swing leg and preparation for heel-strike

During the single limb stance phase, the HAT CG moves up and over the stance foot. The knee fully extends to a locked over-center configuration during this period. Raising the HAT CG due to the extending and locking knee is energetically expensive, so hips rotate in the frontal plane to partially compensate for this rise. The cost of raising the CG is justified by the overall energy savings that come from putting the knee in a stable configuration that does not require additional muscle force to maintain. Additionally, the forward momentum of the body is used to carry the HAT CG

up and over the locked stance leg. During this phase, the alternate leg is in the swing phase.

In the terminal double stance phase the stance leg is unloaded as the weight shifts to the alternate leg. As the weight is transferred, the knee buckles to prepare the foot to clear the ground during swing. This, coupled with the heel-strike of the alternate leg in front of the HAT CG causes the CG to descend slightly.

The swing phase has three main components, each lasting approximately 1/3 of the total swing phase [25]. During the initial portion after the foot leaves the ground and before the leg crosses over the opposite stance leg, the leg muscles rapidly accelerate the leg. During mid-swing the leg continues forward and the knee extends to prepare for heel-strike. During terminal phase of swing the leg muscles actively decelerate the leg to reduce the impact at heel-strike. Table 2-1 combines a temporal and functional representation of the various phases of the gait cycle. The timing for Table 2-1 was derived from [25] and [30].

2.3 Joint motion and energetics

The gait cycle establishes the mechanical conditions that must be met for each phase of walking. In order to create an exoskeleton that also follows the phases of the gait cycle and offers similar strength to a human, it is necessary to determine the exact motion and torques needed for each joint. From these values, the required power for each joint can then be derived and used to choose the appropriate

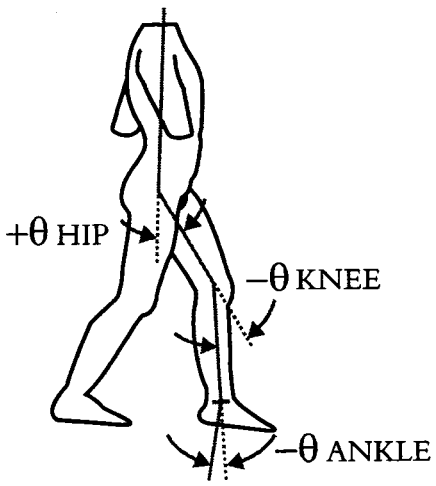


Fig. 2-5 CGA sign conventions (adapted from [31].)

actuators and a power supply. Clinical Gate Analysis (CGA) is the field of biomechanics that focuses on obtaining experimental measurements of body dimension and inertia, joint motion, joint torques, and other metrics associated with locomotion. The design of the first generation BLEEX made extensive use of published standardized CGA data from [32-34]. Details of the data analysis and application in the BLEEX design process can be found in [31, 35]. Fig. 2-5 shows the sign conventions used in the CGA data. Each joint is measured as the positive counterclockwise rotation of the distal link from the proximal link. The anatomical position corresponds to the zero angle for each joint. Torque is considered positive when acting counterclockwise on the distal link.

Ankle angle CGA data

CGA joint data from the University of Dundee [33], Hong Kong University [34], and Winter [32] is compiled for the ankle in Fig. 2-6. The gait cycle begins with heel-strike at time $t = 0$. For each plot, the stance phase occurs between $t = 0$ and $t = 60$. The swing phase occurs between $t = 60$ and $t = 100$. The data are normalized for a 75 kg person walking at 1.3 m/s. The dataset from [32] is one of the most commonly used reference sets from biomechanics literature, however other sources have been included to show variability in studies.

Fig. 2-6-A shows the time history of the ankle angle. The GC begins with a dorsi-flexion of 10-15° during the stance phase as the HAT CG travels up and over the stance leg. The ankle then planter-flexes 15-20° in terminal stance as the foot extends to propel the body forward. At toe-off the ankle dorsi-flexes to the neutral position to ensure the toe clears the ground during swing and then plantar-flexes at the very end of terminal swing to prepare for shock absorption at heel-strike. The ankle range of motion required for walking is small in comparison to the full range of motion (-38° to +35° [22]), however to accommodate maneuvers such as squatting, a much larger range of motion was designed into the exoskeleton ankles (see Chapter 3 for exoskeleton range of motion).

Ankle torque CGA data

The ankle torque plot in Fig. 2-6-B

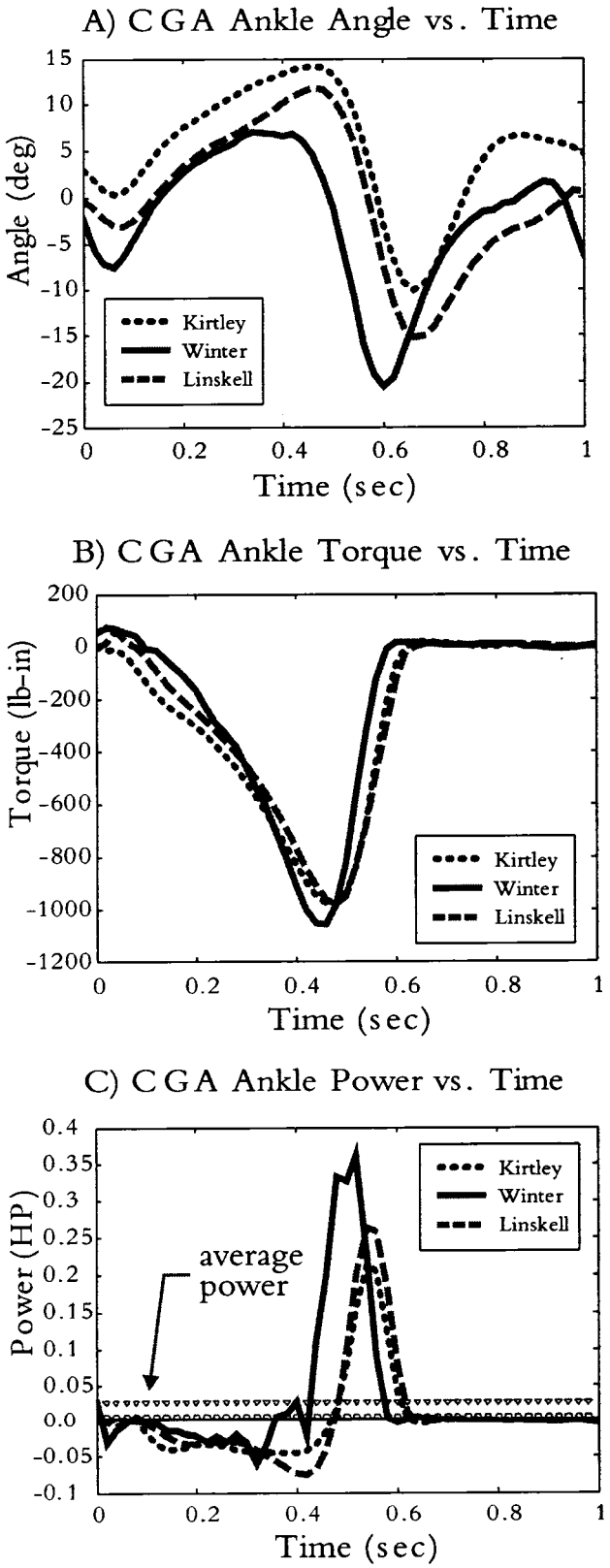


Fig. 2-6 CGA data of ankle angle, torque, and power during a single gait cycle (data sources as indicted).

contains some interesting characteristics. Peak positive torque is reached immediately following heel-strike. The ankle torque during this brief peak prevents the foot from slapping the floor as weight is added to the foot. During the stance phase the ankle provides and almost linearly increasing negative torque up to approximately 50% of the GC. The torque is negative which indicates that it is counteracting the tendency of the body to fall forward over the feet. The period from when the ankle torque begins increasing from the most negative point to where it levels off at toe-off ($t = 0.6$) generates the forward propulsion of the body during walking. For the remainder of swing, the ankle provides almost zero torque. In terms of exoskeleton design, Fig. 2-6-B shows that an actuator at the ankle only need to apply torque in one direction and it does not need to provide any torque during swing. This characteristic can be exploited in the mechanical design to conserve power and size.

Ankle power CGA data

The third important data set that can be derived from biomechanics is the instantaneous power that is consumed by each joint. Periods of negative power correspond to energy absorption while periods of positive power correspond to energy production. For an exoskeleton, positive power mean actuation is required while negative power means power can either be dissipated (wasteful, though difficult to avoid for mechanical systems) or the energy can be captured and stored for later use. Instantaneous power is calculated by taking the time derivative of the angle and multiplying by the instantaneous torque as shown in Eq.

$$P_{joint} = T_{joint} \cdot \frac{d}{dt}\left(\theta_{joint}\right) \tag{2.1}$$

Fig. 2-6-C shows the time history of the ankle instantaneous power. The single large

positive power spike corresponds to the forward propulsion provided by the ankle at the end of stance. It is also important to note that the area above the curve (energy) in the small but steady negative power region that occurs for most of stance is similar in magnitude to the area under the positive power spike before toe-off. For exoskeleton design, this implies that a scheme in which energy is stored during stance and released at the end could be a feasible and not interfere with the human's motion.

Co-contraction and human power consumption

It is important to note that the power plot does not correspond exactly to power generated or dissipated by the human. In the human body muscles provide only one direction of force (through contraction). Consequently each joint contains many pairs of muscles arranged in an opposing

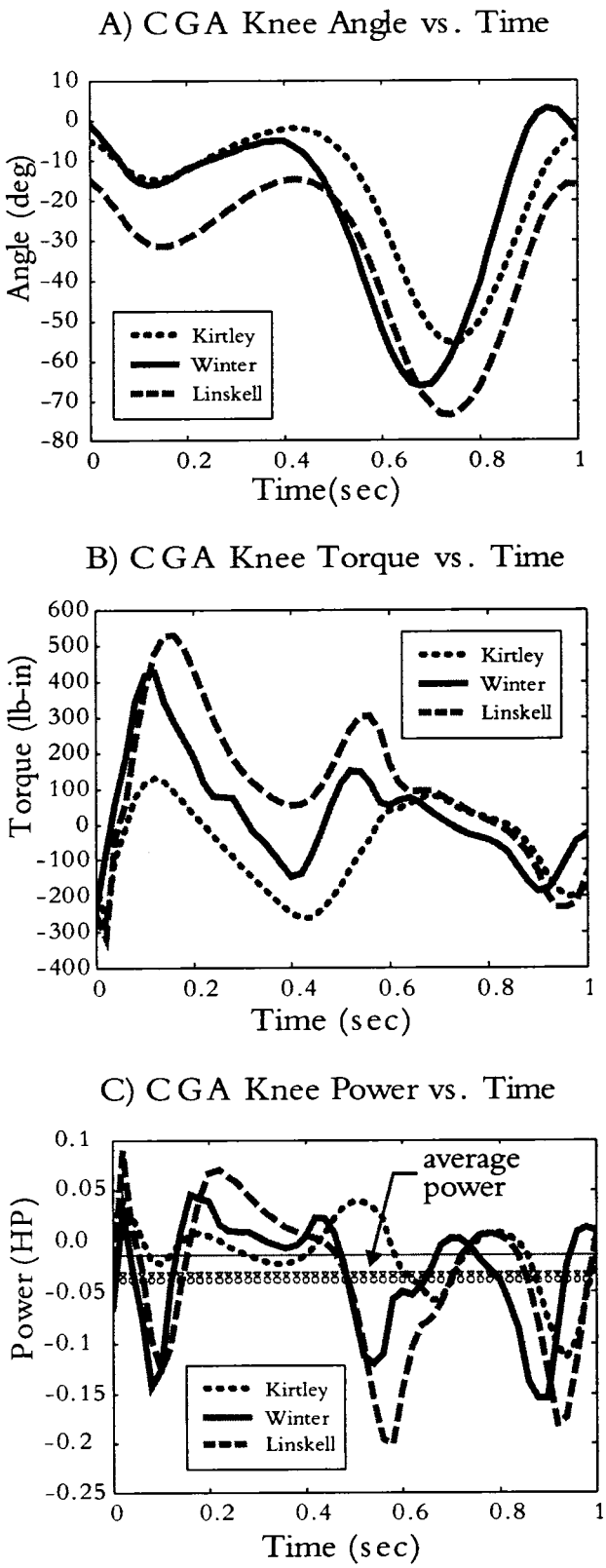


Fig. 2-7 CGA data of knee angle, torque, and power during a single gait cycle (data sources as indicted).

configuration: one group of muscles contract to cause extension while another group contract to cause flexion. In practice, the nervous system does what is called co-contraction (or co-activation) [36]. Both muscle groups exert force and the resultant external torque seen at the joint is the difference between the forces from the co-contracting muscle groups. Through co-contraction both the torque and the stiffness of the joint can be controlled independently [37, 38]. This also means that any amount of power up to the full output of the muscles associated with a joint can be dissipated without exerting a visible external torque [39, 40]. Some researchers have created robotic systems that employ co-contraction [41]. This type of system was evaluated at the beginning of the exoskeleton project but the high power cost associated with controlling stiffness through co-contraciton was prevented us from choosing a truly anthropomorphic actuator for the exoskeleton. Instead, Chapter 4 will discuss how this benefit of stiffness control can be added in the control software.

Knee angle CGA data

Fig. 2-7-A shows the knee angle in the GC as a function of time. The knee buckles approximately 10-20^{0} during early stance as the HAT CG travels over the stance foot. This helps to minimize the rise in the HAT CG and conserve power. Just prior to swing the knee begins a long smooth period of flexion to clear the foot as it swings forward. At the end of swing the knee slightly hyper-extends (positive angle) to ensure that the knee is in an over-center position at heel-strike. In this over-center configuration, the knee will mechanically stabilize to a straight locked position at heel-strike without the need for significant ankle torque. Maintaining this condition in the exoskeleton requires careful sizing such that the exoskeleton leg is fully extended and over-center at heel-strike. As will be shown in Chapter 5, in many experiments the exoskeleton legs were too long for the

wearer, meaning they were in a flexed state at heel-strike. This caused the leg to buckle unexpectedly on the user, leading to discomfort.

Knee torque and power CGA data

Fig. 2-7-B shows the knee torque as a function of time. Of note is the fact that the knee requires large torques in both directions. Fig. 2-7-C indicates that the knee requires multiple regions of both actuation and dissipation throughout the gait cycle. The average power for the knee (dotted lines in figures represent power averages over the complete gait cycle) offers the potential of employing an entirely dissipative mechanism at the knee which could lead to substantial power savings. Choosing an entirely passive design for an exoskeleton knee, while sufficient for level ground walking, would be inadequate for other maneuvers such as squatting or ascending slopes and stairs. For this reason passive knee actuation was

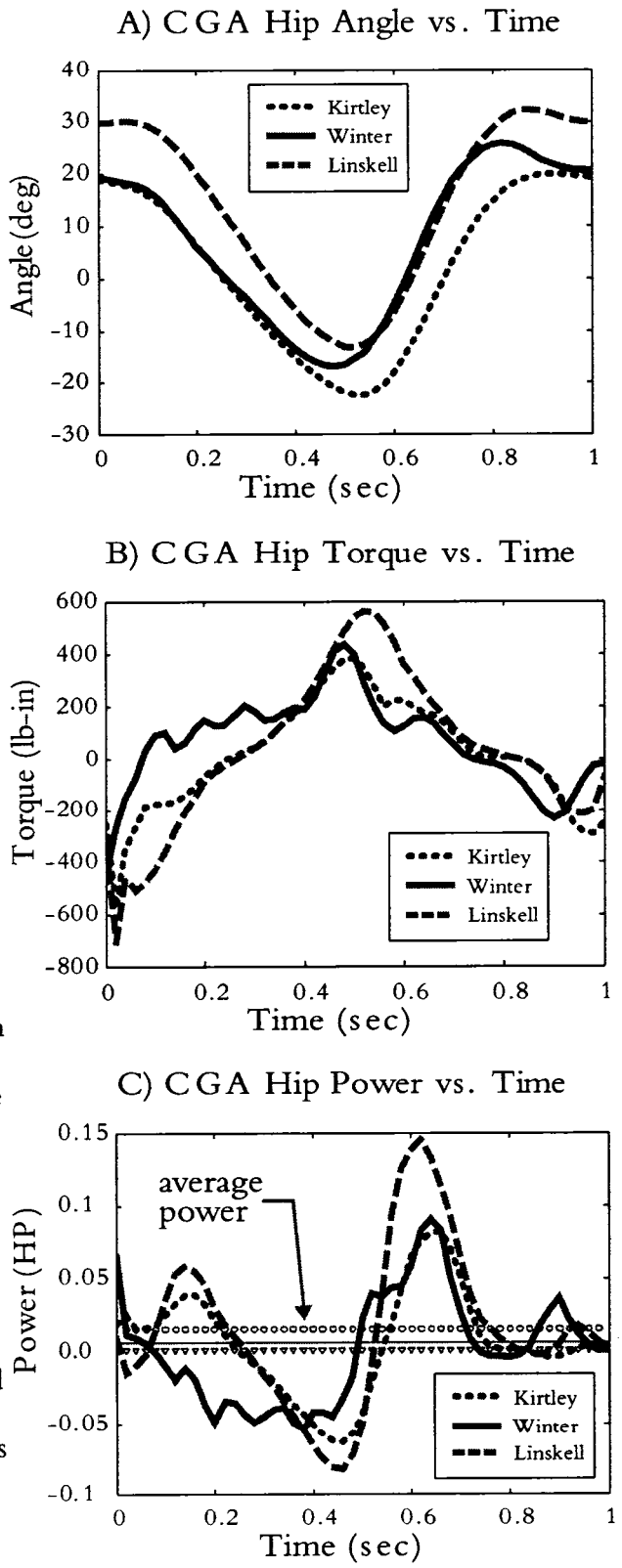


Fig. 2-8 CGA data of hip angle, torque, and power during a single gait cycle (data sources as indicted)..

not employed on BLEEX. In the current generations of the exoskeleton we have reconsidered a passive knee design in order to reduce system power consumption.

Hip angle, torque, and power CGA data

The hip motion during walking is an almost sinusoidal oscillation from 20° to -20°. The joint torques for the hip follow a similar pattern, oscillating almost symmetrically between large positive and large negative torques. Though the average hip power is slightly positive indicating actuation is necessary, the oscillation between power absorption and power generation is relatively slow. For exoskeleton design, this could accomplished by a small actuator and a compliant energy storage and release system like a spring.

Chapter 3

Exoskeleton Design and the BLEEX Project

3.1 History of load carrying exoskeletons

In the early 1960s, the Defense Department expressed interest in the development of a man-amplifier, a "powered suit of armor" which would augment soldiers' lifting and carrying capabilities [5]. In 1962, the Air Force had the Cornell Aeronautical Laboratory study the feasibility of using a master-slave robotic system as a man-amplifier. In later work, Cornell determined that an exoskeleton, an external structure in the shape of the human body which has far fewer degrees of freedom than a human, could accomplish most desired tasks [43]. From 1960 to 1971, General Electric developed and tested a prototype man-amplifier, a master-slave system called the Hardiman [42, 44-46]. The Hardiman was a set of overlapping exoskeletons worn by a human operator. The outer exoskeleton (the slave) followed the motions of the inner exoskeleton (the master), which followed the motions of the human operator. These studies found that duplicating all human motions and using master-

Fig. 3-1 "Hardiman" exoskeleton built by G.E. in the 1960's [42].

slave systems were not practical. Additionally, difficulties in human sensing and system complexity kept the Hardiman from walking or even being run with human operator inside the machine.

Several exoskeletons were developed at the University of Belgrade in the 60's and 70's to aid people with paraplegia resulting from spinal cord injury ([47-50]). Although these early devices were limited to predefined motions and had limited success, balancing algorithms developed for them are still used in many bipedal robots like the Honda Corporations "ASIMO" robot [51].

Current commercially available rehabilitation devices such as the "Locomat" use a similar pre-defined motion strategy to train muscles and nerve pathways for patients with locomotion impairment [53]. The "RoboKnee" is a powered knee brace developed by MIT that functions in parallel to the wearer's knee and transfers load to the wearer's ankle (not to the ground) [54]. Though actually more of a muscle augmentation device than a load supporting exoskeleton, the Roboknee uses an interesting force generator called a series elastic actuator [55, 56]. Series elastic actuators provide an inexpensive and robust way to create a high force, variable impedance actuator by coupling the actuator to a low stiffness force sensor and using closed loop feedback from

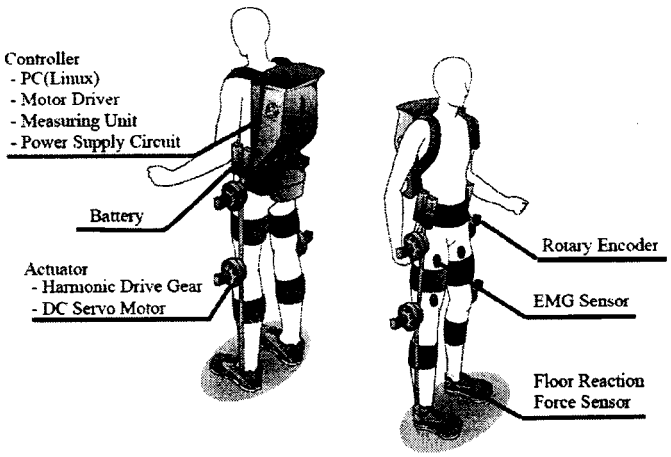


Fig. 3-2 HAL-3 exoskeleton from the Univ. of Tsukuba [52].

the force sensors to create compliance in the system. Though it is a common procedure in robotics to use force sensors and feedback to create compliant actuators, the force sensors used typically have a very high stiffness (e.g. a strain gauge measuring micro-strain magnitude compression of a metal link). This stiffness results in high frequency dynamics that can be excited by high control loop gains and cause chattering and instability in the actuation. Also, robust force sensors are typically very expensive (force sensors used on the BLEEX actuators cost approximately $400 each). The series elastic actuator measures the compression of inexpensive low stiffness springs placed in series with the actuator and relates this through Hooke's law to the force being applied. The effect of the low stiffness spring is to add a mechanical low-pass filter to the feedback loop. A penalty is consequently paid in the overall bandwidth of the closed loop system. Reported performance (565N, 7.5Hz, 28cm/sec, [54]) would be currently insufficient for BLEEX, however the benefits of low cost and simple actuator-level force control might make these a viable alternative in the future.

"HAL," which stands for Hybrid Assistive Leg, (actually, HAL 1 through 5 as of 2006) began as a lower extremity assistive orthotic and has evolved in a full body exoskeleton. Developed by the University of Tsukuba in Japan, HAL is connected to the patient's thighs and shanks and moves the patient's legs as a function of the EMG† signals measured from the wearer [52, 57, 58]. EMG signal based actuation was considered in the concept development phase of the BLEEX project as a way of determining human intent to move. Unfortunately, current EMG technology provides very low resolution data for muscle activation level (typically on, off, and one two three levels in between) [59]. In addition, for BLEEX the goal was to have very high sensitivity to human movement so this path

† Electromyogram: a measure of the minute actions potentials (voltages) generated on the surface of the skin when a muscle is activated.

was not followed. The EMG electrodes used to sense muscle action potentials on the skin must be carefully applied which would be impractical for a soldier load assist device that needs to be rapidly donned and doffed.

Previous exoskeleton related work at U.C. Berkeley

In our research work at Berkeley, we have separated the technology associated with human power augmentation into lower extremity exoskeletons and upper extremity exoskeletons. The reason for this was two-fold; firstly, we could envision a great many applications for either a stand-alone lower or upper extremity exoskeleton in the immediate future. Secondly, and more importantly for the division is that the exoskeletons are in their early stages, and further research still needs to be conducted to ensure that the upper extremity exoskeleton and lower extremity exoskeleton can function well independently before we can venture an attempt to integrate them. With this in mind, we proceeded with the designs of the lower and upper extremity exoskeleton separately, with little concern for the development of an integrated exoskeleton. I will first give a summary of the upper extremity exoskeleton efforts at Berkeley and then we will proceed with the description of the BLEEX project.

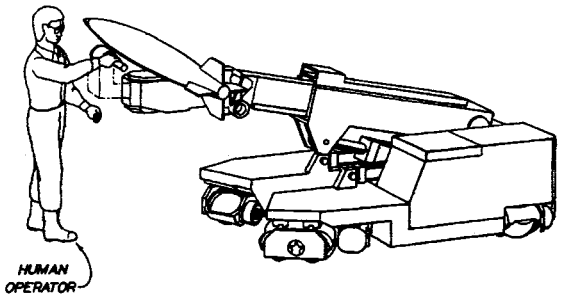


Fig. 3-3 "Human extender" upper body strength amplifier [60]

In the mid-1980s, my research group initiated several research projects on upper extremity exoskeleton systems, billed as "human extenders" [61-63]. The main function of an upper extremity exoskeleton is human power augmentation for manipulation of heavy and bulky objects. These systems, which are also known as assist devices or human power extenders, can

simulate forces on a worker's arms and torso. These forces differ from, and are usually much less than the forces needed to maneuver a load. When a worker uses an upper extremity exoskeleton to move a load, the device bears the bulk of the weight by itself, while transferring to the user as a natural feedback a scaled-down value of the load's actual weight. For example, for a 20 kg (44 lbs) object, a worker might support only 2 kg (4.4 lbs) while the device supports the remaining 18 kg (39.6 lbs). In this fashion, the worker can still sense the load's weight and judge her movements accordingly, but the force she feels is much smaller than what she would feel without the device. In another example, suppose the worker uses the device to maneuver a large, rigid, and bulky object, such as an exhaust pipe in an automotive assembly line. The device will convey the force to the worker as if it was a light, single-point mass. This limits the cross-coupled and centrifugal forces that increase the difficulty of maneuvering a rigid body and can sometimes produce injurious forces on the wrist. In a third example, suppose a worker uses the device to handle a powered torque wrench. The device will decrease and filter the forces transferred from the wrench to the worker's arm so the worker feels the low-frequency components of the wrench's vibratory forces instead of the high-frequency components that produce fatigue.

3.2 The Berkeley Lower Extremity Exoskeleton (BLEEX)

The Berkeley Lower Extremity Exoskeleton (BLEEX) is not an orthotic or a brace; unlike the above systems it is designed to carry a heavy load by transferring the load weight to the ground (not to the wearer). BLEEX has four new features. First, a novel control architecture called Sensitivity Amplification Control was developed that controls the exoskeleton through measurements on the exoskeleton structure rather than between the exoskeleton and the human [64-67]. This eliminates problematic human induced

instability due to sensing the human force [68]. Second, a series of high specific power and specific energy power supplies were developed that were small enough to make BLEEX a true field-operational system [69-71]. Third, a body LAN (Local Area Network) with a special communication protocol and hardware were developed to simplify and reduce the cabling task for all the sensors and actuators needed for exoskeleton control [72] and [73]. Finally, a flexible and versatile mechanical architecture was chosen to decrease complexity and power consumption [35, 74].

3.3 Design of BLEEX

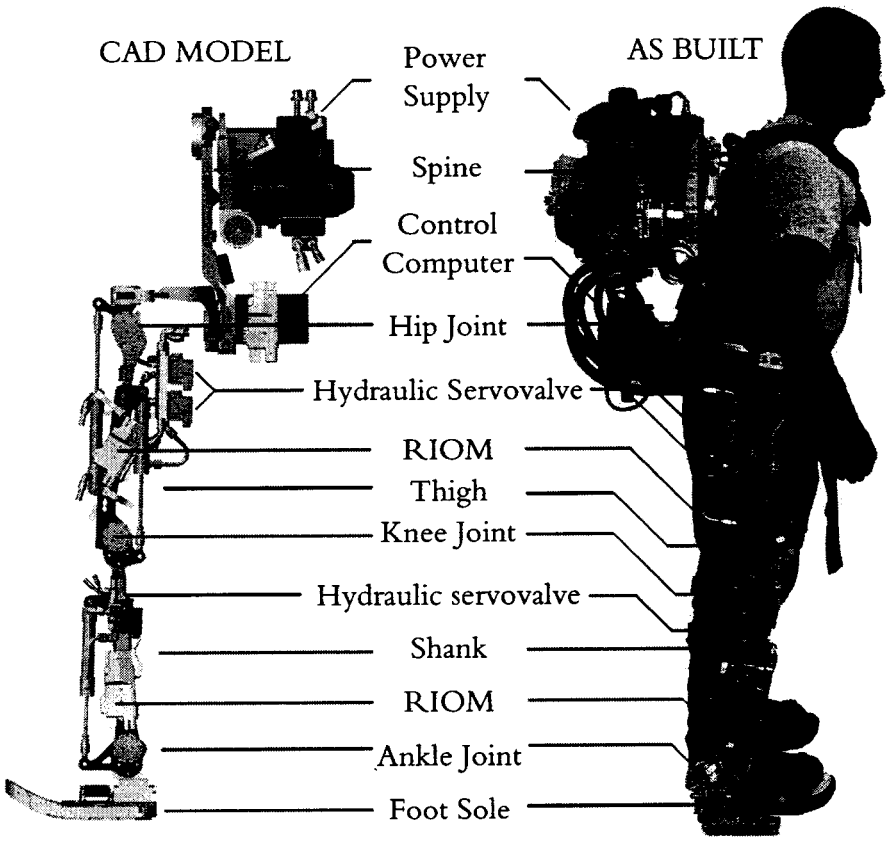
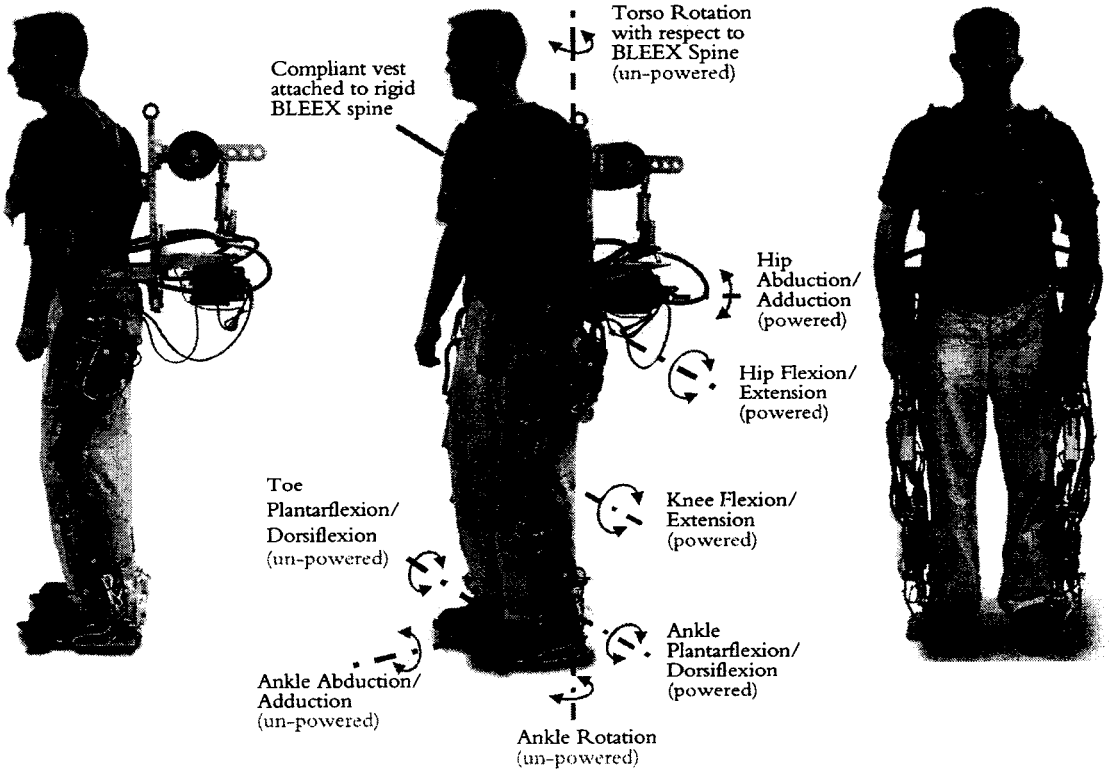


Fig. 3-4 BLEEX 3D CAD model and "as built" system components.

For BLEEX, a pseudo-anthropomorphic design was chosen to keep the device compact, ensure that the exoskeleton can access the same environments as a human, and to allow literature on human biomechanics to be applied as a model for the exoskeleton

kinematics and dynamics. The exoskeleton has a rigid spine that serves as a payload attachment point and an exoskeleton-to-human attachment point through a compliant harness. Three-segment legs, analogous to the human's thigh, shank, and foot, run parallel to the human's leg segments when the device is worn. Single DOF revolute joints connect each leg segment and connect between the thigh and spine on each side. A servo-valve-controlled hydraulic cylinder spans each segment pair to provide an active torque source at the hip (flexion and abduction), knee, and, ankle of each exoskeleton leg.



BLEEX degrees of freedom

Fig. 3-5 BLEEX degrees of freedom and actuation (an equivalent mass used in development is visible in place of the backpack power supply).

As shown in Fig. 3-5, additional un-powered passive degrees of freedom exist at the hip and ankle and include experimentally chosen passive impedances (created by steel springs and elastomers). These additional degrees of freedom were added to allow the exoskeleton to better approximate the DOF found in the human body. They increase

comfort for maneuvers that require motion outside of the sagittal plane such as turning, stepping side-to-side, and squatting.

BLEEX is considered pseudo-anthropomorphic because we have not included every human degree of freedom or attempted to match the joint behavior of the human exactly (e.g. the human knee uses a combination of rotation and sliding but the exoskeleton has a pure rotary joint). We determined, through extensive testing of un-powered mockups both in our lab and independently at the U.S. Army Natick Soldier Testing Center, that the kinematics of the configuration shown in Fig. 3-5 allow for unrestricted walking, running, kneeling, and crawling and therefore is sufficient for this design.

Connections between BLEEX and the human

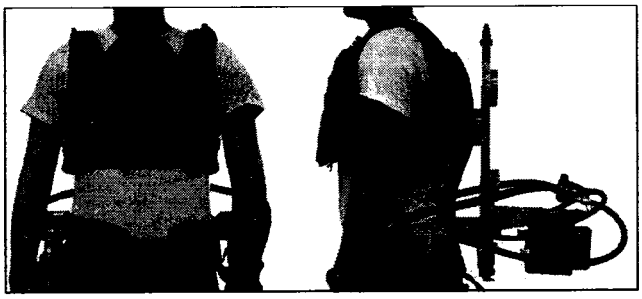


Fig. 3-6 Compliant BLEEX upper body harness.

The pilot and BLEEX have a mechanical connections at the torso and the feet; everywhere else the pilot and BLEEX have compliant or periodic contact (Fig. 3-6, Fig. 3-8). The connection at the torso is made using a custom vest. One of the essential objectives in the design of these custom vests was to allow the distribution of the forces between BLEEX and the pilot, thereby preventing abrasion. The vest is made of several hard surfaces that are compliantly connected to each other using thick fabric. The adjustment mechanisms in the vest allow for a snug fit to the pilot. The vest includes rigid plates (with hole patterns) on the back for connection to the BLEEX torso [31].

The pilot's shoes or boots (Fig. 3-8) attach to the BLEEX feet using a modified quick-release binding mechanism similar to snowboard bindings. A plate with the quick-release

mechanism is attached to the rigid heel section of the BLEEX foot. Early versions of the BLEEX system had the pilot wearing a standard U.S. Army issue boot that has had a mating binding cleat secured to the heel. The cleat on the modified pilot boot does not interfere with normal wear when the pilot is unclipped from BLEEX. Later modifications to this binding allowed an un-modified boot to be securely strapped to the exoskeleton foot.

The BLEEX foot is composed of the rigid heel section with the binding mechanism and a compliant, but load bearing, toe section that begins at midfoot and extends to the toe. The BLEEX foot has a compressible rubber sole with a tread pattern that provides both shock absorption and traction while walking. The rubber sole of the BLEEX foot contains embedded sensors, as shown in Fig. 3-7, that detect the trajectory of the BLEEX-ground reaction force starting from heel-strike to toe-off.

The sensors are constructed in the laboratory by sandwiching a conductive rubber sheet between to thin metal electrodes. The sensor from Fig. 3-7 is placed in a mold and cast in urethane rubber to make the BLEEX foot as shown in the right half of Fig. 3-8. Pressure on the sensor changes the resistance between the plates and circuitry in the BLEEX electronics converts this into digital (on/off) data for each segment of the foot. This information is used in the BLEEX controller to identify the BLEEX foot configuration relative to the ground.

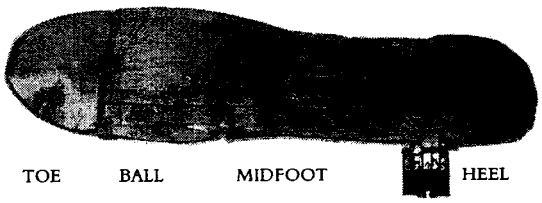


Fig. 3-7 BLEEX foot ground contact sensor.

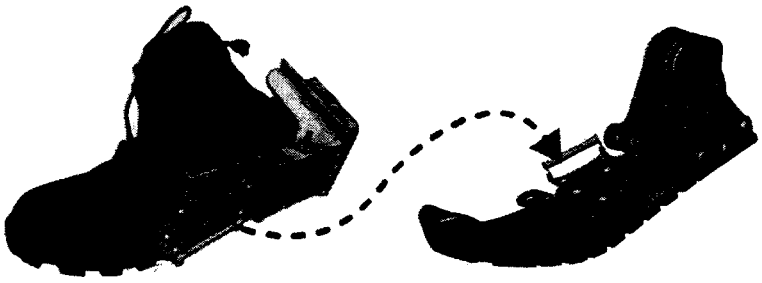


Fig. 3-8 Rigid attachment between human boot (left) and BLEEX foot via a quick-release cleat and binding mechanism.

Because the exoskeleton kinematics are close to human kinematics, appropriate ranges of motion for each degree of freedom could be approximated from human physiological data [35]. Slight human-machine kinematic differences are tolerated for design simplicity. These differences are not uncomfortable for the human because the human and machine are only rigidly connected at the extremities of the exoskeleton (feet and torso). Any other rigid connections would lead to large forces imposed on the operator due to the kinematic differences. However, compliant connections along the leg are tolerable as long as they allow relative motion between the human and machine. Because the inertias and masses of the exoskeleton leg segments were similar to the corresponding human limbs, the desired joint torques for the exoskeleton could be estimated using human Clinical Gait Analysis (CGA) data from [32-34].

BLEEX sensors

The Sensitivity Control Scheme developed in this thesis is based on an accurate inverse dynamics model of the exoskeleton dynamics. The accuracy of this model is dependant upon the static model parameters (i.e. the mass, length, inertia, and CG location of each component), and the state variables input into the system. The static model parameters can be carefully calculated, modeled, or measured during the design process. The state variables must be measured and calculated using appropriately chosen sensors. The state variables for the BLEEX dynamic model are the angle (θ), angular

velocity ($\dot{\theta}$), and angular acceleration ($\ddot{\theta}$) of each joint. Also, the torque at each joint is needed for the local joint torque controllers discussed in Chapter 5.

Details of the selection process for each sensor are covered in [20]. The joint angles are measured using rotary optical quadrature encoders. These sensors only give relative joint angle information (i.e. the angle between successive joints), so a digital inclinometer is added to the torso to give an absolute reference to ground. The joint encoder provides a resolution of 40,960 counts/rev after decoding in the control electronics. The clock rate of the control electronics (20Mhz) sets the upper limit of measurable joint velocity to be less than 3,068 rad/sec based on the minimum measurable increment of time used in the differentiation. This is well within the maximum human joint angular velocity of approximately 10 rad/sec from biomechanics data [23]. The clock rate also determines the slowest measurable angular velocity to be 1.83e-4 rad/sec based on the maximum measurable time increment. The joint angular velocity is determined by differentiating the angle from the encoder.

The joint angular acceleration is measured using pairs of linear accelerometers placed on either end of each BLEEX limb segment and oriented to sense acceleration tangential to the rotation of the limb. The angular acceleration of the limb is the difference in these two measured linear accelerations divided by the distance between the sensors. Maximum linear acceleration of human limb segments during walking is approximately 3g (where g is acceleration due to gravity) so 4g rated accelerometers were selected. The accelerometer bandwidth is rated at 400Hz, which is within the 10X control system bandwidth rule-of-thumb bandwidth needed to prevent phase lag problems.

The torque about each joint is measured by a force sensor placed in-line with the

hydraulic actuator spanning each joint. The geometry of the actuator, joint, and the joint angle can be used to obtain the joint moment arm and consequently, the joint torque. The maximum measurable joint torque on BLEEX is 170 Nm (with a safety factor of 1.5). The maximum torque from walking CGA data is approximately 135 Nm. The precision of the torque measurement is a function of the angle error used to determine the moment arm and the force sensor precision and is approximately 0.1% in the combined system [20].

BLEEX control electronics

Realization of the BLEEX control scheme requires a high-performance physical control architecture. Traditional centralized control architectures where a supervisory controller directly interfaces in a point-to-point fashion with all sensors and actuators in the system have been successfully implemented in the past [30, 51, 61]. They are generally feasible when a controller interfaces with small number of sensors and actuators and requires short wiring to them. Larger sophisticated multi-degree-of-freedom systems frequently require the control network to be compact, easily reconfigurable, expandable, and maintainable. Hence, a networked control system (NCS) was used on BLEEX as an alternative to the conventional centralized control system because of its advantages in flexibility, volume of wiring and capacity of distribution. The exoskeleton electronics system, called EXOLINK, was designed to simplify and reduce the cabling task for all of the sensors and actuators needed for exoskeleton control. The EXOLINK system represents the thesis work presented in [72] and is represented schematically in Fig. 3-9. The EXOLINK control network guarantees strict determinism, optimized data transfer for small data sizes, and flexibility in configuration.

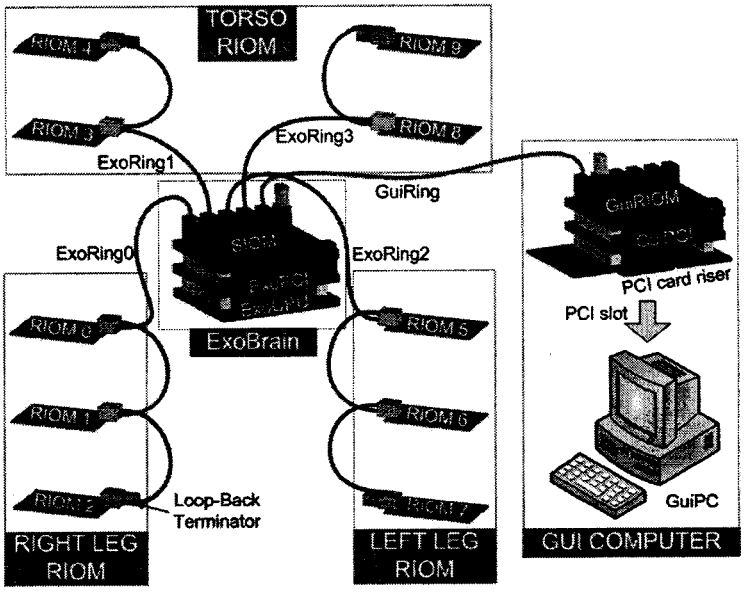


Fig. 3-9 Global view of EXOLINK networked control system and the external GUI debug terminal adapted from [75].

The EXOLINK "body" local area network

EXOLINK relies on a high-speed synchronous ring network topology where several electronic networked data aggregation nodes called Remote Input Output Modules (RIOM) reside in a ring (Fig. 3-10). Each RIOM is in communication with several sensors and one actuator in close proximity. RIOMs include eight sixteen-bit Analog-to-Digital Converters (ADC), two quadrature counters, eight bits of digital input and output, two Digital-to-Analogue converters (DAC) and analog filters for each sensor connection. Each RIOM also includes localized power regulation and isolation to minimize signal noise and system ground loops. A built-in Field Programmable Gate Array (FPGA) manages all RIOM data transaction and filtering. The distribution and location of RIOMs is generally chosen to achieve a minimum volume of wiring and a reasonable and convenient allocation of sensors and actuators to each RIOM. The data gathered by each module is encoded and transmitted digitally to a central computer through the ring network.

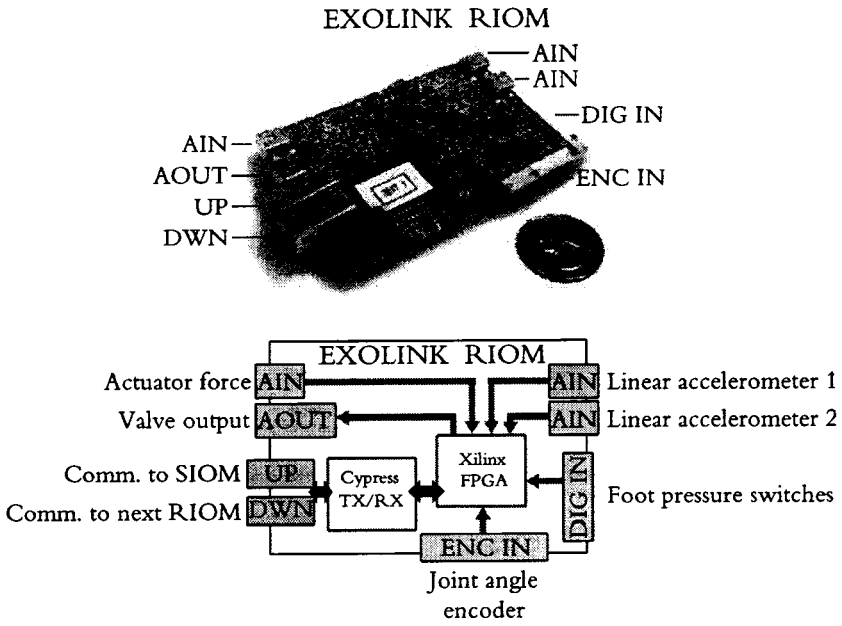


Fig. 3-10 EXOLINK RIOM photo and schematic. Each RIOM provides for 3 analog inputs (AIN), 1 analog output (AOUT), 6 digital inputs (DIG IN), 1 quadrature encoder input (ENC IN), and 2 network communication ports (UP and DWN). Two integrated circuits handle processing (an FPGA from Xilinx Inc.) and network communication (a transceiver from Cypress Semiconductor Inc.) respectively. The corresponding sensors connected to the RIOM are indicated on the schematic representation.

EXOLINK has four rings, two of which are associated with the two legs, each of which includes three Remote Input Output Modules. A third ring is connected to a Graphical User Interface for debugging and data acquisition, and a fourth ring is used to accommodate other electronic and communication gears that are not related to the exoskeleton, but which the pilot must carry. Each ring can accommodate up to eight RIOMs.

The EXOLINK includes a central microcomputer and a Supervisor IO Module (SIOM). The SIOM includes a FPGA programmed to serve as the communication hub for all four rings. A transceiver chip residing in the SIOM and all the RIOMs allows for data transfer at a rate of 1500 Mb/s. Currently, a 650 MHz Pentium PC-104 form factor microcomputer is used to implement the control algorithm, and the current Exoskeleton utilizes 75% of the I/O capability of the EXOLINK. The use of a high-speed synchronous

network in place of the traditional parallel method enables the exoskeleton to reduce the over 200 sensor and actuator wires to only 24 communication and power wires. More importantly, only a single six conductor cable spans each joint of BLEEX, virtually eliminating the presence of unwanted joint torque due to cables being flexed.

The BLEEX sensors are read at the rate of 10 KHz and the control loop repeats at a fixed 2 KHz rate (control sampling time is 500 μsec). Testing of EXOLINK on BLEEX has shown that the network update time for a ten RIOM network, passing 140 bytes of data (all data needed by the controller), requires less than 20 μsec. This leaves 480 μsec out of the 500 μsec control loop to perform the control algorithm calculations on the ExoCPU. More details on the communication implementation can be found in [72] and [73].

Chapter 4

Sensitivity Amplification Control (SAC) for Powered Exoskeletons

The effectiveness of the lower extremity exoskeleton stems from the combined benefit of the human intellect provided by the pilot and the strength advantage offered by the exoskeleton; in other words, the human provides an intelligent control system for the exoskeleton while the exoskeleton actuators provide most of the strength necessary for walking. The control algorithm must ensure that the exoskeleton moves in concert with the pilot with minimal interaction force between the two.

The Sensitivity Amplification Control scheme that will be developed in this chapter needs no direct measurements from the pilot or the human-machine interface (e.g. no force sensors between the two); instead, the controller estimates, based on measurements from the exoskeleton only, how to move so that the pilot feels very little force. This control scheme, which has never before been applied to any robotic system, is an effective method of generating locomotion when the contact location between the pilot and the exoskeleton is unknown and unpredictable (i.e. the exoskeleton and the pilot are in contact in variety of places). This control method differs from compliance control methods employed for upper extremity exoskeletons [76], [62], and [63], and haptic systems [77], and [68] because it requires no force sensor between the wearer and the exoskeleton.

The basic principle for the control of BLEEX rests on the notion that the exoskeleton needs to shadow the wearer's voluntary and involuntary movements quickly, and without

delay. This requires a high level of sensitivity in response to all forces and torques on the exoskeleton, particularly the forces imposed by the pilot. Addressing this need involves a *direct conflict* with control science's goal of minimizing system sensitivity in the design of a closed loop feedback system. If fitted with a low sensitivity, the exoskeleton would not move in concert with its wearer. However, maximizing system sensitivity to external forces and torques leads to a loss of robustness in the system, so the trade-off must be evaluated.

Taking into account this new approach, the goal is to develop a control system for BLEEX with high sensitivity. This presents two realistic concerns. First, an exoskeleton with high sensitivity to external forces and torques would respond to external forces not initiated by its pilot. For example, if someone or something pushed against an exoskeleton that had high sensitivity, the exoskeleton would respond in the same manner that it would to forces from its pilot.

The fact that the exoskeleton does not stabilize its behavior on its own in response to other forces may sound like a serious problem. If it did attempt to stabilize itself (e.g. using a gyroscope and feedback loop), the pilot would *receive* motion from the exoskeleton unexpectedly and would have to struggle with it to avoid unwanted movement. The key to stabilizing the exoskeleton and preventing the pilot and exoskeleton from falling in response to external forces is the pilot's ability to move quickly (e.g. step back or sideways) to create a stable situation for herself and the exoskeleton. For this to occur, a very wide control bandwidth is needed so the exoskeleton can respond to both pilot's voluntary and involuntary movements (i.e. reflexes).

The second concern is that systems with high sensitivity to external forces and torques

are not robust to variations in the model parameters and therefore the precision of the system performance will be proportional to the precision of the exoskeleton dynamic model. Although this is a serious drawback, it is accepted as unavoidable for the time being. Chapter 5 will return to this issue after analyzing the performance of the SAC scheme on the exoskeleton hardware and a partial solution to this problem will be presented. Nevertheless, experimental systems in our laboratory have proved the overall effectiveness of the SAC method in unobtrusively shadowing the pilot's movement.

4.1 Simple 1 DOF system model

The development of the exoskeleton control is motivated here through use of the simple 1 DOF example shown in Fig. 4-1. This figure schematically depicts a human leg attached to and interacting with a 1 DOF exoskeleton leg in a configuration similar to the swing phase of the walking gait cycle (no interaction with the ground). For simplicity, the exoskeleton leg is shown as a rigid link pivoting about the hip joint. A linear actuator that can provide a torque about the joint is shown connected between ground and the exoskeleton leg. This setup also assumes that the upper body can react off of the fixed ground reference to generate forces and torques on the exoskeleton. Kinematic mismatch between the exoskeleton is not considered as the figure is intended only as a framework for understanding the control.

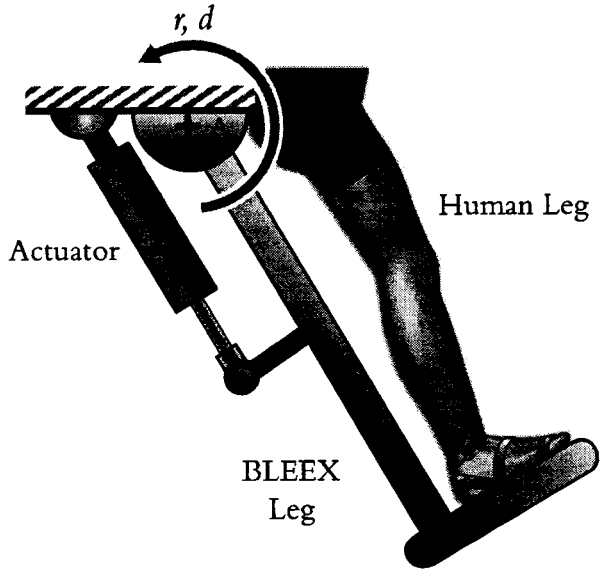


Fig. 4-1 Simple 1 DOF representation of an exoskeleton leg interacting with the pilot leg.

Although the pilot is attached securely to the exoskeleton at the foot, other parts of the pilot leg, such as the shanks and thighs, can contact the exoskeleton and impose forces and torques on the exoskeleton leg. The location of the contacts and the direction of the contact forces (and sometimes contact torques) vary and are therefore considered unknown values in this analysis. In fact, one of the primary objectives in designing BLEEX was to ensure a pilot's unrestricted interaction with BLEEX. The equivalent torque on the exoskeleton leg, resulting from the pilot's applied forces and torques, is represented by d.

4.2 Controller development

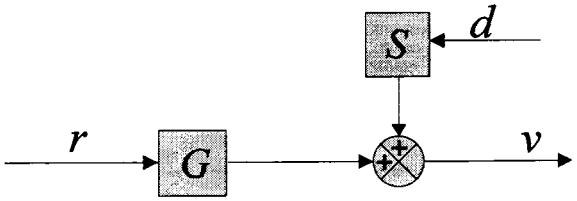


Fig. 4-2 Block diagram showing the exoskeleton angular velocity as a function of the input to the actuators and the torques imposed by the pilot onto the exoskeleton.

In the absence of gravity, the block diagram of Fig. 4-2 and equation (4.1) represent

the dynamic behavior of the exoskeleton leg regardless of any kind of internal feedback the actuator may have.

$$v = Gr + Sd \tag{4.1}$$

Where G represents the transfer function from the actuator input, r, to the exoskeleton angular velocity, v (actuator dynamics are included G). In the case where multiple actuators produce controlled torques on the system, r is the vector of torques imposed on the exoskeleton by the actuators. The form of G and the type of internal feedback for the actuator is immaterial to the discussion here. Also bear in mind the omission of the Laplace operator in all equations for the sake of compactness.

The exoskeleton velocity, as shown by (4.1), is affected by forces and torques from its pilot. The sensitivity transfer function S, represents how the equivalent human torque affects the exoskeleton angular velocity. S maps the equivalent pilot torque, d, onto the exoskeleton velocity, v. If the actuator already has some sort of primary stabilizing controller, the magnitude of S will be small and the exoskeleton will only have a small response to the imposed forces and torques from the pilot or any other source. For example, a high gain velocity controller in the actuator results in small S, and consequently a small exoskeleton response to forces and torques. Also, non-backdrivable actuators (e.g. large transmission ratios or servo-valves with overlapping spools) result in a small S which leads to a correspondingly small response to pilot forces and torques.

Note that d (resulting torque from pilot on the exoskeleton) is not an exogenous input; it is a function of the pilot dynamics and variables such as position and velocity of the pilot and the exoskeleton legs. These dynamics change from person-to-person, and within a person as a function of time and posture. These dynamics will be added to the

analysis in later paragraphs, but it is unrelated to the purpose of current discussion. The assumption is also made that d is only from the pilot and does not include any other external forces and torques.

The objective of exoskeleton control is to increase exoskeleton sensitivity to pilot forces and torques through feedback, but without measuring d. In other words, we are interested in creating a system that allows the pilot to swing the exoskeleton leg easily. Measuring d to create such systems develops several hard, but ultimately solvable problems in the control of a lower extremity exoskeleton. Some of those problems are briefly described below:

1) Depending on the architecture and the design of the exoskeleton, one needs to install several force and torque sensors to measure all forces from the pilot on the exoskeleton because the pilot is in contact with the exoskeleton at several locations. While the example in Fig. 4-1 indicates a single attachment between human foot and exoskeleton foot, the real device is designed to closely follow the contours of the leg in order to keep the overall profile as small as possible. This can result in intermittent and unpredictable contact along the entire length of the structure. These contact locations are typically not known in advance. For example, we have found that some pilots are interested in having compliant braces or straps connecting BLEEX and the human at the shanks, while others prefer having them on the thighs. Inclusion of sensors on a leg to measure all possible human forces and torques may result in a system suitable for a laboratory setting, but the complexity and the need to carefully fit each individual such that all sensors are properly aligned would result in a device not robust enough to be deployed in the field.

2) If the BLEEX design is such that the forces and torques applied by the pilot on the exoskeleton are limited to a specified location, (e.g., the pilot foot), the sensor that measures the pilot forces and torques will also inadvertently measure other forces and torques that are not intended for locomotion. This is a major difference between measuring forces from, for example, the human hands, and measuring forces from the human lower limbs. Using our hands, we are able to impose controlled forces and torques on upper extremity exoskeletons and haptic systems with very few uncertainties [78-80]. However, our lower limbs have other primary and non-voluntary functions like load support that take priority over locomotion. To give an example, if force sensors were placed along the length of the leg in order to capture data from many possible contact points, some would inadvertently pick up the persistent small oscillatory balance movements made unconsciously by the CNS‡ in the lower body [30]. Differentiating between these movements and user locomotion could become problematic.

3) One option we have experimented with is the installation of sensing devices for forces on the bottom of the pilot's boots, where they are connected to BLEEX. Since the force on the bottom of the pilot's boot travels from heel to toe during normal walking, several sensors are required to measure the pilot force. Ideally, we would have a matrix of force sensors between the pilot and exoskeleton feet to measure the pilot forces at all locations and at all directions, though in practice, only a few sensors could be accommodated in our testing: at the toe, ball, midfoot, and the heel. Still, this option leads to thick and bulky soles. Other research groups have also pursed force sensor systems for the shoe though none have created a device compact and robust enough for use on BLEEX [81, 82].

‡ Cental Nervous System

4) The bottoms of the human boots experience cyclic forces and torques during normal walking that lead to fatigue and eventual sensor failure if the sensor is not designed and isolated properly.

For the above reasons and the collective experience of the research group in the design of various lower extremity exoskeletons, it became evident that the existing state of technology in force sensing could not provide robust and repeatable measurement of the human lower limb force on the exoskeleton. The goal then shifted to developing an exoskeleton with a large sensitivity to forces and torques from the operator using measurements only from the exoskeleton (i.e. no sensors on the pilot or the exoskeleton interface with the pilot). Creating a feedback loop from the exoskeleton variables only, as shown in Fig. 4, the new closed-loop sensitivity transfer function is presented in (4.2).

$$S_{NEW} = \frac{v}{d} = \frac{S}{1+GC} \tag{4.2}$$

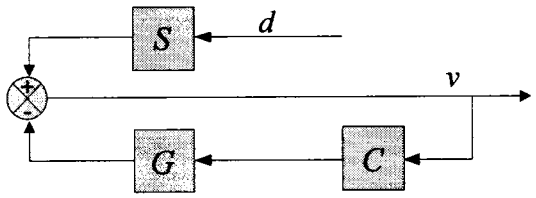


Fig. 4-3 Negative feedback loop added to the block diagram of Fig. 4-2. C is the controller operating only on the exoskeleton state variables.

Observation of (4.2) reveals that $S_{NEW} \leq S$, and therefore any negative feedback from the exoskeleton, leads to an even smaller sensitivity transfer function. With respect to (4.2) the exoskeleton design goal is to create a controller for a given S and G such that the closed loop response from d to v (the new sensitivity function as given by (4.2)) is greater than the open loop sensitivity transfer function (i.e. S) within some bounded frequency range. This design specification can be written in the form of inequality (4.3).

$$|S_{NEW}| > |S| \qquad \forall \omega \in (0, \omega_0) \tag{4.3}$$

or alternately:

$$|1 + GC| < 1 \qquad \forall \omega \in (0, \omega_0) \tag{4.4}$$

where ω_0 is the exoskeleton maneuvering bandwidth.

In classical and modern control theory, every effort is made to minimize the sensitivity transfer function of a system to external forces and torques. But for exoskeleton control, one requires a totally opposite goal: *maximize the sensitivity of the closed loop system to forces and torques.* In classical servo problems, negative feedback loops with large gains generally lead to small sensitivity within a bandwidth, which means that they reject forces and torques (usually called disturbances). However, the above analysis states that the exoskeleton controller needs a large sensitivity to forces and torques. From the perspective of the pilot, this has the effect making the exoskeleton feel and behave like a very small mass when the sensitivity of the closed loop system to forces and torques is high.

To achieve a large sensitivity function, we use the inverse of the exoskeleton dynamics as a positive feedback controller so that the loop gain for the exoskeleton approaches unity (slightly less than 1).

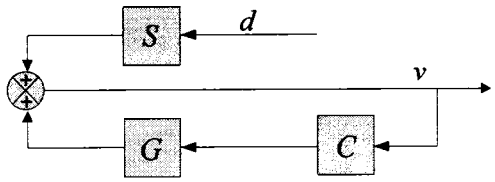


Fig. 4-4 Block diagram of exoskeleton with positive feedback loop.

Assuming positive feedback as shown in Fig. 4-4, the sensitivity transfer function from (4.2) can be written as

$$S_{NEW} = \frac{v}{d} = \frac{S}{1 - GC}. \tag{4.5}$$

If C is chosen to be the inverse of the plant dynamics multiplied by gain slightly less than 1, for example $C = 0.9G^{-1}$, then the new sensitivity transfer function is $S_{NEW} = 10S$ (ten times the force amplification). This positive feedback controller can be written more explicitly as

$$C = \left(1 - \alpha^{-1}\right) G^{-1}. \tag{4.6}$$

where α is an amplification factor greater than unity (for the above example, $\alpha = 10$ led to the choice of $C = 0.9G^{-1}$). Equation (4.6) simply states that a positive feedback controller needs to be chosen as the inverse dynamics of the system dynamics scaled down by $\left(1 - \alpha^{-1}\right)$. Note that (4.6) prescribes the controller in the absence of un-modeled high-frequency exoskeleton dynamics. In practice, C also includes a unity gain low pass filter to attenuate the un-modeled high-frequency exoskeleton dynamics.

The above method works well if the system model (i.e. G) is well-known to the designer. If the model is not well known, then the system performance will differ greatly from the one predicted by (4.5), and in some cases instability will occur. The above simple solution comes with an expensive price: robustness to parameter variations. In order to get the above method working, one needs to know the dynamics of the system well. The next section discusses this tradeoff.

4.3 Robustness to model parameter uncertainty

Taking variations of the new sensitivity transfer function when positive feedback is used gives

$$\frac{\Delta S_{NEW}}{S_{NEW}} = \frac{\Delta S}{S} + \frac{GC}{1 - GC} \cdot \frac{\Delta G}{G}. \tag{4.7}$$

If GC is close to unity (when the force amplification number, α, is large) any parameter variation on modeling will be amplified as well. For example if the parameter uncertainty in the system is 10%, i.e.:

$\left|\frac{\Delta G}{G}\right| = 0.10$ and $\left|\frac{\Delta S}{S}\right| = 0$, then (4.7) results in

$$\left|\frac{\Delta S_{NEW}}{S}\right| = \left|\frac{GC}{1-GC}\right| \cdot 0.10. \tag{4.8}$$

Now assume C is chosen such that $C = 0.9G^{-1}$. Substituting into (4.8) results in

$$\left|\frac{\Delta S_{NEW}}{S}\right| = 0.90. \tag{4.9}$$

Equation (4.9) indicates that any parameter variation directly affects the system behavior. In the above example, a 10% error in model parameters results in nine times the variation in the sensitivity function. This is why model accuracy is crucial to exoskeleton control.

To get the above method working properly, one needs to understand the dynamics of the exoskeleton quite well, as the controller is heavily model based. This problem can be seen as a tradeoff: the design approach described above requires no sensor (e.g. force or EMG) in the interface between the pilot and the exoskeleton; one can push and pull against the exoskeleton in any direction and at any location without measuring any variables on the interface. However, the control method requires a very good model of the system. Experiments with the actual BLEEX hardware have shown that this control scheme—which does not stabilize BLEEX—forces the exoskeleton to follow wide-bandwidth human maneuvers while carrying heavy loads.

4.4 Pilot dynamics

There are two approaches to human muscle modeling. One is based on the investigation of the molecular or fiber range of the muscle [83, 84], while the second is based on the relationship between the input and output properties of the muscle [85]. The second approach has been used previously in the design of upper extremity human power amplifier systems in [61, 76, 86].

In the Sensitivity Amplification Control scheme however, there is no need to include the internal components of the pilot limb model; the detailed dynamics of nerve conduction, muscle contraction and CNS processing are explicitly accounted for in constructing dynamic models of the pilot limbs. The pilot force on the exoskeleton, d, is a function of both the pilot dynamics, H, and the kinematics of the pilot limb (e.g., velocity, position or a combination thereof). In general, H is determined primarily by the physical properties of the human dynamics. Here H is assumed to be some nonlinear operator representing the pilot impedance as a function of the pilot kinematics.

$$d = -H(v) \tag{4.10}$$

The specific form of H is not known other than that it results in the human muscle force on the exoskeleton.

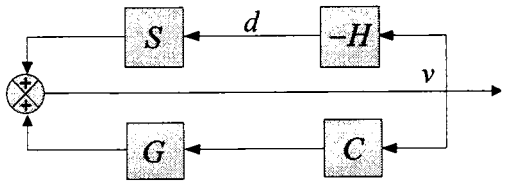


Fig. 4-5 Block diagram representing the overall behavior of the exoskeleton. The upper feedback loop shows how the pilot moves the exoskeleton through applied forces. The lower feedback loop shows how the controller drives the exoskeleton.

Fig. 4-5 represents the closed loop system behavior when pilot dynamics is added to the block diagram of Fig. 4-4. Examining Fig. 4-5 reveals that (4.5), representing the new exoskeleton sensitivity function from output d to input v, is not affected by the feedback loop containing H.

Fig. 4-5 shows an important characteristic for exoskeleton control. One can observe two feedback loops in the system. The upper feedback loop represents how forces and torques from the pilot affect the exoskeleton. The lower loop shows how the controlled feedback loop affects the exoskeleton. While the lower feedback loop is positive (potentially destabilizing), the upper feedback loop stabilizes the overall system of pilot and exoskeleton taken as a whole.

4.5 The Effect of pilot dynamics on closed loop stability

How does the pilot dynamic behavior affect the exoskeleton behavior? In order to get an understanding of the system behavior in the presence of pilot dynamics return to the 1 DOF system from Fig. 4-1 and assume H is a linear transfer function. The stability of the system shown in Fig. 4-5 is decided by the closed-loop characteristic equation:

$$1+SH-GC=0. \tag{4.11}$$

In the absence of a feedback controller, the pilot carries the entire load (payload plus the weight of the exoskeleton torso). The stability in this case is decided by the characteristic equation:

$$1+SH=0 \tag{4.12}$$

Characteristic equation (4.12) is always stable since it represents the coupled pilot and exoskeleton behavior without any controller (i.e., when $GC=0$). When feedback a loop

with controller C is added, the closed loop characteristic equation changes from (4.12) to (4.11), and the closed loop stability is guaranteed as long as inequality (4.13) is satisfied.

$$|GC| < |1 + SH| \qquad \forall \omega \in (0, \omega_0) \tag{4.13}$$

According to (4.6), C is chosen such that $|GC| < 1$ and therefore in the absence of uncertainties, (4.13) is guaranteed as long as $1 \leq |1 + SH|$. Unlike control methods utilized in the control of the upper extremity exoskeletons [62, 68] and [63], the human dynamics in the control method described here has little potential to destabilize the system. Even though the feedback loop containing C is positive, the feedback loop containing H stabilizes the overall system of pilot and exoskeleton.

The following example is put forward to illustrate this concept. For the 1 DOF system of Fig. 4-1, assume the human and machine leg have equivalent masses and inertias. The human sensitivity transfer function and the exoskeleton leg dynamics would both be the inertial effects of a simple rotating mass, $S = G = 1/Js$, where J is the inertia and s is the Laplace operator. For this example, assume the human impedance, H is modeled as a series connection of a spring and a damper, $H = M_H s + C_H$, where M_H is a positive damping and C_H is a positive spring rate. If the exoskeleton sensitivity amplification factor is $\alpha = 10$ and consequently the controller is chosen from (4.6) as $C = 0.9Js$, the new closed loop sensitivity transfer function with a positive feedback loop around the exoskeleton variables is

$$S_{NEW} = \frac{v}{d} = \frac{S}{1 - GC} = 10S. \tag{4.14}$$

The system characteristic equation when $C = 0$ (no control running) is given by

(4.15) and always results in a stable system because the roots of the characteristic equation will always be in the left half of the s-plane.

$$1+SH=\frac{(J+M_H)s+C_H}{Js} \tag{4.15}$$

When a positive feedback loop with the controller $C = 0.9Js$ is added, characteristic equation (4.16) of the overall system is still stable.

$$1+SH-GC=\frac{(0.1J+M_H)s+C_H}{Js} \tag{4.16}$$

Even if α is chosen as a very large number, the system in the absence of parameter uncertainties, is stable. Now suppose uncertainty is added to the model inertia, $\frac{\Delta J}{J}=-20\%$, i.e. $\frac{\Delta S}{S}=\frac{\Delta G}{G}=20\%$. The variation in new sensitivity function is,

$$\frac{\Delta S_{NEW}}{S_{NEW}}=\frac{\Delta S}{S}+\frac{GC}{1-GC}\cdot\frac{\Delta G}{G}=200\%. \tag{4.17}$$

In this case, $GC=\frac{1}{0.8Js}\cdot 0.9Js=\frac{9}{8}$, $S=\frac{1}{0.8Js}$, and the closed-loop characteristic polynomial is represented by

$$1+SH-GC=\frac{(10M_H-J)s+10C_H}{8Js}. \tag{4.18}$$

Equation (4.18) states that the system is unstable if $J>10M_H$. In an intuitive sense, this is saying that the system inertia is so great that the human is unable to provide enough damping, then the system would become unstable. Thus, with the Sensitivity Amplification Control scheme the system is vulnerable to model parameter uncertainties and the controller discussed here is stable when worn by the pilot as long as parameter

uncertainties are kept to a minimum. The definition of minimum for the parameter uncertainty is dependant on the unknown human dynamics. In practice, the controller can be run with the sensitivity amplification gain set to zero (with the operator supporting the weight of the system). The gain can then be increased gradually until the system feels comfortable and responsive to the wearer. Adding a safety overhead to the sensitivity amplification factor can be used to prevent the operator from running a marginally stable state. If the gain is set too high the exoskeleton joints can begin to vibrate from high frequency oscillations of the unstable system and the system can be temporarily disabled. If a lower than optimal value is chosen for the sensitivity amplification factor, the operator perceives proportionally more weight and inertia from the exoskeleton and payload. Choosing the sensitivity amplification factor in an automated fashion is a potential opportunity for future work.

Chapter 5

Application of SAC on BLEEX

Chapter 4 motivated the Sensitivity Amplification Control scheme using a 1 DOF system. BLEEX, as shown in Fig. 3-5, is a system with many degrees of freedom and therefore implementation of the SAC scheme on BLEEX needs further attention.

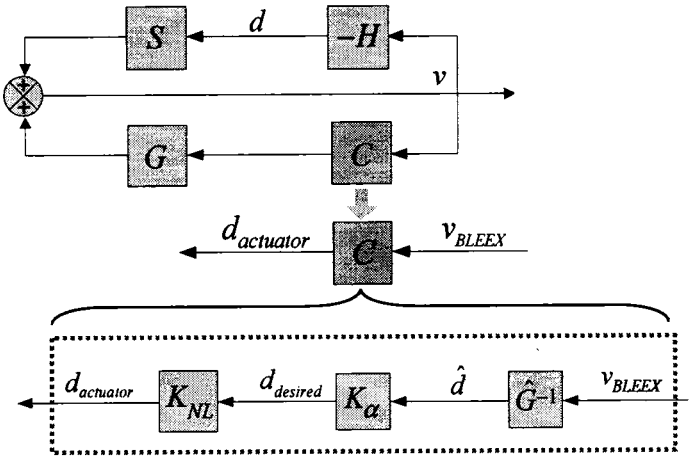


Fig. 5-1 BLEEX Sensitivity Amplification Controller expanded to show inverse dynamics, sensitivity amplification gain, and local control loop around an actuator.

Fig. 5-1 expands the controller block of Fig. 4-5 to show the three major internal components of C needed to implement the SAC scheme. The G^{-1} block represents the full inverse dynamics of the multi-DOF BLEEX robot. These equations take the instantaneous values of the state variables of the exoskeleton (position, velocity, and acceleration of each joint) as an input and output a vector of torques produced at each joint by the dynamics of the exoskeleton and the payload. The K_α block is the sensitivity amplification gain applied to the output of the dynamic equations. The K_{NL} block contains a local non-linear control loop that is applied to each individual actuator. The

SAC scheme generates a desired force or torque command and assumes the actuator can be simply commanded to produce the required force or torque. However, BLEEX uses linear hydraulic cylinders for actuation in which a servo valve controls the flow of fluid in and out of the cylinder. Therefore, a non-linear multiple sliding surface (MSS) feedback loop through local cylinder pressure is added to each actuator allowing it to be treated as a force generator. This local joint control was proposed and simulated in [20]. Results from the implementation and tuning of this controller on the actual BLEEX hardware will be briefly discussed.

The BLEEX controller structure

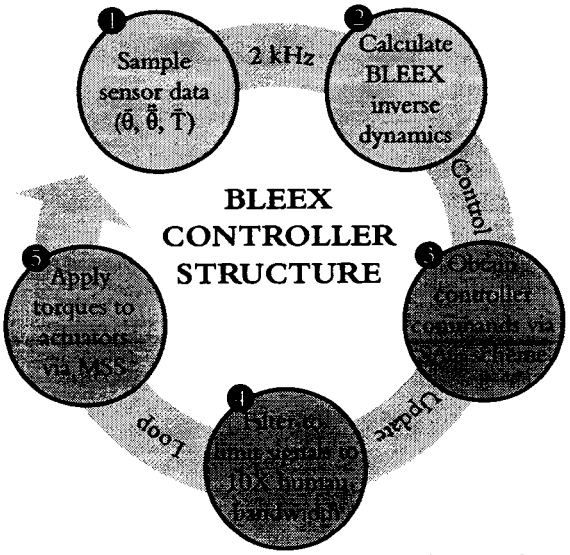


Fig. 5-2 Control loop showing major components as implemented in BLEEX Software.

Fig. 5-2 shows a graphical view of the structure of the SAC scheme as it has been implemented in the BLEEX software. The loop executes at a fixed 2kHz rate. The following steps are performed sequentially: 1) collect and scale data from all sensors to get values for the instantaneous $\theta,\dot{\theta},\ddot{\theta}$ at each joint, 2) calculate the full BLEEX inverse dynamics using $\theta,\dot{\theta},\ddot{\theta}$, 3) calculate using the SAC scheme the required torque to apply at each joint, 4) filter the command signals to eliminate high frequency effects of un-

modeled dynamics, and 5) finally send the desired joint torques to the local non-linear MSS controller that causes the hydraulic actuator to behave as a linear torque source. Additional components not shown include error checking and safety interlocks to ensure that BLEEX safely shuts down in the event of a sensor or hardware failure. Also, during each control loop cycle, the computer communicates with an optional Graphical User Interface (GUI) if it is present in the system (the GUI can be plugged into and removed from the controller while it is running without affecting performance) [73].

5.2 BLEEX dynamic equations

For simplicity in control BLEEX is considered to have three distinct phases in the gait cycle, (Fig. 5-3), which manifest to three different dynamic models:

Single support: one leg is in the stance phase while the other leg is in swing.

Double support: both legs are in the stance phase and situated flat on the ground.

Double support with one redundancy: both legs are in stance phase, but one foot is situated flat on the ground while the other one is not.

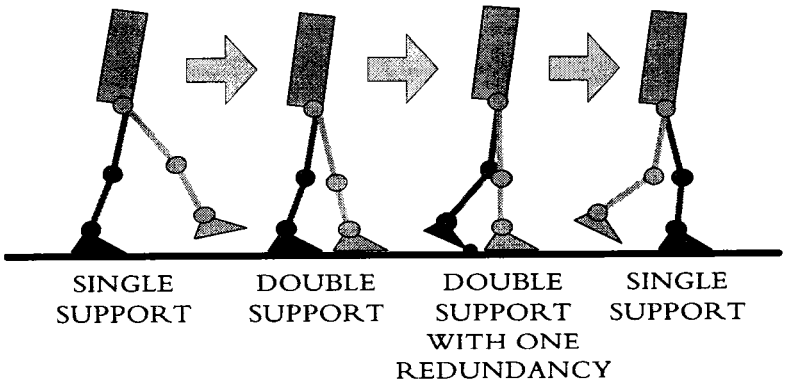


Fig. 5-3 Simplified gait cycle used as a basis for BLEEX dynamic equations

Using the information from the sensors in the foot sole discussed in chapter 3, the controller determines in which phase BLEEX is operating and which of the three dynamic

models apply.

Redundant double support stance phase

The double support with one redundancy phase refers to the redundant degree of freedom created when the exoskeleton foot is touching at the toe or heel. This distinction has implications in terms of the kinematic description of the system. The dynamic models for each phase of the gait cycle are based on a two dimensional sagittal plane only representation of the exoskeleton. The model of BLEEX is simplified to a series of rigid limb segments (foot, shank, thigh, torso) connected together by simple revolute joints. The dynamic equations for each model are a function of the number of segments connected together.

When the foot is flat on the ground its velocity and acceleration are both zero and consequently it can be dropped from the equations of motion. This creates a system with five segments and six degrees of freedom. When the foot is touching at the toe or heel, it can pivot with respect to ground. This results in a system with six segments and seven degrees of freedom. The two phases were separated into distinct models (rather than just model both as double support with one redundancy and let the terms multiplied by zero drop out in the math) because it was important to reduce the mathematical overhead for the embedded control computer whenever possible.

When one foot is touching the ground at the toe and another at the heel, BLEEX is considered to be in double support-one redundancy mode and the foot touching only on the heel is assumed to be flat on the ground. A fourth type of double support exists, which was discussed in the original presentation of the dynamics from [20] but not implemented with in the Sensitivity Amplification Controller for reliability and safety reasons: double

support with double redundancy. This is the case with both feet touching on the toe or both on the heel. For these cases, the two extra degrees of freedom added in the feet and the lack of any source of actuation between the toe and ground make it impossible to control the torque between BLEEX and the ground. When either of these cases is detected via the foot sensors, the exoskeleton uses double support model. This allows the human to stand on the toes or heels by applying the necessary additional torque at the ankle through her own muscles.

Abduction-adduction control

In the initial experiments with BLEEX, the control of the non-sagittal plane abduction-adduction DOF at the hip was decoupled from the control of joints in the sagittal plane. This was done to simplify the initial control task and was based on the observation through measurements that the abduction-adduction movements during normal walking (less than 0.9 m/s or 2 mph) are rather slow in comparison with the movements in the sagittal plane. The abduction-adduction movements are considered quasi-static maneuvers with little dynamic affects on the rest of system. This indicates that the exoskeleton dynamics in the sagittal plane are affected only by the abduction-adduction angle and not by the abduction-adduction dynamics. The following sections describe the control method in the sagittal plane for a given set of abduction-adduction angles. Performance of BLEEX with the addition of actively controlled abduction-adduction has been presented in [74].

Partitioned dynamic model of the BLEEX

The BLEEX inverse dynamics model was developed in the thesis of [20] and will be covered here briefly to establish the procedure for applying the Sensitivity Amplification Control technique. The dynamic model for walking is partitioned according to the

ground contact conditions shown in Fig. 5-3. Beginning with the swing state, the exoskeleton is supported by a single leg which can be represented as a seven DOF serial chain of links. The dynamics can be written in the form

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + P(\theta) = T + d \tag{5.1}$$

where $\theta = [\theta_1 \ \theta_2 \ldots \theta_7]^T$ and $T = [0 \ T_1 \ T_2 \ldots T_6]^T$.

M is a 7×7 inertia matrix and is a function of θ, $C(\theta,\dot{\theta})$ is a 7×7 centripetal and Coriolis matrix and is a function of θ and $\dot{\theta}$, and P is a 7×1 vector of gravitational torques and is a function of θ only. T is the 7×1 actuator torque vector with its first element set to zero since there is no actuator associated with joint angle θ_1 (i.e. angle between the BLEEX foot and the ground). d is the effective 7×1 torque vector imposed by the pilot on BLEEX at various locations.

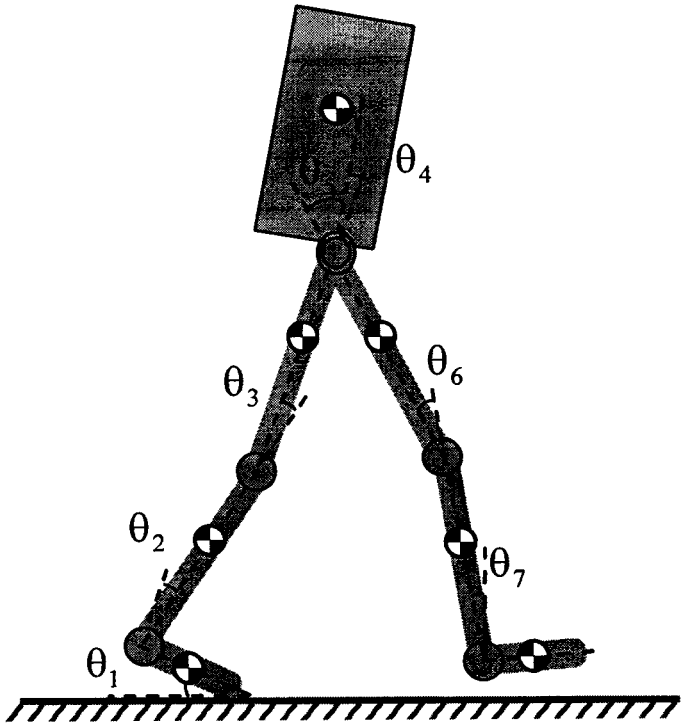


Fig. 5-4 Sagittal plane representation of BLEEX in the single support swing phase. The "torso" in the figure includes the combined exoskeleton torso, payload, control computer, and power source.

Lagrangian derivation of inverse dynamics

The equations are developed via Lagrange's method. Each limb segment has the following properties: mass m_n, inertia I_n, length L_n, distance to link CG along the segment from the its distal joint L_{Gn}, and distance to the segment CG perpendicular to the length of the segment h_{Gn}. $n \in (0 \ldots 7)$ and represents the numbered limb segments. Reference coordinate frames are first established on each limb segment and are of the form $\bar{e}_{ij} = [\bar{e}_{i1} \quad \bar{e}_{i2} \quad \bar{e}_{i3}]$ for the ith coordinate frame starting with in inertial coordinate frame $i = 0$, affixed to the ground.

Rotation matrices can then be written for each joint angle, $\theta_i = [\theta_1 \ldots \theta_7]$, to convert between reference frames and are of the form:

$$Q_{(i-1)i} = \begin{bmatrix} \cos\theta_i & -\sin\theta i & 0 \\ \sin\theta_i & \cos\theta_i & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{5.2}$$

A vector from frame $\vec{e}_i$ can be written in frame $\vec{e}_{(i-1)}$ as $\vec{e}_{(i-1)} = Q_{(i-1)i}\vec{e}_i$. Furthermore, rotation matrices can be combined to write vector $\vec{e}_n$ in the inertial frame as

$$\vec{e}_{0/n} = Q_{01}Q_{02}\cdots Q_{(n-1)n}\vec{e}_n. \tag{5.3}$$

The unit vectors are written as $\vec{e}_{ij/k}$ where i is the source frame, j is the unit vector, and k is the frame the unit vector is being written with respect to.

Lagrange's method requires writing the kinetic and potential energy for the system and then combining them using the Lagrange equation to produce the final set of inverse dynamic equations. The kinetic energy for BLEEX can be written as

$$KE = KE_1 + KE_2 + \ldots + KE_n. \tag{5.4}$$

And each individual limb segment kinetic energy can be written as

$$KE_n = \frac{1}{s} m_n \vec{V}_{Gn/0} \bullet \vec{V}_{Gn/0} + \frac{1}{2} I_n \omega_{n0} \bullet \omega_{n0}, \tag{5.5}$$

where $\vec{V}_{Gn/0}$ is the velocity of the CG of segment n written in inertial frame 0 and ω_{n0} is the angular velocity $\dot{\theta}_n$ of each segment with respect to the inertial frame. ω_{n0} can be found from

$$\omega_{n0} = \omega_{n(n-1)} + \omega_{(n-1)(n-2)} + \ldots + \omega_{(1)0}. \tag{5.6}$$

Limb segment CG velocities $\vec{V}_{Gn/0}$ can be found from

$$\vec{V}_{Gn/0} = \vec{V}_{n/0} + \vec{\omega}_{n0} \times \vec{r}_{0nG}, \tag{5.7}$$

where $\vec{V}_{n/0}$ is the velocity of the joint prior to the CG on the same limb segment. $\vec{r}_{0nG}$ is the vector from the inertial frame to the CG of limb segment n.

The potential energy for the system can be written as

$$PE = PE_1 + PE_2 + \ldots + PE_n \tag{5.8}$$

And the potential energy of each segment can be written as

$$PE_n = m_n g\left(\vec{r}_{0nG} \bullet \vec{e}_{02}\right) \tag{5.9}$$

Where g is acceleration due to gravity and the dot product of the vector from the inertial frame to the limb CG with the inertial frame unit vector in the vertical direction gives the vertical height of the limb CG with respect to the inertial frame.

The Lagrangian is defined as

$$L = KE - PE \tag{5.10}$$

And the Lagrangian equation can now be used to assemble the complete dynamic equations for each joint i.

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = T_i + d_i \tag{5.11}$$

Where T_i is the ith joint torque and d_i is the unknown torque that is applied by the human. Equation (5.11) can then be calculated for each joint in BLEEX. The combination of the rotation matrix, the vector cross product for velocity, and the partial derivatives taken in the Lagrangian equation causes the dynamic terms to be become extremely large as n gets larger than 2 or 3 limb segments. In practice, these terms are

almost never expanded by hand for complex systems and the process is usually automated with a symbolic mathematical solver program such as Matlab[4] or Mathematica[5]. The full dynamic equations and Matlab scripts for generating them have been published in [20]. The result of the expanded Lagrange equation can be re-arranged in the form of (5.1), which is repeated for reference: $\vec{M}(\theta)\ddot{\theta} + \vec{C}(\theta,\dot{\theta})\dot{\theta} + \vec{P}(\theta) = \vec{T} + \vec{d}$.

Dynamic equations for the double support phases

The above derivation was for the single support phase where only one foot is touching the ground. The equations for this, though tedious, are straightforward to implement because each limb segment is connected in series. For the double support cases where both feet are flat on the ground (double support) and when one foot is flat on the ground and one is touching by only the toe or heel (double support with one redundancy), the exoskeleton becomes a parallel kinematic linkage with a closed kinematic chain. Closed kinematic chains pose a challenge. The additional constraint created when a linkage is connected to a reference point in multiple locations means that in the dynamic equations there will be one more unknown variable than the number of independent equations. There are several techniques for solving the inverse dynamics of parallel kinematic chains: obtain a measure of the force or torque associated with the additional constraint using a additional sensors, solve the complete system explicitly using Newton's method with an added geometry based constraint equation [87-89], or partition the system and make an assumption about the behavior of the partitioned system [90]. We chose the latter technique for reasons that will be discussed shortly. Adding an additional sensor to the exoskeleton was not pursued due to the added complexity and the desire for the project as a whole to minimize complexity. The second technique was not pursued both because of

4 Matlab® is a scientific computing software package from The MathWorks Corporation. http://www.mathworks.com/
5 Mathematica is a scientific computing software package from Wolfram Research Inc. http://www.wolfram.com/

the extremely complicated derivation that would result (which would be more prone to contain errors) and because this technique assumes the kinematic chain is permanently affixed to the base frame (which is true for typical parallel robot). Though BLEEX is standing on the ground, there is a potential that the feet could slip and so a geometry constraint equation could produce unreliable results. In addition, there are other more advanced techniques discussed in the literature that were not pursued. These include recursive techniques [91], the method of virtual work [92-94], and other techniques [95-98].

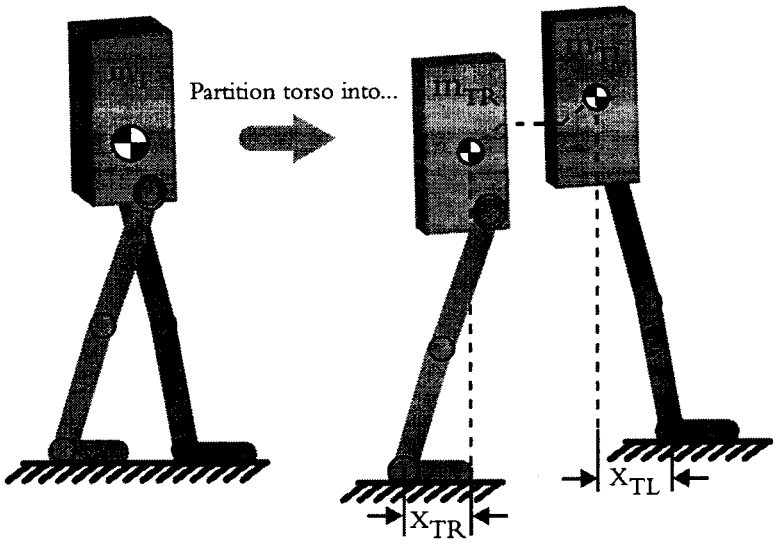


Fig. 5-5 Partitioning of double stance BLEEX at the torso. The torso mass is split into left (m_{TL}) and right (m_{TR}) components. Also, the horizontal distances of the half torso CGs from the ankle joint are indicated.

The partitioning approach divides the double stance configuration of the exoskeleton at the torso into a left half and a right half as shown in Fig. 5-5. The dynamic equations are then formulated independently for each half of the exoskeleton in the same manner as for the stance configuration. The difference is that now, each half-body dynamic model is only composed of four segments: foot, shank, thigh, and half-torso. The only additional step is to distribute the torso mass between the two halves of the system. The

contributions of the torso mass, m_T on each leg (i.e., m_{TL} and m_{TR}) are chosen as functions of the location of the torso center of mass relative to the locations of the exoskeleton ankles such that:

$$m_T = m_{TL} + m_{TR} \tag{5.12}$$

$$\frac{m_{TL}}{m_{TR}} = \frac{x_{TL}}{x_{TR}} \tag{5.13}$$

x_{TL} and x_{TR} are the horizontal distances in the sagittal plane between the torso CG and the left and right BLEEX ankles respectively. This distribution causes the stance leg at heel-strike to gradually accept the load of the torso as the CG moves forward over it. At the same time, the load is transitioned off of the opposite leg at the end of stance in a smooth manner, allowing the person to toe-off into swing phase without resistance.

(5.13) is valid only for quasi-static conditions, where the accelerations and velocities are small. This is in fact the case, since in the double support phase, both legs are on the ground and BLEEX's angular acceleration and velocities are small in comparison to those seen in swing. Creating a load distribution scheme that factored in dynamic effects was discussed in [20]. A proportional gain term related to the net force on the upper body was added to a variation of (5.13) such that the leg with the greatest mechanical leverage would receive a larger portion of the partitioned upper body mass. This was implemented on BLEEX and tested while walking at moderate speeds (0.1-1.3 m/s). The dynamic term caused an uncomfortable sensation for the operator in which rapid movement would cause the exoskeleton to shift its load distribution between the left and right legs. The load distribution scheme was then simplified to (5.13).

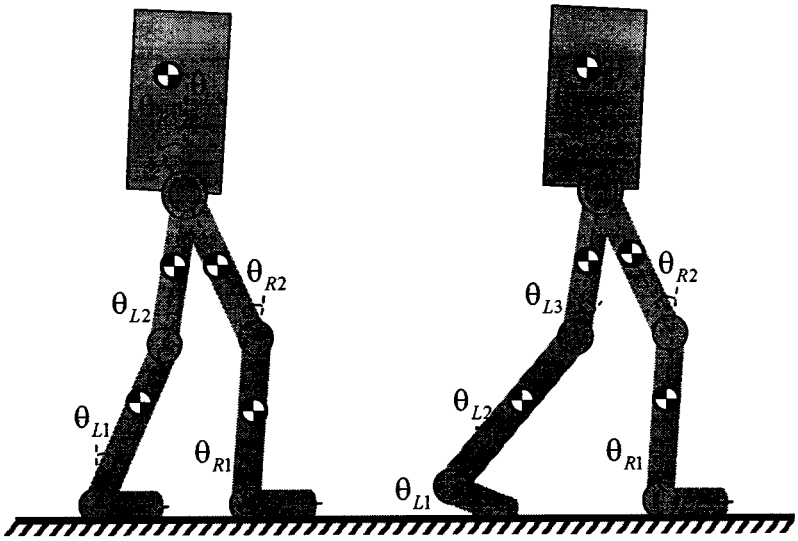


Fig. 5-6 Sagittal plane representation of BLEEX in the double support (left) and double support with one redundancy (right) configurations.

Following the same Lagrangian procedure from single stance, the dynamic equations for the two halves of the partitioned model can be written as,

$$M_L(m_{TL},\theta_L)\ddot{\theta}_L + C_L(m_{TL},\dot{\theta}_L,\theta_L)\dot{\theta}_L + P_L(m_{TL},\theta_L) = \vec{T}_L + \vec{d}_L \qquad (5.14)$$

$$M_R(m_{TR},\theta_R)\ddot{\theta}_R + C_R(m_{TR},\dot{\theta}_R,\theta_R)\dot{\theta}_R + P_R(m_{TR},\theta_R) = \vec{T}_R + \vec{d}_R \qquad (5.15)$$

Where $\theta_L = [\theta_{L1} \quad \theta_{L2} \quad \theta_{L3}]^T$ and $\theta_R = [\theta_{R1} \quad \theta_{R2} \quad \theta_{R3}]^T$. $\vec{T}_L$ and $\vec{T}_R$ are the actuator torques associated with the joints on the left and right legs of the exoskeleton. $\vec{d}_L$ and $\vec{d}_R$ are the unknown torques applied by the human to left and right leg joints.

5.3 Implementation of the SAC on the exoskeleton

The dynamic equations for the model of each gait phase form the G^{-1} block from Fig. 4-5. For each joint, the measured angle, angular velocity, and angular acceleration are fed into appropriate joint equation to calculate the torque at the joint produced by the weight of the exo, payload, and the dynamic torques resulting from their motion. The control law then directs the actuators to apply an appropriate torque to cancel out the load

induced torques. The result is that the exoskeleton responds to any additional torques generated by the human as if it had very little mass and inertia – in other words, it feels to the pilot that the exo follows her motion with very minimal interaction torques.

Swing phase SAC

According to (4.6), the controller is chosen to be the inverse of the BLEEX dynamics scaled by $\left(1-\alpha^{-1}\right)$, where α is the sensitivity amplification factor.

$$\bar{T} = \hat{P}(\theta) + (1-\alpha^{-1})\left[\hat{M}(\theta)\ddot{\theta} + \hat{C}(\theta,\dot{\theta})\dot{\theta}\right] \tag{5.16}$$

$\hat{C}(\theta,\dot{\theta})$, $\hat{P}(\theta)$, and $\hat{M}(\theta)$ are the estimates of the Coriolis matrix, gravity induced torque vector, and the inertia matrix respectively for the system as shown in Fig. 5-4. Equation (5.16) results in a 7×1 actuator torque. Since there is no toe actuator between the BLEEX foot and the ground for the first link in the chain, the torque prescribed by the first element of $\bar{T}$ must be provided externally (by the operator).

Toe torque specified by the dynamic equations

In reality two possibilities exist. If the toe torque is positive, it is counteracted by the presence of the ground under the foot. If the toe torque is negative, it indicates that the exoskeleton is in an unstable configuration and is trying to tip forward over the toe. Experience testing this condition in the laboratory has shown that the operator can readily detect the sensation that the load is tipping forward using the same unconscious balance mechanisms of the CNS that allow a person to maintain posture when not attached to an exoskeleton. As the operator leans forward into a configuration where the weight begins to tip the exoskeleton forward (and a negative torque is generated by the dynamic equations for the toe) the operator gets the sensation that the exoskeleton is lifting up on

the bottom of her foot. It is then very easy to correct the unbalance by straightening or leaning back slightly. The compliance of the exoskeleton foot and the large contact patch created by the compliance when the operator is standing on her toes provides the explanation for the separation between the time when the operator first feels the sensation of the exoskeleton tipping forward and the point at which the load line of the exoskeleton is in front of the feet and a fall is unavoidable.

Fig. 5-7 attempts to illustrate these two possibilities for handling the toe torque. In Fig. 5-7-A, the torque specified by the dynamic equations for the toe is positive (note that there is no actuator at the toe to provide this torque). The positive torque is balanced by the reaction force from the ground and the moment arm created by the length of the foot. In Fig. 5-7-B, the torque specified by the dynamic equations is positive, indicating that the exoskeleton is tipping forward due to the location of its CG or its current dynamic state. The exoskeleton does not immediately force the human to fall forward when this torque becomes negative because the compliance of the BLEEX foot creates a large toe contact patch with the ground. This allows the human to exert a balancing force using the toe muscles in the same way that a human balances when not wearing an exoskeleton.

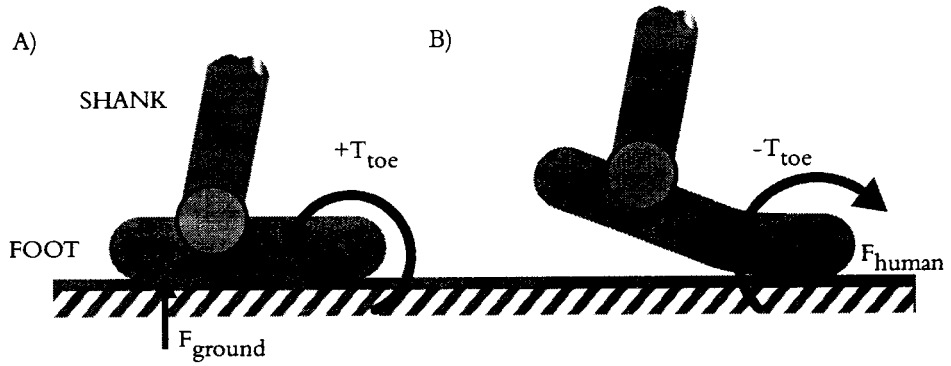


Fig. 5-7 Two possibilities for toe torque from the dynamic equations: A) shows the positive torque specified by the dynamic equations and the ground reaction force that balances it, B) shows the case of a negative torque at the toe, indicating that the exoskeleton is tipping forwards. Small values of negative toe torque can be compensated for by muscles in the human toe, which generate F_{human}.

The magnitude of the torque the human can generate is small, though the sensation of having to apply this torque to keep from falling provides the proprioceptive feedback to maintain the overall balance of herself and the exoskeleton. As a safety precaution, the actuator torques in the other exoskeleton joints are reduced when the torque crosses a negative threshold to provide a clear indicator to the human that she is in an unstable position. Experiments in the laboratory have shown that a threshold of -5 Nm at the toe provides adequate proprioceptive feedback for balance without jeopardizing the safety of the operator.

Result of Sensitivity Amplification Controller feedback – single support

Substituting $\bar{T}$ from (5.16) into (5.1) yields,

$$M(\theta)\ddot{\theta} + C(\theta,\dot{\theta})\dot{\theta} + P(\theta) = \hat{P}(\theta) + (1-\alpha^{-1})\left[\hat{M}(\theta)\ddot{\theta} + \hat{C}(\theta,\dot{\theta})\dot{\theta}\right] + d \qquad (5.17)$$

In the limit when $M(\theta) = \hat{M}(\theta)$, $C(\theta,\dot{\theta}) = \hat{C}(\theta,\dot{\theta})$, $P(\theta) = \hat{P}(\theta)$, and α is sufficiently large, the torques between the human and the exoskeleton, $\bar{d}$ will approach zero, meaning the pilot can walk as if BLEEX did not exist. However, it can be seen from (5.17) that the force felt by the pilot is a function of α and the accuracy of the estimates

$\hat{C}(\theta,\dot{\theta})$, $\hat{P}(\theta)$, and $\hat{M}(\theta)$. As was shown in chapter 4, small errors in the dynamic model can lead to large errors and instability in the controller. In general, the more accurately the system is modeled, the less the force between the human and exoskeleton, d, will be. In the presence of variations in abduction-adduction angles, only $P(\theta)$ in equations (5.1) and (5.16) needs to be modified.

Result of Sensitivity Amplification Controller feedback – double support

Similar to the single stance phase, the controllers are chosen such that,

$$\vec{T}_L = \hat{P}_L(m_{TL},\theta_L)+(1-\alpha^{-1})\left[\hat{M}_L(m_{TL},\theta_L)\ddot{\theta}_L+\hat{C}_L(m_{TL},\theta_L,\dot{\theta}_L)\dot{\theta}_L\right] \quad (5.18)$$

$$\vec{T}_R = \hat{P}_R(m_{TR},\theta_R)+(1-\alpha^{-1})\left[\hat{M}_R(m_{TR},\theta_R)\ddot{\theta}_R+\hat{C}_R(m_{TR},\theta_R,\dot{\theta}_R)\dot{\theta}_R\right] \quad (5.19)$$

Substituting $\vec{T}_L$ from (5.18) into (5.14) yields

$$\begin{aligned} &M_L\left(m_{TL},\theta_L\right)\ddot{\theta}_L+C_L\left(m_{TL},\dot{\theta}_L,\theta_L\right)\dot{\theta}_L+P_L\left(m_{TL},\theta_L\right) \\ &=\hat{P}_L(m_{TL},\theta_L)+(1-\alpha^{-1})\left[\hat{M}_L(m_{TL},\theta_L)\ddot{\theta}_L+\hat{C}_L(m_{TL},\theta_L,\dot{\theta}_L)\dot{\theta}_L\right]+\vec{d}_L \end{aligned} \quad (5.20)$$

Substituting $\vec{T}_R$ from (5.19) into (5.15) yields

$$\begin{aligned} &M_R\left(m_{TR},\theta_R\right)\ddot{\theta}_R+C_R\left(m_{TR},\dot{\theta}_R,\theta_R\right)\dot{\theta}_R+P_R\left(m_{TR},\theta_R\right) \\ &=\hat{P}_R(m_{TR},\theta_R)+(1-\alpha^{-1})\left[\hat{M}_R(m_{TR},\theta_R)\ddot{\theta}_R+\hat{C}_R(m_{TR},\theta_R,\dot{\theta}_R)\dot{\theta}_R\right]+\vec{d}_R \end{aligned} \quad (5.21)$$

As with the single stance phase, in the limit when $M(\theta)=\hat{M}(\theta)$, $C(\theta,\dot{\theta})=\hat{C}(\theta,\dot{\theta})$, $P(\theta)=\hat{P}(\theta)$, and α is sufficiently large, the torques between the human and the exoskeleton, $\vec{d}$ will approach zero.

Choosing the sensitivity amplification factor

As was discussed in Chapter 4, the sensitivity amplification factor, α, determines the proportionality between the human input forces and torques and the response of the exoskeleton to these torques. For example, a choice of $\alpha = 10$ would mean that BLEEX would respond to an input human torque of 1 Nm *as if the human had just applied* 10 Nm. In application on the exo, it was discovered through testing to be advantageous to split α into independent amplification gains for the static and dynamic components of the BLEEX equations of motion. This allowed a portion of the static gravity load of the exoskeleton to be intentionally applied to the person's body. For example, the wearer could be made to feel 5 kg out of the total 75 kg weight of the exo and payload. Applying a portion of the static exo load intentionally to the wearer's body (through the compliant upper body vest) provides the wearer with a comfortable level of proprioceptive feedback about the movement of BLEEX and the location of the BLEEX CG without adding undue strain to the wearer. All pilots who tested BLEEX in the laboratory indicated that having this feedback about the static load of BLEEX made the sensation of walking with BLEEX significantly more comfortable. In terms of the control law, this would modify (5.16) to be,

$$\vec{T} = \left(1-\alpha_1^{-1}\right)\hat{P}(\theta) + \left(1-\alpha_2^{-1}\right)\left[\hat{M}(\theta)\ddot{\theta} + \hat{C}(\theta,\dot{\theta})\dot{\theta}\right]. \tag{5.22}$$

In equation (5.22), α_1 is the sensitivity amplification gain associated with the static gravity induced forces and torques generated by BLEEX. α_2 is the sensitivity amplification gain associated with the dynamic forces and torques generated by the acceleration and velocity of the BLEEX and the payload. While the controller was stable for α_1 set infinitely large

(i.e. the term $\left(1-\alpha_1^{-1}\right)=1$), α_1 was typically chosen to be 10~20, resulting in 5~10 kg of the static weight of BLEEX being supported by the wearer. The magnitude of the dynamic terms component of the sensitivity amplification gain, α_2, was limited by the ability of the damping in the human feedback loop to stabilize the overall human-machine system. Through experimentation in the laboratory, a value of $3 \leq \alpha_2 \leq 10$ provided a safe margin of stability given the accuracy of the dynamic model while still feeling responsive and non-fatiguing to the wearer when walking at a moderate pace of 1.3 m/sec.

Dynamic model transitions

The BLEEX control software monitors the status of the foot sensors, chooses the appropriate dynamic model, and calculates the desired actuator torque commands in each iteration of the control loop. The control loop runs at 2KHz, therefore the controller can adapt to a change in the gait phase every 500μsec. No constraint is placed on the dynamic models to force the commanded torque outputs for each joint to align between transitions.

For example, if a person wearing the exo is standing with both feet side-by-side, the controller would use the double support equations and divide the weight of the BLEEX payload and structure evenly between both feet. If the human suddenly lifts one foot, the controller would switch to using the single support equations and the command torque would change dramatically. The joints of the leg in air, which now only needs to supply enough torque to support the weight of the BLEEX leg, would decrease. The other stance leg would suddenly have to begin supporting the full weight of the payload and its torques would double.

The controller is designed to adapt to sudden changes provided they occur within the human motion bandwidth, which is typically up to a maximum of 10Hz for fast reflex

actions (due to human limb inertia, lower limb reflexes movement is on the order of 2-5Hz) [30]. Relatively little information on the model transition problem for human exoskeletons is available in the literature. In the field of bipedal robots (no humans), there are many theories for handling the transition between multiple dynamic models and a review can found in [99] and [100]. A second order low-pass filter has been added to the command torques to attenuate frequencies above this range. In the continuous time domain, the filter is of the form:

$$\frac{T_{output}}{T_{input}} = \frac{1}{(1+s\beta)^2} \tag{5.23}$$

This has been implemented in discrete time on the BLEEX control software as in Direct Form II Transposed filter implementation using an infinite impulse response difference equation of the form:

$$T_{output} = -b_1 T_{filtered(n)} - b_2 T_{filtered(n-1)} + a_3 T_{(n-2)} + a_2 T_{(n-1)} + a_1 T_{(n)} \tag{5.24}$$

The coefficients for the filter were chosen using the Matlab Filter Design Toolbox and are given in Table 5-1 for a selection of the cutoff frequencies that felt comfortable for different individuals during testing. In terms of the actual feel of the device for the operator, lower cut-off frequencies add a slight sensation of sluggishness to movement while walking. Higher cut-off frequencies cause the exo to feel very responsive and effortless to move in at the cost of the sensation of a brief (10~20 msec) high frequency vibration at each gait phase transition. The frequencies in Table 5-1 represent a compromise that felt comfortable to most operators.

Table 5-1

Low-pass Cutoff Frequency	Coefficients for Low-Pass Filter in Equation (5.24)				
	a_1	a_2	a_3	b_1	b_2
10 Hz	2.4135e-4	4.8271e-4	2.4413e-4	-1.95557	9.5654e-1
20 Hz	9.4469e-4	1.8893e-3	9.4469e-4	-1.91119	9.1497e-1
50 Hz	5.5427e-3	1.1085e-3	5.5427e-3	-1.77863	8.0080e-1
80 Hz	1.3359e-3	2.6718e-2	1.3359e-2	-1.64748	7.0089e-1

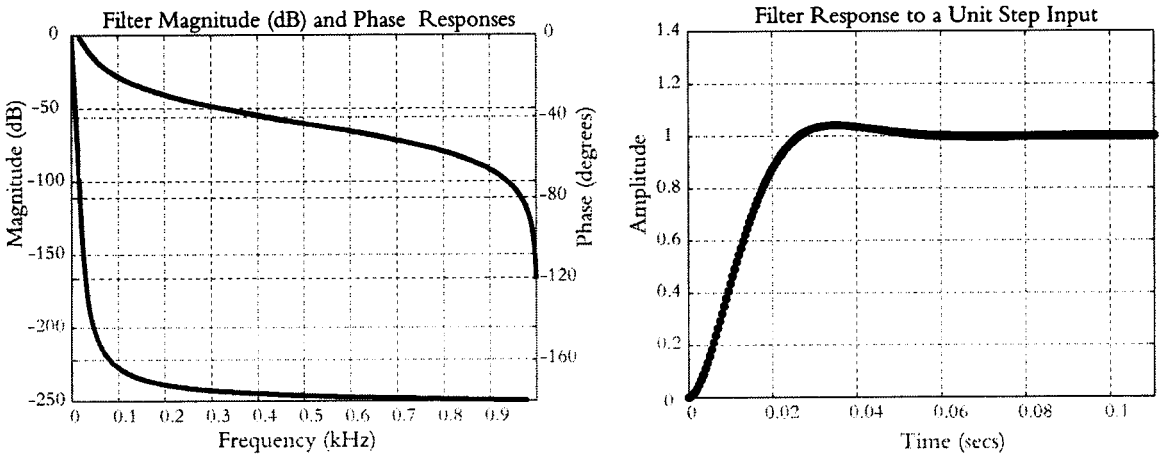


Fig. 5-8 Bode plot and step response for second order low-pass filters used to smooth model transitions.

Local torque control of the hydraulic actuators

As was mentioned in the introduction to this chapter, the SAC scheme calculates a desired actuator torque command each cycle through the control loop. The last stage of the control block in Fig. 5-1 is the non-linear actuator controller, labeled K_{NL} that was developed in [20] and subsequently tuned as part of this thesis work to achieve acceptable performance on the BLEEX hardware. This controller runs independently (though synchronously) from the SAC scheme and it allows the hydraulic actuator (an inherently flow controlled device) to be treated as a linear torque source with unity gain. The SAC controller can output an arbitrary torque command within the capabilities of the BLEEX hardware and power supply, and the non-linear joint controller ensures that this torque is applied to the joint within the bandwidth of human motion.

The actuators used on BLEEX are double acting hydraulic cylinders (fluid can be added to either side of the piston) that are coupled to high precision aerospace 3-way servo valves with carefully ground non-overlapping spools that offer high bandwidth flow control of the pressurized hydraulic fluid. The detailed analysis and selection of hydraulic components is covered in [31]. The command voltage sent to the servo valve from the BLEEX control computer varies the cross sectional area of a set of flow restriction orifices in the servo valve. This allows pressurized hydraulic fluid to flow in one side of the cylinder and out of the other to a low pressure reservoir. By applying an opposite polarity of voltage to the servo valve, the pressurized hydraulic fluid can be made to flow into the opposite side of the cylinder, reversing the supply and exhaust ports. The result is that the actuator can apply a bi-directional force. The linkage connecting it across a joint in the BLEEX converts this force into a torque at the joint.

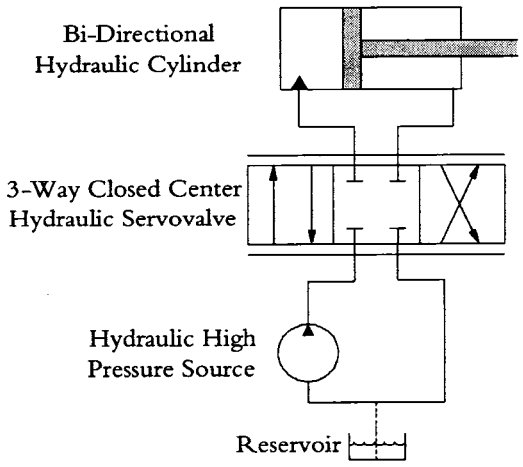


Fig. 5-9 BLEEX hydraulic actuation system.

The controller used for the joint was a multiple sliding surface non-linear controller. In [20], many options for achieving high-bandwidth force control of the hydraulic cylinders were analyzed, including linear control, feedback linearization, linear control with a servo valve spool observer, sliding mode control, multiple sliding surface control and adaptive multiple sliding surface control. The multiple sliding surface controller was selected and the simulated performance tracking performance in Fig. 5-10-A was presented.

Multiple Sliding Surface (MSS) control is an extension of sliding mode control (also known as variable structure control) that was first proposed by [101] and is a form of high speed switching control. Conceptually, a mathematical surface is defined such that, when the chosen system variables are on the surface, the system behaves as one would desire. A discontinuous switching law forces the system to move back towards the surface when the system parameters move outside a specified error bound. With proper choice of the surface definition and with an infinitely fast control loop, [101, 102] proved that perfect tracking can be achieved in spite of the system nonlinearities. In essence, this control strategy replaces the difficult to control non-linear dynamics of a system with a simple control law and relatively simple dynamics. MSS is an extension to sliding mode control to deal with systems with mismatched uncertainties and difficult to differentiate terms.

As part of my thesis work, the gains for the multiple sliding surfaces were experimentally tuned to achieve performance similar to the simulation presented in [20]. The result of this tuning is shown in the torque tracking results recorded on the actual BLEEX hardware (Fig. 5-10-B). The performance of the MSS controller has been sufficient to enable the tacking of high bandwidth (3-10Hz) torque commands necessary for the successful implementation of the global SAC scheme.

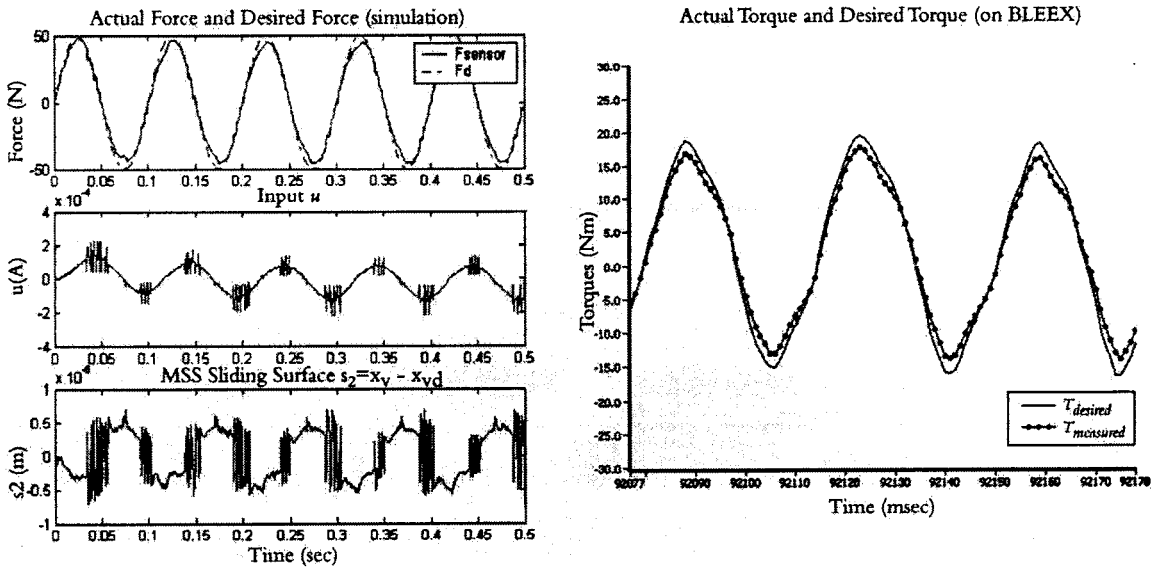


Fig. 5-10 Comparison of simulated (A) and actual (B) tracking performance. (A) is simulation data from [20] and (B) is the tracking on the actual BLEEX hardware after tuning the MSS controller.

5.4 Performance analysis of the SAC

Unlike traditional robotics projects in which a device can be built and characterized in a structured environment free of human intervention, BLEEX must be attached to a person in order to function. In fact, the controller mandates that the human be part of the control loop in order to ensure that the overall system is stable. The most straightforward performance measure for the system was the overall comfort and sensation as reported by the operator while wearing the device. Though subjective, this form of testing was well suited to evaluating subtle changes to the parameters of the control software.

Because the device was designed never to exceed the human range of motion, force, or natural limb velocity, it was also relatively safe to make changes to the control system and test them directly as the operator. One of the major difficulties of this process was getting the test subject to recognize and accurately describe the sensations of walking in the device. Often the most reliable way to get feedback on a particular change to the controller was to act as both programmer and test subject in BLEEX. Later in the project,

a formal test protocol was developed and submitted for approval to the U.C. Berkeley panel on human testing (see Appendix B). As of this time, this testing has been transferred to an independent testing laboratory at the U.S. Army Natick Soldier Training Center in Natick, MD and trials have been deferred in order to test a second generation of the exoskeleton hardware currently in development.

BLEEX Testing Setup

Testing of the 1st generation BLEEX shown in Fig. 1-3, was performed both in the Berkeley Robotics and Human Engineering laboratory in Etcheverry Hall on the U.C. Berkeley campus. Walking and other maneuverability testing was performed on a treadmill that offers programmable speed, and inclination as well as in the open floor space of the laboratory. In this location, BLEEX was powered by a DC motor version of the backpack hydraulic power supply connected via a power tether to the building's 240V mains. An overhead safety tether was attached to BLEEX at all times to ensure that the operator could never fall to the ground. The safety tether was a lightweight nylon strap (similar to seatbelt material) that was attached to a set of overhead rails with ball-bearing trucks, allowing it to be maneuvered with little force anywhere in an approximately 4.5m square area of the laboratory. The cable was kept sufficiently loose during testing such that it did not impart any measurable forces on the operator and was set to a length that allowed the operator to bend over and squat without obstruction.

Additionally, testing was performed using an autonomous gasoline IC engine based hydraulic power supply in a remote facility that offers a large unobstructed covered concrete area (10m x 12m) for untethered walking and unstructured maneuvers. Again, an overhead safety tether on a movable rail was used to ensure that the operator could not fall and hit the ground in the event of a loss of balance or a failure on the exoskeleton

BLEEX has been tested with up to 35kg of payload, however for the full energetically autonomous tests, the backpack power supply accounted for 25kg of the total payload. At 35kg, BLEEX performance is limited by the flow and pressure capability of its backpack power supply. Preliminary results of an advanced version of the power supply under development in this research group have been reported in [70]. This improved design would occupy less volume than the current design and could increase the payload to 50kg.

Continuous level-ground walking

Overall, the SAC scheme provided the first stable and comfortable level ground walking in the Berkeley Lower extremity exoskeleton. The smoothness and responsiveness of the SAC scheme, for the first time, allowed the operator to balance naturally without the need for hand support and without feeling the weight of the payload. Also, the SAC scheme allowed the operator to turn, squat, start and stop motion, and balance comfortably with up to 35kg of payload and without the need for any external support or balance assist devices. A maximum walking speed of 1.4 m/sec was achieved. Walking speed was limited by the maximum hydraulic flow rate of the backpack power supply [70]. For maneuvers such as squatting, the required torques were beyond the actuator limits for the knee, causing the wearer to have to provide additional torque to stand from a squatting position.

Deviations between predicted and measures torques

BLEEX was designed under the assumption that a device with similar mass and inertia parameters to a human, moving with the same motion (the walking gait cycle), should experience similar joint dynamics (velocity, acceleration, and torque). This allowed the biomechanics CGA data to be used as templates for the BLEEX motion and actuation requirements. To test this assumption, representative data from an operator walking

continuously at a moderate pace (1.3 m/s) on a treadmill are presented along with joint CGA data from [32] (Winter), [34] (Kirtley), and [33] (Linskell) taken at approximately the same speed. The BLEEX data was recorded with the controller's sensitivity amplification gain tuned to feel comfortable to the operator with 25kg of payload. The operator described the sensation of walking as "feeling a light [torso] load of 5-10 lbs and a slight heaviness in the feet immediately after toe-off." This was considered an acceptable level of performance at the time of the data collection.

Because the human and exo limbs are coupled together and move in synchrony, time has been eliminated in Fig. 5-11 by plotting joint torque as a function of angle. The area inside the loops on the plots represents the work done by the actuator. The actuator saturation limits have been included for each plot to verify that the commanded torques are within the BLEEX limits (i.e. the controller is not saturating). Also, the BLEEX plots include both the desired torque for the joint calculated via the Sensitivity Amplification Control scheme and the actual measured torque at the joint. These two curves, in comparison to each other, show the tracking performance of the local non-linear joint torque controller. In comparison with the curves of the human CGA plots, they can help evaluate the appropriateness of the initial design assumption that human CGA data can be used as a model for BLEEX walking.

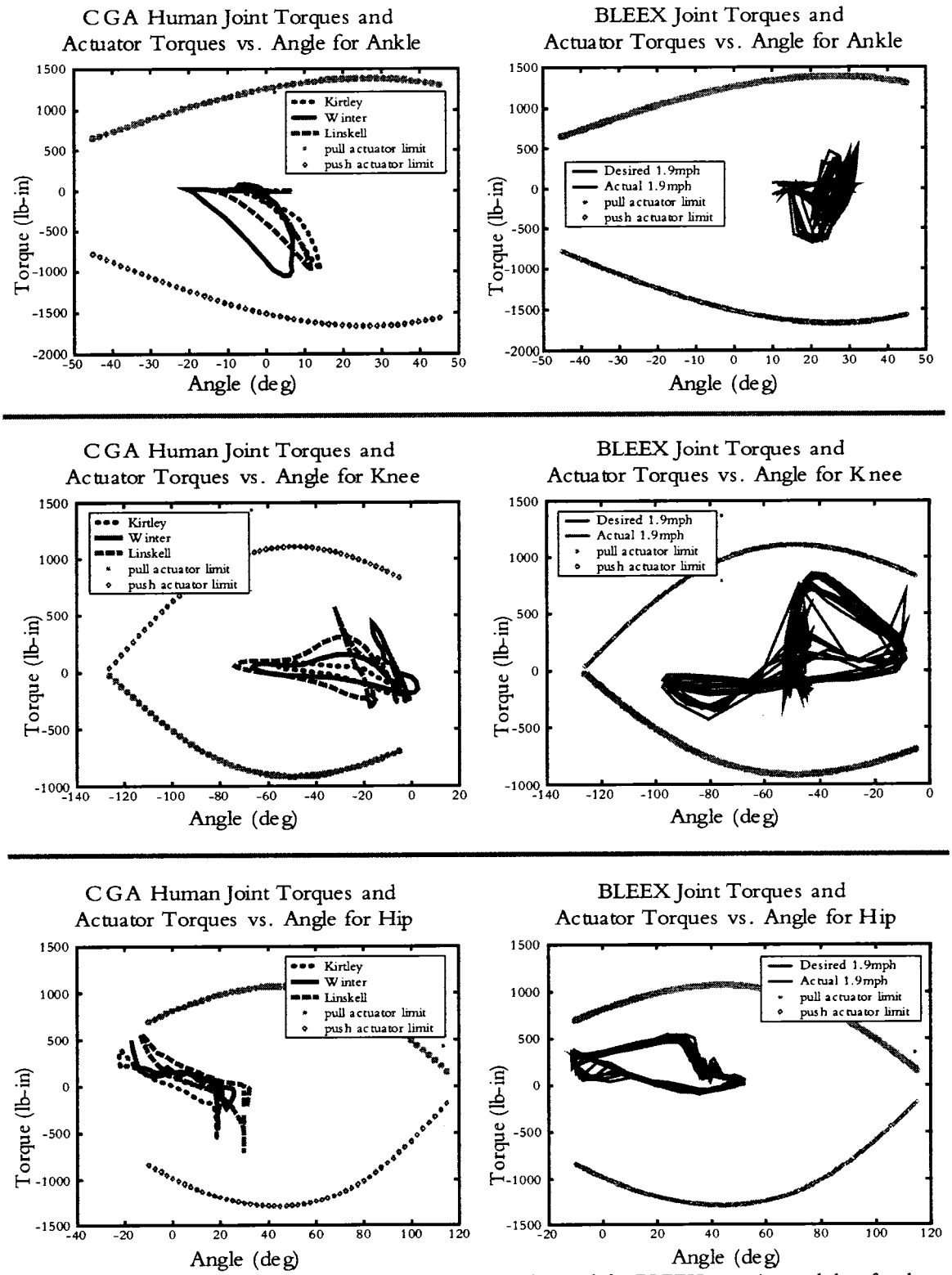


Fig. 5-11 Comparison of walking between human CGA data and the BLEEX experimental data for the ankle, knee, and hip joints. Plots of torque as a function of angle include push and pull actuator saturation limits for reference.

There are very noticeable differences between curves for each joint in Fig. 5-11, however the overall shape and the location in the torque-angle plane is similar. This suggests, at least to a first approximation, that the dynamic model is performing properly and generating estimated joint torques similar to those that occur in the human body during walking. For the ankle, little positive torque is required for the human (as seen in the left hand CGA plot). The BLEEX curve for the desired ankle torque exhibits the same characteristic. The maximum BLEEX extension torque (negative torque) is approximately half the magnitude of the maximum human extension torque. One explanation for this difference could be the difference in the posture between the two plots. The combined human and BLEEX CG is shifted posterior to the unassisted (no exoskeleton) human CG. When walking without BLEEX, the ankle extension torque becomes large to prevent the human from moving forward too fast during the swing phase. The addition of the BLEEX mass to the human, though not borne by the human directly, does act as a counterbalance to the human CG and might account for the decreased ankle torque.

The measured torque at the ankle shows significant tracking error (up to 40%) and a large positive torque overshoot (300 in-lb). The poor tracking performance is most likely due to problems getting the local joint MSS controller to behave well. After tuning the controller, torque tracking was typically within 5% of the desired torque, however this performance was affected by changes in operator and day-to-day variation in experiments. Torque tracking was not as large of a problem for the knee and hip joints. Alternatives to the non-linear joint controller are discussed in [20] and are analyzed in simulation. Future work could include implementing these alternate strategies on the BLEEX hardware in order to create a more robust local torque control loop. Improving local joint torque control for the hydraulic actuators is an important direction for future work.

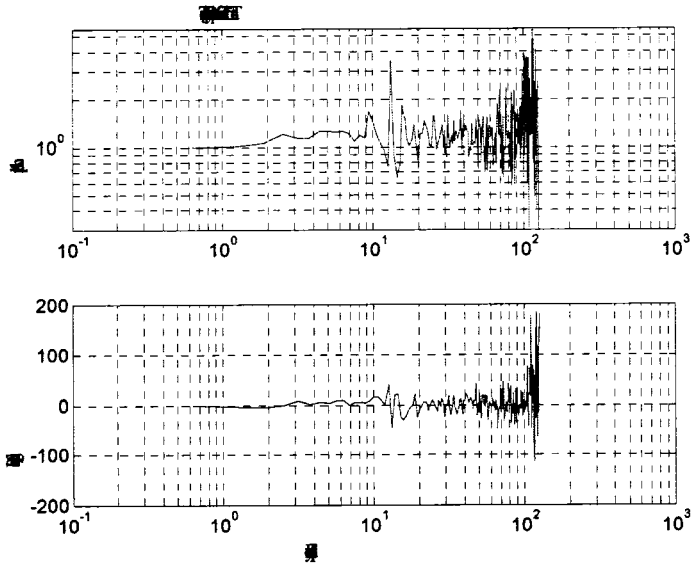


Fig. 5-12 Bode plot showing tracking performance of MSS controller.

Figure Fig. 5-12 shows an experimentally obtained Bode plot of the local MSS actuator controller. The bode plot was obtained by tracking a sinusoidal input signal and sweeping the sine wave frequency from 0 Hz to 100 Hz. The amplitude is flat out to approximately 2 Hz. Beyond 10 Hz the controller does not track well. Nonetheless, the bode plot indicates that the controller bandwidth should be sufficient for tracking the motion of human walking. Future work to improve the robustness of the controller and to automate tuning process for the MSS control gains could still be beneficial.

The knee plots from Fig. 5-11 show similar trends between human CGA data and BLEEX. The magnitude of the BLEEX torques during walking is greater than the human knee torques, however this is most likely due to the geometric sizing of BLEEX to the human operator's body. BLEEX legs were typically sized to be slightly longer than the human's legs. This ensured that BLEEX could still exert an upward force on the spine and payload when the human is standing up straight. The consequence of the longer BLEEX legs is that they cannot take advantage of the over-center characteristic of the human knee. An over-center configuration, which refers to the position of the load line in the

sagittal plane anterior to the joint, causes the knee to remain stably in a locked position under load without the need to apply torque. Second and third generations of BLEEX currently in development have been designed to be over-center during stance. Also, the next generation current exoskeleton fitting policy is to set the length of the legs to be as close to the length of the human legs as possible.

The BLEEX hip torque most closely matched the human walking CGA data. Additionally, tracking of the hip torque was the best of the three joints. In Chapter 2 it was noted that the torque for the hip was a smoothly varying almost sinusoidal signal. A sinusoidal input signal is what is used during the tuning process for the MSS controller so it stands to reason that this joint would have good tracking behavior.

Sensor data accuracy

The joint angle data was verified via measurement of limb segment absolute angles using a digital protractor from Cole-Parmer (model PRO360 950-317) that provided 0.1 deg accuracy. Force sensors used on each actuator were calibrated on an external load cell and a linear curve fit was created for data points sampled at 100 lb intervals throughout the full range of the sensors. During periodic repairs of the exoskeleton where the force sensors were removed, they were re-tested on the load cell and typically were within 1% of the original calibration data points, even after extensive use. Unfortunately, the linear accelerometers used to calculate the joint angular velocity proved to be very problematic. Each was calibrated to remove the static offset due to gravity. During operation, high frequency vibrations in the exoskeleton due to the internal combustion engine on the backpack power supply and due to shock created at heel-strike when the exoskeleton foot strikes the ground, showed up on the accelerometer output. This resulted in high-frequency oscillation of the calculated joint angular acceleration for each joint. These

oscillations frequently had amplitudes of up to 20% of the estimated joint angular acceleration amplitude that would be seen during walking. To mitigate this problem, aggressive low-pass filtering was applied to the incoming accelerometer data. Cutoff frequencies between 5 Hz and 10 Hz produced acceptable behavior during normal walking. Unfortunately, this prevented the sensitivity amplification controller from responding to high frequency human motions such as reflexes. For rapid reflex-like motion, the wearer typically complained that of a heavy or non-responsive sensation from BLEEX. Reducing the vibrations due to the internal combustion engine used in the backpack power supply would probably have eliminated most of the problems with the accelerometers. Because the control is based on an accurate model of the system and accurate data about the motion of each joint on BLEEX, finding a more reliable sensor for measuring the joint angular acceleration would contribute significantly to the overall performance of the Sensitivity Amplification Controller.

Model parameter refinement

Chapter 4 stressed the direct connection between the performance of the Sensitivity Amplification Control scheme and the accuracy of the dynamic model. During the experimental testing phase of the BLEEX project, much work was done to improve the accuracy of the model parameters for the link CG location, link mass, and link length. In some cases this meant adding detail to the CAD solid models used to calculate the CG location and link mass. The mass of each leg segment of BLEEX was experimentally verified and the wiring, electronics, and anything else attached to BLEEX (e.g. cable tie downs) was carefully weighed and incorporated into the inverse dynamics software. In an attempt to at least partially circumvent this time consuming task, one researcher in our group pursued an off-line system identification program [103]. This allowed the system

parameters to be found experimentally in an automated process; however the algorithm required the exoskeleton to be placed on a test stand separate from the human. One avenue for future study would be to create a system identification system that could refine the inverse dynamics model parameters as the system is running.

One of the most significant shortcomings of the Sensitivity Amplification Control scheme is the need for an accurate model of the BLEEX torso, because the torso model includes the mass and CG location of the payload being carried. The payload is inherently a dynamic variable in the system as the operator can add or remove items being carried at any time. This can cause both the mass and CG location of the torso to change significantly. In addition, the fuel being consumed by the backpack power supply causes the torso mass and CG to shift as a function of time. In practice, this limitation did not prevent successful demonstration of walking with BLEEX because the operator was given manual control of the torso mass and CG location model parameters in the control software via the BLEEX GUI. The operator could start BLEEX and then adjust these variables gradually until the backpack load was fully supported by BLEEX. Setting the torso mass or CG location variable too large would cause BLEEX to over actuate each joint, resulting in the actuator extending each joint against its hard stop. This would leave the wearer standing up straight, unable to bend or otherwise buckle the legs. This situation had to be carefully avoided by increasing the torso mass variable slowly, stopping the adjustment as soon as the load felt comfortably balanced by the BLEEX actuators. In a real world application however, this strategy would not be practical. One alternative explored to circumvent the problem of an unknown and varying torso mass and CG was to add a six-axis force sensor at the attachment point between the compliant human harness and the exoskeleton spine. This would have allowed the controller to estimate the

mass and CG of the torso, however because of time constraints on the project, this functionality was not fully implemented in the BLEEX control code.

An alternative BLEEX control scheme was explored by Huang [66] which eliminated the problem created by an unknown and varying torso mass by combining the Sensitivity Amplification Control scheme and a master-slave style of position control. This new control scheme, called hybrid BLEEX control, divides the walking gait cycle into stance control and swing control phases. Traditional master-slave style position control is used for the BLEEX stance leg (including torso and backpack) and the sensitivity amplification controller is used for the swing leg. The hybrid controller is designed to smoothly transitions between these two schemes as the person walks. For the stance leg (i.e. the leg that is on ground), position control is used to servo BLEEX joint angles to track the human's joint angles. Since the BLEEX torso weight is carried by the stance leg, there is no need to know the mass and center of gravity (CG) properties of the torso. As a tradeoff, the position control used in this method requires the human to wear seven inclinometers on the human body to measure human limb and torso angles. These additional sensors require careful design to securely fasten them to the human and increase the time to don (and doff) BLEEX.

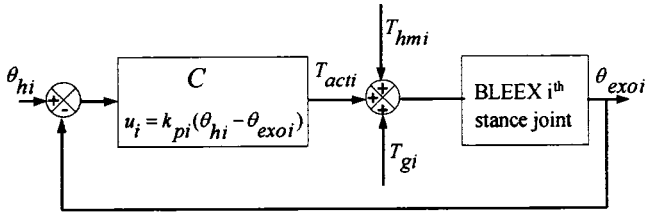


Fig. 5-13 Shows the position control block diagram used for the joints of the stance leg (ankle, knee, and hip) when the exoskeleton is in hybrid BLEEX control mode.

The position controller is implemented as a proportional controller that servoes the BLEEX joint angle to match the joint angle measured on the human body. Because this

control scheme is based on the assumption that the human can move while attached to BLEEX such that a joint angle difference is created between the human and exoskeleton joint angles, it is necessary to modify the connection between the person and BLEEX to have additional compliance. This additional compliance caused the human to have less proprioceptive feedback from the exoskeleton, making balancing difficult for the hybrid control scheme. In laboratory experiments with hybrid BLEEX control, a person could walk while steadying herself using a handrail, but could not walk unassisted. This was primarily due to the balance problem caused by the added compliance at the human-exoskeleton connection points (i.e. the shoe and the vest attachment).

Position-control worked adequately for the stance leg where, for normal walking, motion is relatively small. For the swing leg, the position-controller bandwidth did not allow the exoskeleton to accurately track the rapid natural movement of human leg. For this reason, the Sensitivity Amplification Control scheme was activated for the leg in swing as soon as the controller detected that the foot had left the ground. With hybrid BLEEX control, a person could walk in BLEEX at a maximum speed of 0.5 m/sec with a maximum payload of 18 kg (40 lbs)—tested in a laboratory setting on treadmill. As a downside, this control scheme required the attachment of sensors on the human body and required power and data cables to be run between each of these sensors and the exoskeleton control computer. Attaching these sensors to the body and connecting each of the sensor cables added significant time to the exoskeleton donning and doffing procedures. The tradeoff for this additional complexity is robustness of the controller to the exoskeleton payload. Future research that explored ways to increase the performance of the hybrid scheme could be very beneficial. In particular, research should focus on solving the balance problem created in the current implementation of hybrid control.

Chapter 6

Conclusion

While there is still significant work left before the Berkeley Human Exoskeleton project is complete, BLEEX has successfully walked, carrying its own weight and producing its own power. This makes it the first lower extremity exoskeleton capable of carrying a payload and being energetically autonomous. Currently BLEEX has been demonstrated to support up to 50 kg (exoskeleton weight + payload), walk at speeds up to 1.3 m/s, and shadow the operator through most maneuvers without any human sensing or pre-programmed motions.

BLEEX was proven to work not just in highly controlled experimental setting, but also during free unstructured walking and with complete freedom of the operator to move as he or she pleases. The BLEEX controller never overrides human intent, allowing a person to turn, squat, start and stop walking at any time, and change walking pace at any time.

6.1 Sensitivity Amplification Control: A New Paradigm

BLEEX is not a typical servo-mechanism. While providing disturbance rejection along some axes preventing motion in response to gravitational forces, BLEEX actually encourages motion along other axes in response to pilot interface forces. This characteristic requires large sensitivity to pilot forces which invalidates certain assumptions of the standard control design methodologies, and thus requires a new design approach.

The controller described here uses the inverse dynamics of the exoskeleton as a

positive feedback controller so that the loop gain for the exoskeleton approaches unity (slightly less than 1). Experiments with BLEEX have shown that this control scheme has two superior characteristics: 1) it allows for the same wide bandwidth maneuvers a human is capable of performing; 2) it is unaffected by changing human dynamics (i.e. no changes to the controller are required when pilots are switched). The trade off is that it requires a relatively accurate model of the system.

6.2 Lessons learned from powered exoskeletons

The BLEEX project successfully tackled the one hurdle that has stalled almost every attempt to create a human exoskeleton over the past 50 years: integration. Despite the complexity in terms of the number of actuated degrees of freedom, the number of sensors and wires, the control software, and the portable hydraulic power supply, all components were successfully integrated in the final design without the need for any power or data tethers. What's more, a person can put on the exoskeleton and walk while carrying a load without any specific training and without the need to make any changes to the controller.

The mantra that came out of the experience of designing and testing BLEEX in order of priority was:

1) Reduce Complexity

2) Reduce Power

3) Improve Operator Comfort.

6.3 Reducing complexity

In reality, this complexity did cause an almost endless headache for the engineers involved on the project. From intermittent contact inside of electrical connectors to

hydraulic leaks, to fatigue failures in the compliant foot components, there was a constant queue of issues to be resolved to keep the exoskeletons (we built two BLEEX models) running.

6.4 Reducing power consumption

The issue of power consumption became a large roadblock late in the project. The IC engine based power supplies developed for BLEEX were only marginally adequate in terms of power output. Increasing the engine power came at the price of additional weight. Though the operator didn't feel the weight of the power supply, its does reduce the total amount of payload that BLEEX can carry. Our team explored every feasible portable technology we could locate from fuel cells to advanced batteries like lithium-polymer and zinc-air, monopropellant engines. At the time, only a gasoline based IC engine could provide the needed power density and reliability to make BLEEX fully autonomous. Still, the IC engine was unattractive due to its loud operation noise, high operating temperature, need for active cooling, and the inability to safely operate it indoors. Though new power technologies may one day emerge that can make the power supply problem for mobile robotics irrelevant, for the foreseeable future the only solution is to reduce the overall system power requirements. Moving forward, current exoskeleton development in our lab is incorporating passive impedance control (e.g. damping) rather than active control for some or all of the joints of the exoskeleton depending on the specific application.

6.5 Improving pilot comfort

Finally, pilot comfort still needs significant improvement before BLEEX can be said to be truly transparent to the operator. Additional compliance needs to be added for non

sagittal plane motions in order to accommodate a wider range of movement in the exoskeleton. The BLEEX spine, which is currently rigid, needs to have compliance added to allow the operator to more naturally twist side to side and bend over. The harness also needs improvement to allow for easier donning and doffing and to improve long term comfort. Currently, a compliant attachment strap is used to connect the human shanks to the exoskeleton shanks. The ergonomics of this location as a connection point have not yet been evaluated and some users complain that this connection is uncomfortable when the BLEEX is worn for long periods of time.

6.6 Extension to activities outside of walking

Currently, BLEEX has been tested walking at arbitrary speeds self-selected by the wearer in unstructured environments. The wearer is free to turn, start, and stop motion at will without any explicit commands or changes to the BLEEX controller. In addition, BLEEX has been demonstrated with the user squatting and balancing on one leg while carrying a payload of up to 35 kg. Further work needs to be done to improve the power supply, actuation capability, and the local joint torque control algorithm such that high speed maneuvers (i.e. jogging and running) can be tested. Stair and ramp descent and ascent have only been tested in limited scenarios (short flights of one to three steps and shallow ramps of 6-15 deg) and have been shown to work with the current Sensitivity Amplification Control scheme. However, more extensive testing needs to be done. In particular, the current implementation of the control scheme begins to transfer weight to the forward stance leg too early for stair and ramp descent and ascent. This is because the weight distribution scheme for the double stance case does not take into account the vertical position of the foot in the sagittal plane (only the horizontal position is considered). Other activities that have not been explored include sitting in a chair or seat,

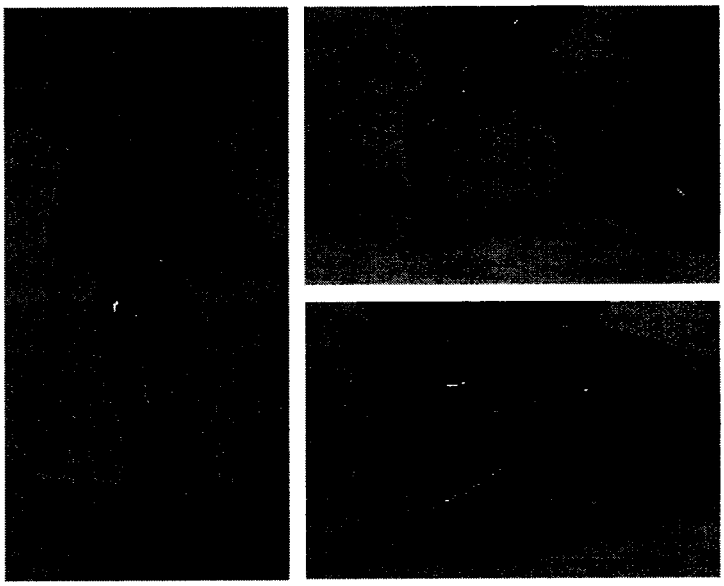
kneeling, lying prone on the ground, crawling and standing up from a prone or kneeling position. Many of these will require modifications to the control and some, such as crawling or kneeling, may require additional sensors to detect the ground contact of other parts of the exoskeleton structure.

6.7 The future of human exoskeletons

Human exoskeletons are truly in their infancy. Despite 50 years of design progress, only a handful of working prototypes have successfully been developed. Only within the last five years has computing and control technology made the success of an autonomous load-carrying human exoskeleton a possibility. Nevertheless, the same problems of weight, energetic efficiency, and power consumption that derailed human exoskeleton projects as far back as the original General Electric "Hardiman" project, threaten the practicality of even the most advanced concepts currently under development. Science fiction often portrays soldiers of the future fighting epic battles as super-human exoskeleton-enhanced warriors. A far more likely future will see the fundamental technologies behind exoskeletons slowly creep into markets such as orthotics, prosthetics, sports equipment, body armor support, and other even more mundane applications. Many of these applications will require little or no control or actuation. However, the experience gained from creating robotic systems that are intimately interconnected with the human body, both in form and operation, will be what enables the progress of exoskeleton design and control in these new directions.

The second and third generations of BLEEX currently under development in the Berkeley Robotics and Human Engineering Laboratory is following one of these pathways that is leading towards an exoskeleton that requires no external power supply at all.

Instead, it will harness energy for the moving mass of the human body and payload to control the stiffness and damping of its joints as a function of joint angle and phase in the gait cycle. A tradeoff is made for this system in that it can no longer inject power in a joint, meaning the energy needed for climbing stairs and ascending ramps will have to be supplied by the wearer. Additionally, these new systems will not be able to compensate for the dynamic forces and torques generated when moving the mass of the exoskeleton and



payload. At this time, our research group believes that the benefits of having a lighter system (no internal combustion engine power supply), a longer runtime (unlimited runtime for some of our newest energy harvesting and purely mechanical designs), and a simpler more robust system (fewer sensors, cables, actuators, and failure modes) combine to more than outweigh the downsides of this approach.

Fig. 6-1 Second generation BLEEX system currently under development.

As a comparison, the one of the second generation BLEEX designs (Fig. 6-1), which incorporates passive knee joints with controlled damping and passive hip and ankle joints with angle modulated stiffness, has a total system weight of 14 kg (compared to nearly 70

kg for the first generation BLEEX). This second generation design runs hydraulic control valves, sensors, and a control computer for up to 96 hours on approximately 0.6 kg of lithium-polymer batteries (compared to a 3 hour runtime with a 23 kg power supply for the first generation BLEEX). This new system no longer suffers from the power supply flow rate and torque tracking limitations of the first generation BLEEX. It has allowed a person to run at up to 5.3 m/sec while carrying payloads of up to 50 kg (compared to 1.4 m/sec with a 15 kg payload for the first generation of BLEEX).

The future of human exoskeletons is beginning to take shape; hopefully the work of this thesis will provide one more stepping stone to help drive this progress forward. It is now conceivable that the problem of load carriage on the human body and the associated physical cost may one day become a forgotten memory.

References

[1] N. Shachtman, "The 4th annual: Year In Ideas; Exoskeleton Strength," in *The New York Times*, vol. 6. New York, NY, pp. 68, 2004.

[2] J. J. Knapick, K. L. Reynolds, and E. Harman, "Load carriage using packs: A review of physiological, biomechanical, and medical aspects," *Applied Ergonomics*, vol. 27 **(3)**, pp. 207-216, 1996.

[3] Rippel E. M. Facility, Dartmouth University, "Formicidae SEM image," [Online Document], 2004, [cited Mar. 3, 2006], Available HTTP: http://remf.dartmouth.edu/images/insectPart3SEM/source/31.html

[4] "Exoskeleton," in *The American Heritage Dictionary of the English Language*, 4th ed: Houghton Mifflin Company, 2004.

[5] General Electric Co., "Exoskeleton Prototype Project, Final Report of Phase I," Schenectady, NY, pp., 1966.

[6] E. T. Carre, "Historical review of the load of the foot-soldier," *United States Army Quartermaster Research and Development Center, Natck, MA*, vol. 8, Tentage and Equipage Series Report, 1952.

[7] J. G. Donald, "March fractures: A study with special reference to etiological factors," *J. Bone Joint Surg.*, vol. 29, pp. 297-300, 1947.

[8] J. J. Knapick, K. L. Reynolds, and E. Harman, "Soldier load carriage: historical, physiological, biomechanical, and medical aspects," *Mil. Med.*, vol. 169 **(1)**, pp. 45-56, 2004.

[9] E. T. Renbourn, "The knapsack and pack; an historical and physiological survey with particular reference to the British soldier," *J. R. Army Med. Corp*, vol. 100, pp. 1-15,77-88,193-200, 1954.

[10] W. H. Harper and J. J. Knapick, "Investigation of Female Load Carrying Performance," *U.S. Army Research Laboratory Human Research and Engineering Directorate Annual Report, Aberdeen Proving Ground, MD*, vol. 95MM5589, pp. 1-48, 1995.

[11] M. F. Haisman, "Determinants of load carrying ability," *Applied Ergonomics*, vol. 19 **(2)**, pp. 111-121, 1988.

[12] J. J. Knapick, "Physiological, Biomechanical and Medical Aspects of Soldier Load

Carriage," NATO Res. and Tech. Org. Human Factors and Medicine Panel, Kingston, Canada, pp. 34-52, 2000.

[13] S. Clifford and K. Kingsbury, "Otfitting a Soldier," in *Life*, pp. 5, Dec. 3, 2004.

[14] "Orthotic," in *The American Heritage Stedman's Medical Dictionary*, 2nd ed: Houghton Mifflin Company, 2002.

[15] S. E. Irby, K. R. Kaufman, J. W. Mathewson, and D. H. Sutherland, "Automatic Control Design for a Dynamic Knee-Brace System," *IEEE Trans. on Rehab. Eng.*, vol. 7 **(2)**, pp. 135-139, 1999.

[16] K. Nagai and I. Nakanishi, "Power assisting control of robotic orthoses considering human characteristics on assisted motions," Proc. of the 1999 IEEE Intl. workshop on Robotics and Human Interaction, Pisa, Italy, pp. 1-6, 1999.

[17] J. A. Doubler and D. S. Childress, "Design and evaluation of a prosthesis control system based on the concept of extended physiological proprioception," *J. Rehabil. Res. Dev.*, vol. 21 **(1)**, pp. 19-31, 1984.

[18] A. Bar, G. Ishai, P. Meretsky, and Y. Koren, "Adaptive microcomputer control of an artificial knee in level walking," *J. Biomedical Eng.*, vol. 5 **(2)**, pp. 145-150, 1983.

[19] L. Peeraer, K. Tilley, and G. V. d. Perre, "A computer-controlled knee prosthesis: a preliminary report," *J. Med. Eng. Technol.* , vol. 13 **(1-2)**, pp. 134-135, 1989.

[20] J.-L. Racine, "Control of a Lower Extremity Exoskeleton for Human Performance Augmentation," in *Ph.D. Thesis, Mechanical Engineering*. Berkeley, CA: U.C. Berkeley, 2003.

[21] "Planes," in *Taber's Cyclopedic Medical Dictionary*, D. Venes, Ed., 20th ed. Philadelphia, PA: F.A. Davis Company, 2005.

[22] W. Woodson, B. Tillman, and P. Tillman, *Human Factors Handbook*. New York, NY: McGraw-Hill, 1992.

[23] D. A. Winter, *Biomechanics and motor control of human movement*. New York, NY: Wiley, 1990.

[24] S. I. Fox, *Human Physiology*. Boston, MA: McGraw-Hill, 2004.

[25] J. Perry, *Gait Analysis*. Thorofare, NJ: SLACK Incorporated, 1992.

[26] M. P. Murray, A. B. Drought, and R. C. Kory, "Walking patterns of normal men," *J. Bone Joint Surg.*, vol. 46A **(2)**, pp. 335-360, 1964.

[27] W. T. Dempster, "Space requirements of the seated operator," Wright-Patterson Air Force Base, Dayton, OH WADCTR55-159, 1955.

[28] J. Rose and J. G. Gamble, *Human Walking*, 2nd ed. Baltimore, MD: Williams & Wilkins, 1994.

[29] P.-O. Astrand, K. Rodahl, H. A. Dahl, and S. Stromme, *Textbook of Work Physiology*, 4th ed. Champaign, IL: Human Kinetics, 2003.

[30] D. Popovic and T. Sinkjaer, *Control of Movement for the Physically Disabled*. London, England: Springer, 2000.

[31] A. Chu, "Design of the Berkeley Lower Extremity Exoskeleton (BLEEX)," in *Ph.D. Thesis, Dept. Mechanical Engineering*. Berkeley, CA: U.C. Berkeley, 2005.

[32] A. Winter,International Society of Biomechanics, Biomechanical Data Resources, "Gait Data," [Online Document], [cited 2004], Available HTTP: http://www.isbweb.org/data/

[33] J. Linskell,CGA Normative Gait Database, "Limb Fitting Centre, Dundee, Scotland, Young Adult," [Online Document], [cited 2004], Available HTTP: http://guardian.curtin.edu.au/cga/data/

[34] H. Kirtley,CGA Normative Gait Database, "10 Young Adults," [Online Document], [cited 2004], Available HTTP: http://guardian.curtin.edu.au/cga/data/

[35] A. Chu, H. Kazerooni, and A. Zoss, "On the Biomimetic Design of the Berkeley Lower Extremity Exoskeleton," IEEE International Conf. on Robotics and Automation, Barcelona, Spain, 2005.

[36] Y. Yamazaki, T. Ohkuwa, H. Itoh, and M. Suzuki, "Reciprocal activation and coactivation in antagonistic muscles during rapid goal-directed movements," *Brain Research Bulletin*, vol. 34 **(6)**, pp. 587-593, 1994.

[37] D. J. Bennett, "Torques generated at the human elbow joint in response to constant position errors imposed during voluntary movements," *Exp. Brain Research*, vol. 95 **(3)**, pp. 488-498, 1993.

[38] N. Hogan, "Adaptive Control of Mechanical Impedance by Coactivation of Antagonist Muscles," *IEEE Trans. on Automatic Control*, vol. 29 **(8)**, pp. 681-690,

1984.

[39] S. J. DeSerres and T. E. Milner, "Wrist muscle activation patterns and stiffness associated with stable and unstable mechanical loads," *Exp. Brain Research*, vol. 86 **(2)**, pp. 451-458, 1991.

[40] T. E. Milner and C. Cloutier, "Damping of the wrist joint during voluntary movement," *Exp. Brain Research*, vol. 122 **(3)**, pp. 309-317, 1998.

[41] R. Q. V. d. Linde, "Design, Analysis, and Control of a Low Power Joint for Walking Robots, by Phasic Activation of McKibben Muscles," *IEEE Trans. on Robotics and Automation*, vol. 15 **(4)**, pp. 599-604, 1999.

[42] "Hardiman I Prototype Project, Special Interim Study," N. Schenectady, General Electric Report S-68-1060, pp., 1968.

[43] N. J. Mizen,"Preliminary Design for the Shoulders and Arms of a Powered, Exoskeletal Structure," VO-1692-V-4, pp., 1965.

[44] P. F. Groshaw,"Hardiman I Arm Test, Hardiman I Prototype," General Electric Co., N. Schenectady, Report S-70-1019, pp., 1969.

[45] B. J. Makinson,"Research and Development Prototype for Machine Augmentation of Human Strength and Endurance, Hardiman I Project," General Electric Co., N. Schenectady, General Electric Report S-71-1056, pp., 1971.

[46] R. S. Mosher, "Force-Reflecting Electrohydraulic manipulator," *Electro-Technology*, vol. Dec., 1960.

[47] M. Vukobratovic, D. Hristic, and Z. Stojiljkovic, "Developement of active anthropomorphic exoskeleton," *Medical and Biological Engineering*, vol. 12 **(January)**, pp. 66-80, 1973.

[48] M. Vukobratovic, V. Ciric, and D. Hristic, "Contribution to the Study of Active Exoskeletons," Proc. of the 5th IFAC Congress, PAris, 1972.

[49] M. Vukobratovic and Y. Stepanenko, "On the stability of anthropomorphic systems," *Mathematical Biosciences*, vol. 15 **(1)**, 1972.

[50] M. Vukobratovic, "How to control the artificial anthropomorphic systems," *IEEE Trans. Syst., Man, Cybernetics*, vol. SMC-3, pp. 497, 1973.

[51] K. Hirai, M. Hirose, Y. Haikawa, and T. Takenaka, "The development of Honda

humanoid robot," Proc. of the 1998 IEEE International Conference on Robotics & Automation, Leuven, Belgium, pp. 1321-1326, 1998.

[52] H. Kawamoto and Y. Sankai, "Power Assist System HAL-3 for gait Disorder Person," International Conference on Computers for Handicapped Persons, Linz, Austria, pp. 196-203, 2002.

[53] G. Colombo, M. Jorg, and V. Dietz, "Driven Gait Orthosis to do Locomotor Training of Paraplegic Patients," 22nd Annual International Conf. of the IEEE - EMBS, Chicago, IL, pp. 3159-3163, 2000.

[54] J. Pratt, B. Krupp, C. Morse, and S. Collins, "The RoboKnee: An Exoskeleton for Enhancing Strength and Endurance During Walking," IEEE Intl. Conf. on Robotics and Automation, New Orleans, LA, pp. 2430-2435, 2004.

[55] G. Pratt and M. Williamson, "Series elastic actuators," IEEE International Conf. on Intelligent Robots and Systems, Pittsburgh, PA, pp. 399-406, 1995.

[56] J. Pratt, B. Krupp, and C. Morse, "Series elastic actuators for high fidelity force control," *Industrial Robot: An International Journal*, vol. 29 **(3)**, pp. 234-241, 2002.

[57] H. Kawamoto, S. Kanbe, and Y. Sankai, "Power Assist Method for HAL-3 Estimating Operator's Intention Based on Motion Information," Proc. of 2003 IEEE Workshop on Robot and Human Interactive Communication, Millbrae, CA, pp. pp. 67-72, 2003.

[58] S. Lee and Y. Sankai, "Power assist control for walking aid with HAL-3 based on EMG and impedance adjustment around knee joint," IEEE International Conf. on Intelligent Robots and Systems, Lausanne, Switzerland, pp. 1499-1504, 2002.

[59] M. DiCicco, L. Lucas, and Y. Matsuoka, "Comparison of Control Strategies for an EMG Controlled Orthotic Exoskeleton for the Hand," IEEE Iternational Conference on Robotics & Automation, New Orleans, LA, pp. 1622-1627, 2004.

[60] J. F. Jansen, "Apparatus and Methods for a Human Extender," Patent No. 6,272,924, Lockheed Martin Energy Research Corporation, USA, 2001.

[61] H. Kazerooni, "The Human Power Amplifier Technology at the University of California, Berkeley," *Journal of Robotics and Autonomous Systems*, vol. 19, pp. 179-187, 1996.

[62] H. Kazerooni, "Human-Robot Interaction via the Transfer of Power and Information Signals," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 20 **(2)**, pp. 450-463, 1990.

[63] H. Kazerooni and J. Guo, "Human Extenders," *ASME Journal of Dynamic Systems, Measurements, and Control*, vol. 115 **(2B)**, pp. 281-289, 1993.

[64] H. Kazerooni and R. Steger, "The Berkeley Lower Extremity Exoskeleton," *Journal of Dynamic Systems, Measurement, and Control*, vol. 128 **(1)**, pp. 14-25, 2005.

[65] H. Kazerooni, J.-L. Racine, L. Huang, and R. Steger, "On the Control of the Berkeley Lower Extremity Exoskeleton (BLEEX)," IEEE International Conf. on Robotics and Automation, Barcelona, Spain, pp. 4353-4360, 2005.

[66] L. Huang, R. Steger, and H. Kazerooni, "Hybrid control of the Berkeley Lower Extremity Exoskeleton (BLEEX)," 2005 ASME International Mechanical Engineering Congress and Exposition, Orlando, FL, 2005.

[67] H. Kazerooni, R. Steger, and L. Huang, "Hybrid Control of the Berkeley Lower Extremity Exoskeleton," *The International Journal of Robotics Research*, vol. 25 **(5)**, 2006.

[68] H. Kazerooni and T. Snyder, "A Case Study on Dynamics of Haptic Devices: Human Induced Instability in Powered Hand Controllers," *Journal of Guidance, Control, and Dynamics*, vol. 18 **(1)**, pp. 108-113, 1995.

[69] T. McGee, J. Raade, and H. Kazerooni, "Monopropellant-Driven Free Piston Hydraulic Pump for Mobile Robotic Systems," *Journal of Dynamic Systems, Measurement and Control*, vol. 126, pp. 75-81, 2004.

[70] K. Amundsen, J. Raade, N. Harding, and H. Kazerooni, "Hybrid Hydraulic-Electric Power Unit for Field and Service Robots," IEEE Int. Conf. on Intelligent Robots and Systems, Edmunton, Canada, Aug. 2005.

[71] J. Raade and H. Kazerooni, "Analysis and Design of a Novel Power Supply for Mobile Robots," *IEEE. Trans. Autom. Sci. Eng.*, vol. 2 **(3)**, pp. 226-232, 2004.

[72] S. Kim, G. Anwar, and H. Kazerooni, "High-speed Communication Network for Controls with Application on the Exoskeleton," American Control Conference, Boston, MA, pp. 355-360, 2004.

[73] S. Kim and H. Kazerooni, "High Speed Ring-based distributed Networked control system For Real-Time Multivariable Applications," ASME International Mechanical Engineering Congress and Exposition, Anaheim, CA, 2004.

[74] A. Zoss, "On the Mechanical Design of the Berkeley Lower Extemity Exoskeleton," IEEE International Conf. of Intelligent Robots and Systems, Edmunton, Canada, 2005.

[75] R. Steger, S. Kim, and H. Kazerooni, "Control Scheme and Networked Control Architecture for the Berkeley Lower Extremity Exoskeleton (BLEEX)," IEEE International Conf. on Robotics and Automation, Orlando, FL, pp. (in pub), 2006.

[76] H. Kazerooni and S. Mahoney, "Dynamics and Control of Robotic Systems Worn By Humans," *ASME Journal of Dynamic Systems, Measurements, and Control*, vol. 113 **(3)**, pp. 379-387, 1991.

[77] H. Kazerooni and M. Her, "The Dynamics and Control of a Haptic Interface Device," *IEEE Trans. on Robotics and Automation*, vol. 10 **(4)**, pp. 453-464, 1994.

[78] K. Nagai, I. Nakanishi, and H. Hanafusa, "Development of an 8 DOF robotic orthosis for assisting human upper limb motion," IEEE International Conf. on Robotics and Automation, Leuven, Belgium, pp. 3486-3491, May 1998.

[79] K. Kiguchi, T. Tanaka, K. Watanabe, and T. Fukuda, "Exoskeletons for human upper-limb motion support," IEEE International Conf. on Robotics and Automation, Taipei, Taiwan, pp. 2206-2211, 2003.

[80] K. Kiguchi and T. Fukuda, "A 3DOF Exoskeleton for Upper-Limb Motion Assist-Consideration of the Effect of Bi-Articular Muscles," IEEE International Conf. on Robotics and Automation, New Orleans, LA, pp. 2424-2429, 2004.

[81] K. Nishiwaki, Y. Murakami, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "A Six-axis Force Sensor with Parallel Support Mechanism to Measure the Ground Reaction Force of Humanoid Robot," IEEE International Conf. on Robotics and Automation, Washington, DC, pp. 2277-2282, May 2002.

[82] F. Pin and R. Lind,Oakridge National Laboratory Fact Sheet, "Foot force-torque sensors," [Online Document], 2005, [cited February, 2005], Available HTTP: http://www.ornl.gov/sci/engineering_science_technology/roboticsenergetics/humanamplifying.htm

[83] A. V. Hill, "The abrupt transition from rest to activity in muscle," *Proc. of the Royal Society of London, Series B*, vol. 136 **(884)**, pp. 399-420, 1949.

[84] D. R. Wilkie, "The relation between force and velocity in human muscle," *J. Physiology*, vol. K110, pp. 248-280, 1950.

[85] J. M. Winters and L. Stark, "Analysis of fundamental human movement patterns through the use on in-depth antagonistic muscle models," *IEEE Trans. on Biomedical Engineering*, vol. BME32 **(10)**, pp. 826-839, 1985.

[86] H. Kazerooni, "The extender technology at the University of California, Berkeley,"

Journal of the Society of Instrument and Control Engineers in Japan, vol. 34 **(4)**, pp. 291-298, 1995.

[87] W. Q. D. Do and D. C. H. Yang, "Inverse Dynamic Analysis and Simulation of a Platform Type of Robot," *J. Robotic Systems*, vol. 5 **(53)**, pp. 209-227, 1988.

[88] K. Y. Tsai and D. Kohli, "Modified Newton-Euler Computational Scheme for Dynamic Analysis and Simulation of Parallel Manipulators with Applications to Configuration based on R-L Actuators," Proceedings of the 1990 ASME Design Engineering Technical Conferences, pp. 111-117, 1990.

[89] P. Guglielmetti and R. Longchamp, "A Closed Form Inverse Dynamics Model of the Delta Parallel Robot," Proceedings of the 1994 International Federation of Automatic Control Conference on Robot Control, pp. 39-44, 1994.

[90] R. Murray, Z. Li, and S. S. Sastry, *A Mathematical Introduction to Robotic Manipulation.* Boca Raton, FL: CRC Press, 1994.

[91] W. A. Khan, V. N. Krovi, S. K. Saha, and J. Angeles, "Recursive Kinematics and Inverse Dynamics for a Planar 3R Parallel Manipulator," *ASME J. of Dynamic Systems, Measurement, and Control*, vol. 127 **(DEC)**, pp. 529-536, 2005.

[92] L.-W. Tsai, "Solving the Inverse Dynamics of a Stewart-Gough Manipulator by the Principle of Virtual Work," *ASME J. of Mechanical Design*, vol. 122 **(1)**, pp. 3-9, 2000.

[93] C. D. Zhang and S. M. Song, "An Efficient Method for Inverse Dynamics of Manipulators Based on the Virtual Work Principle," *J. Robotic Systems*, vol. 5, pp. 605-627, 1993.

[94] K. Miller, "Experimental Verification of Modeling of Delta Robot Dynamics by Application of Hamilton's Principle," IEEE International Conf. on Robotics and Automation, pp. 532-537, 1995.

[95] J. J. Murray and G. H. Lovell, "Dynamic Modeling of Closed-Chain Robotic Manipulators and Implications for Trajectory Control," *IEEE Transactions on Robotics and Automation*, vol. 5, pp. 522-528, 1989.

[96] K. Sugimoto, "Kinematic and Dynamic Analysis of Parallel Manipulators by Means of Motor Algebra," *ASME Journal of Mechanisms, Transmissions, and Automation in Design*, vol. 109 **(1)**, pp. 3-5, 1987.

[97] J. Y. S. Luh and Y. F. Zheng, "Computation of Input Generalized Forces for Robots with Closed Kinematic Chain Mechanisms," *IEEE Journal of Robotics and*

Automation, vol. RA-1, pp. 95-103, 1985.

[98] J. Wang and C. M. Gosselin, "A New Approach for the Dynamic Analysis of Parallel Manipulators," *Multibody System Dynamics*, vol. 2, pp. 317-334, 1998.

[99] J. W. Grizzle, G. Abba, and F. Plestan, "Asymptotically stable walking for biped robots: Analysis via systems with impulse effects," *IEEE Trans. on Automatic Control*, vol. 46, pp. 51-64, 2001.

[100] E. R. Westervelt, J. W. Grizzle, and C. Canudas, "Switching and PI control of walking motions of planar biped walkers," *IEEE Transactions on Automatic Control*, vol. 48 **(2)**, pp. 308-312, 2003.

[101] J. Green and J. K. Hedrick, "Nonlinear Speed Control for Automotive Engines," Proc. 1990 American Control Conference, San Diego, CA, pp. 2891-2898, 1990.

[102] J. K. Hedrick and P. P. Yip, "Multiple Sliding Surface Control: Theory and Application," *Journal of Dynamic Systems, Measurement, and Control*, vol. 122 **(4)**, pp. 586-593, 2000.

[103] J. Ghan and H. Kazerooni, "System Identification for the Berkeley Lower Extremity Exoskeleton (BLEEX)," IEEE International Conf. on Robotics and Automation, Orlando, FL, pp. (in pub.), 2006.

Appendix A

BLEEX Control Software

The BLEEX control software is written is ANSI standard C and was compiled using the GCC compiler for x86 processor platforms. The code is designed to run as a standalone executable in DOS and it interfaces with the custom BLEEX electronics hardware attached to the BLEEX control computer. The control software is divided into multiple pairs of text files of the form "name.c" and "name.h". The "name.c" file contains functions and program code and the corresponding "name.h" file contains function declarations and variable definitions.

The following files comprise the BLEEX control firmware (for each .c file, a corresponding .h file exists with function declarations and defined variables):

Defines.h	Contains all model parameters and controller variables
ExoMain.c	Main routine
Sensors.c	Functions to read in, scale, and apply calibration curves to all exoskeleton sensors except the linear acceleroomters (see Accel.c / Accel.h)
Accel.c	Functions to read in, scale, and apply calibration curves to all exoskeleton linear accelerometers as well as calculate joint angular acceleration

JointCtl.c	Local non-linear joint torque controller
Fhm.c	Sensitivity Amplification Control scheme is implemented here
Jump.c	Inverse dynamics for a 3-link chain (leg swinging in the air)
SSup.c	Inverse dynamics for a 7-link chain (one foot on the ground)
1Red.c	Inverse dynamics for two 3-link chains (both feet flat on ground)
DSup.c	Inverse dynamics for two 3-link and one 4-link chain
Filters.c	Generic second-order Butterworth lowpass filter
PCI.c	Functions to communicate with the external BLEEX LabView based GUI
Record.c	Functions to allow for real-time data-logging while BLEEX is running.

Appendix A.1 – Defines.h

```
#define THM_ERROR_CONTROL
#define FREQ                    2000              // sampling frequency (Hz), 2kHz default
#define TS                      0.0005             // sampling time (sec), 500us default
//#define FREQ                  1400              // sampling frequency (Hz), 1400Hz
//#define TS                    7.14285714286e-4   // sampling time (sec), 714.286us

#define GUI_DATA_ARRAY_SIZE     300                 // number of variables being passed to GUI in packet, added by JRS,
06-10-2004
#define RECORDED_DATA_FREQ      50                  // Hz, frequency at which you want to log data locally on the exo,
added by JRS, 06-10-2004
#define MAX_RECORED_DATA_POINTS 10000               // number of data points that will be logged to flash disk on exo,
added by JRS, 06-10-2004
#define NUM_RECORDED_FILES      7

#define HEEL_ANKLE_TOE_ANGLE    1.176               // +ve Angle between the heel and dorsal foot segments  (rad)
#define ANKLE_TOE_HEEL_ANGLE    0.994               // +ve Angle between the dorsal foot and sole segments  (rad)
#define ACC_THIGH_ANGLE         8.726646e-3
#define KF                      0                   // double support distribution factor
#define DEFAULT_DYNDISTFACTOR   0.00333
#define DEFAULT_VALVE_IN        0                   // default valve input voltage at initialization
#define DEFAULT_MANUAL_TORQUE   0                   // default value for manual torque at initialization
#define DEFAULT_THM_STATUS      1                   // default status of accelerometer gains, 0=OFF, 1=ON
#define VALVE_OFFSET_CURRENT    0                   // valve offset voltage = V_offset / (Rseries + Rcoil)  s.t. valve is
closed
#define MAX_VOLTAGE             4.8                  // maximum valve voltage before saturation
#define MIN_VOLTAGE             -4.8               // minimum valve voltage before saturation
#define A(I,J)                  (*(a+(I)*n+(J)))   // used in MatVectMult
#define Pi                      3.141592653589793   // mysterious engineering constant?
#define g                       9.81               // gravity constant in Berkeley, CA is actually = 9.7994 via Helmert's
equation, lat=37.871 deg N, elev=12192cm
#define D_TO_A_CONVERSION       6553.6              // 16 bit DAC value to post  = DesiredVoltage x D_TO_A_CONVERSION
#define A_TO_D_CONVERSION       1.5258789e-4       // input voltage from ADC = 16 bit ADC value x A_TO_D_CONVERSION
#define RADIANS_PER_COUNT       1.5707963e-4       // 2*PI/40000  converts an encoder counter value to an angle (rads),
corrected by JRS 2004-07-08
#define MINIMUM_ANKLE_TORQUE    -10                  // default is -5.0 Nm
#define UCLOCKS_PER_MICROSECOND (1193180/1000000)  // = 1.19318 since UCLOCKS_PER_SEC = 1,193,180 uclock ticks/second


// Useful Substitutions, added by JRS
#define FALSE                   0
#define TRUE                    1
#define OFF                     0
#define ON                      1
#define IDLE                    0
#define RUNNING                 1
#define ZERO                    0.0
#define LEFT                    -1
#define NONE                    0
#define RIGHT                   1


// Control modes, added by JRS
#define STOP                    0
#define VALVE_CTL               1
#define MANUAL_TORQUE_CTL       2
#define AUTO_TORQUE_CTL         3
#define POSITION_CTL            4
#define EXIT                    5
#define KNEEE_LOCK_POSITION_CTL 6

#define DEFAULT_CTRL_MODE       STOP


// FHM Gains, added by JRS
#define SWING_LANKLE_KPFHM      0.5 //0.5 // 10/21/03, increased from 0.4 b.c ankle feels "sluggish"
#define SWING_LKNEE_KPFHM       0.5 //0.6 // 10/13/03, reduced from 0.7 b/c knee always locking in sstance/dstance
#define SWING_LHIP_KPFHM        0.5 //0.7

#define SWING_RANKLE_KPFHM      0.5 //0.5 // 10/21/03, increased from 0.4 b.c ankle feels "sluggish"
#define SWING_RKNEE_KPFHM       0.5 //0.6 // 10/13/03, reduced from 0.7 b/c knee always locking in sstance/dstance
#define SWING_RHIP_KPFHM        0.5 //0.7

#define DSTANCE_LANKLE_KPFHM    0.5 //0/0 // 10/14/03
#define DSTANCE_LKNEE_KPFHM     0.5 //0.7
#define DSTANCE_LHIP_KPFHM      0.5 //0.4

#define DSTANCE_RANKLE_KPFHM    0.5 //0.0
#define DSTANCE_RKNEE_KPFHM     0.5 //0.7
#define DSTANCE_RHIP_KPFHM      0.5 //0.4


// EXOSKELETON MASS AND GEOMETRY PROPERTIES
// MD  04/13/04 Entered values for EXO2 from excel spreadsheet
// JRS 05/13/04 -- EXO2 weighed with HPUE and harness = 58.606Kg
// 4kg unaccounted for in code, distributed to foot,shank,thigh,torso as 10%,25%,25%,40%
#define HEEL_LENGTH             0.154             // NOT USED
#define HEEL_LCG                1                 // NOT USED
#define HEEL_HCG                1                 // NOT USED
#define    HEEL_MASS            2.752             // NOT USED
#define    HEEL_INERTIA         0.019743         // NOT USED

#define FOOT_MASS               (2.752+0.2)         // foot mass properties exclude ankle bearing or actuator
#define FOOT_INERTIA            0.01924             // Marco  //0.019743 // solidworks
#define FOOT_LENGTH             0.234
#define FOOT_LCG                0.0561848         //0.052    // dist from ankle to foot CG
```

```
#define FOOT_HCG                -0.1015           //Marco //-0.1 // dist from ankle to foot CG

#define SHANK_MASS              (3.157864+0.5)    //Marco //3.33 // shank mass properties includes ankle bearings and
actuator
#define SHANK_INERTIA           0.043217        //Marco //0.035207506 // solidworks but excludes knee bearings and actuator
#define SHANK_LENGTH            (0.4088256 + 0.0127 * 5)    //Marco //0.446126
#define SHANK_LCG               (SHANK_LENGTH - 0.23970234) //Marco //0.23256086775767
#define SHANK_HCG               0.01829816        //Marco //-0.00343616931818

#define THIGH_MASS              (4.852727+0.5)    //Marco //5.12 // thigh mass properties includes knee bearings, knee and
hip actuators,
#define THIGH_INERTIA            0.085376        //Marco //0.0847828    // solidworks thigh actuator, but excludes hip
bearings.
#define THIGH_LENGTH            (0.4062476 + 0.0127 * 5)    //Marco 0.444348
#define THIGH_LCG               (THIGH_LENGTH - 0.20745704) //Marco 0.18589698678168
#define THIGH_HCG               -0.03298952        //Marco //0.00241894284148

#define UPPERBODY_MASS          (33+1.6);         // Soumma prefers 22kg, Adam prefers 21kg //27.625=HPUE // 28.399kg=HPU1
// 21.271kg (28.399kg w/o accum.) = upper body properties excludes hip bearings and actuators
#define UPPERBODY_INERTIA       0.38              //0.4648 // HPUE (estimated - not exact) // 0.38 = Erik Vaaler first gen
HPU (used as a dummy load while JLR was testing exo)
#define UPPERBODY_LENGTH        1                 // irrelevant, not used
#define UPPERBODY_LCG           0.216             //0.198    // HPUE (estimated - not exact) // 0.216
#define UPPERBODY_HCG           -0.24             //-0.27 // HPUE (estimated - not exact) // -0.24

#define TORSOSENSOR_L           0.3815107         // vertical distance of ATI F/T sensor from hip axis
#define TORSOSENSOR_H           -0.1396746         // horizontal distance ...

#define    HEEL_DIST            0.08             // distance spanned by each footswitch along the sole (meters)
#define    MIDFOOT_DIST         0.06
#define    BALL_DIST            0.08
#define    TOE_DIST             0.08


// SENSOR OFFSETS AND GAINS
// Inclinometer gains and offsets
// changed 2003-11-17 for RIOM#18 ADC1
// checked 2003-11-20 values are accurate
// changed 2004-01-23 for RIOM#24 ADC1
// changed 2004-04-07 for RIOM#24 ADC1 Serial #1420
// changed 2004-05-04 for RIOM#24 ADC1 Serial #1443, changed by JRS, 05-04-2004
#define TORSO_INCL_SLOPE        1.614127
#define TORSO_INCL_OFFSET       (-0.031681712-Pi)

// Encoder gains and offsets
#define LANKLE_ENC_OFFSET       (-0.240688984)
#define LKNEE_ENC_OFFSET        (0.9955358844)
#define LHIP_ENC_OFFSET         (-0.322813251)
#define LHIP_ROT_ENC_OFFSET     (0)
#define LHIP_ABD_ENC_OFFSET     (0)

#define RANKLE_ENC_OFFSET       (0.158999494)
#define RKNEE_ENC_OFFSET        (0.80791291)
#define RHIP_ENC_OFFSET         (-0.658163659)
#define RHIP_ROT_ENC_OFFSET     (0)
#define RHIP_ABD_ENC_OFFSET     (0)

#define FT_VIN                  5 // F/T backpack sensor excitation voltage (V)

// Accelerometer gains and offsets
#define LFOOT_ACC_GAIN1         0 //-13.27372808  //19.47342 // 28.77075355  // -13.27372808    // acc 1 (top) GAIN on left
foot
#define LFOOT_ACC_OFFSET1       0 //0.645202098  //-1.32859 // -1.921730811 // -0.017557893    // acc 1 (top) OFFSET on
left foot
#define LFOOT_ACC_GAIN2         0 //-12.99639     // 13.92141659 // -13.25782341
#define LFOOT_ACC_OFFSET2       0 //0.82843       // 0.87617763  // 0.816324531

#define LSHANK_ACC_GAIN1        13.82338359  //12.86893526
#define LSHANK_ACC_OFFSET1      0.8022025    //-0.008029123
#define LSHANK_ACC_GAIN2        12.84228722  //12.71297371
#define LSHANK_ACC_OFFSET2      0.828934818  //0.822634644

#define LTHIGH_ACC_GAIN1        5.156823125  //5.215242957
#define LTHIGH_ACC_OFFSET1        0.009413845  //-0.028138544
#define LTHIGH_ACC_GAIN2        12.8691386   //12.96252416
#define LTHIGH_ACC_OFFSET2        -0.004863989 //-0.019200056

#define RFOOT_ACC_GAIN1           -13.32914653 //13.89398865    // acc 1 (top) GAIN on right foot
#define RFOOT_ACC_OFFSET1       0.82356288   //-0.046432238    // acc 1 (top) OFFSET on right foot
#define RFOOT_ACC_GAIN2           -13.52444538 //13.22833281
#define RFOOT_ACC_OFFSET2       -0.005975934 //-0.027544142

#define RSHANK_ACC_GAIN1        12.86881188  //12.82398203
#define RSHANK_ACC_OFFSET1        -0.00641532  //-0.001453414
#define RSHANK_ACC_GAIN2        13.00980944  //12.97864312
#define RSHANK_ACC_OFFSET2        -0.064428743 //-0.058337783

#define RTHIGH_ACC_GAIN1        5.292352947  //5.238549419
#define RTHIGH_ACC_OFFSET1        -0.152364023 //-0.124433419
#define RTHIGH_ACC_GAIN2        13.28474856  //13.21278576
#define RTHIGH_ACC_OFFSET2        -0.047428344 //-0.037582237

#define UPPERBODY_ACC_GAIN1       0             //-5.014855712
#define UPPERBODY_ACC_OFFSET1   0             //-0.0685
#define UPPERBODY_ACC_GAIN2       0             //-5.086855069
#define UPPERBODY_ACC_OFFSET2   0             //-0.1685


// Force sensor gains (N/V) and offset (N) (includes effects of ADC1)
```

```
// using 4.448 N/lb
// updated for EX02 by JRS, 04-06-2004
//
// Left Ankle -- Serial # 912074
#define LANKLE_FSENSOR_GAIN        -448.5320354            //-452.0614633
#define LANKLE_FSENSOR_OFFSET      (14.18903011-68.1)      //-0.348880157
// Left Knee -- Serial # 912602
#define LKNEE_FSENSOR_GAIN         -485.6384051            //-411.872 //change new force sensor 010503 by Lihua //old one-
446.1908837
#define LKNEE_FSENSOR_OFFSET     (-26.74497988+176.1)      //-22.0352 //change new force sensor 010503 by Lihua  5.987991154
// Left Hip -- Serial # 912070
#define LHIP_FSENSOR_GAIN        -441.2164321             //-469.894
#define LHIP_FSENSOR_OFFSET         (-11.65476684-15.2)      //-8.8515
// Right Ankle -- Serial # 911379
#define RANKLE_FSENSOR_GAIN         -453.5430341            //-457.7128385   //-447.2173965 // RIOM 21 //-447.5595674 //
RIOM 20
#define RANKLE_FSENSOR_OFFSET     -39.96462659            //-38.69173895-57.90  //1.240369596  // RIOM 21 //1.069284135  //
RIOM 20
// Right Knee -- Serial # 913386
#define RKNEE_FSENSOR_GAIN         -458.3500874            //-455.1577775           //-450.522834
#define RKNEE_FSENSOR_OFFSET     -43.8342088            //  -2.311331039-19.6+2.4 //-10.49434899
// Right Hip -- Serial # 913390
#define RHIP_FSENSOR_GAIN        -488.6957159            //-483.8260834 //-396.7892063
#define RHIP_FSENSOR_OFFSET         -28.50864459           //-22.19428661+3.5+1.5  //


// DAC GAINS AND OFFSETS // RIOM #
// updated 2003-NOV-13
// NOTES: 1) ankle RIOMs and knee RIOMs were switched 2003-DEC-1
//        2) RHIP RIOM 2 has no footswitch connector, 2004-07-28
#define LANKLE_DAC_GAIN            1.0128 // RIOM 14
#define LKNEE_DAC_GAIN           1.0017 // RIOM 7
#define LHIP_DAC_GAIN             1.0121 // RIOM 20

#define RANKLE_DAC_GAIN          1.0165 // RIOM 6, swapped with RHIP RIOM 2 on 2004-07-28
#define RKNEE_DAC_GAIN           1.0195 // RIOM 16
#define RHIP_DAC_GAIN            0.9999 // RIOM 2, swapped with RANKLE RIOM 2 on 2004-07-28

#define LANKLE_DAC_OFFSET         0.0138 // RIOM 14
#define LKNEE_DAC_OFFSET         0.0108 // RIOM 7
#define LHIP_DAC_OFFSET           -0.0248 // RIOM 20

#define RANKLE_DAC_OFFSET        -0.0342 // RIOM 6
#define RKNEE_DAC_OFFSET         0.0208 // RIOM 16
#define RHIP_DAC_OFFSET             0.0391 // RIOM 2

// valve offsets (V) to center spool and get zero flow
#define LANKLE_VALVE_OFFSET      0.0054583330    //  0.0050
#define LKNEE_VALVE_OFFSET        -0.0220           // -0.0210
#define LHIP_VALVE_OFFSET         0.0622           //  0.0522
#define RANKLE_VALVE_OFFSET       -0.021125
#define RKNEE_VALVE_OFFSET        -0.042625
#define RHIP_VALVE_OFFSET         0.0375

// JOINT STIFFNESS SLOPES AND OFFSET
#define     ANKLE_STIFFNESS_SLOPE_SLOPE               0.8528    // the slope of the ankle stiffness velocity slope
#define ANKLE_STIFFNESS_SLOPE_OFFSET            1.3498    // the offset of the ankle stiffness velocity slope
#define     ANKLE_STIFFNESS_OFFSET_POS_VEL           -1.55    // the slope of the ankle stiffness offset when vel > 0
#define ANKLE_STIFFNESS_OFFSET_NEG_VEL           -2.5    // the offset of the ankle stiffness offset when vel < 0

#define     KNEE_STIFFNESS_SLOPE_SLOPE               -0.3148    // the slope of the knee stiffness velocity slope when
angle < 1.5 rad
#define KNEE_STIFFNESS_SLOPE_OFFSET               0.3798    // the offset of the knee stiffness velocity slope when angle
< 1.5
#define     KNEE_STIFFNESS_OFFSET_SLOPE               -0.5546    // the slope of the knee stiffness velocity offset when
angle < 1.5
#define KNEE_STIFFNESS_OFFSET_OFFSET            1.3069    // the offset of the knee stiffness velocity offset when angle <
1.5
#define     KNEE_STIFFNESS_SLOPE_A2                   10.932    // velocity polyn coeffs of knee stiffness vs. angle slope
when angle > 1.5
#define     KNEE_STIFFNESS_SLOPE_A1                   10.167
#define     KNEE_STIFFNESS_SLOPE_A0                    2.1762
#define     KNEE_STIFFNESS_OFFSET_A2                -17.264    // velocity polyn coeffs of knee stiffness vs. angle offset
when angle > 1.5
#define     KNEE_STIFFNESS_OFFSET_A1                -17.49
#define     KNEE_STIFFNESS_OFFSET_A0                 -1.7733

#define     HIP_STIFFNESS_SLOPE_SLOPE                -2.0494    // the slope of the hip stiffness velocity slope
#define HIP_STIFFNESS_SLOPE_OFFSET               3.4218    // the offset of the hip stiffness velocity slope
#define     HIP_STIFFNESS_OFFSET_SLOPE_POS_VEL        -0.5087    // the slope of the hip stiffness velocity offset for vel>0
#define HIP_STIFFNESS_OFFSET_OFFSET_POS_VEL       1.7525    // the offset of the hip stiffness velocity offset for vel>0
#define     HIP_STIFFNESS_OFFSET_SLOPE_NEG_VEL        -2.1719    // the slope of the hip stiffness velocity offset for vel<0
#define HIP_STIFFNESS_OFFSET_OFFSET_NEG_VEL       -0.3958    // the offset of the hip stiffness velocity offset for vel<0


// MAX DATA VALUES, USED FOR ERROR CHECKING
#define ANKLE_MIN                -0.80    // -0.7853982           /* joint limits in rad */
#define ANKLE_MAX                 0.78    // 0.7853982

#define KNEE_MIN                 0.10    // 0.08726646
#define KNEE_MAX                 2.199115

#define HIP_MIN                    -2.00    // -2.007129
#define HIP_MAX                     0.35    // 0.1745329

#define HIP_ABD_MIN              -3.14    // temp
#define HIP_ABD_MAX               3.14    // temp

#define HIP_ROT_MIN              -3.14    // temp
```

```
#define HIP_ROT_MAX                3.14      // temp

#define TORSO_MIN                -3.141593    // torso tilt angle limit in rad
#define TORSO_MAX                 3.141593

#define MAX_VELOCITY_SAT         30.00  // velocity is saturated at this value in software
#define MIN_VELOCITY_SAT        -30.00

// Define min/max offsets for soft stops
#define ANKLE_MIN_SOFT_OFFSET    0.189193  // result = -35deg
#define ANKLE_MAX_SOFT_OFFSET    0.169165  // result = +35deg

#define KNEE_MIN_SOFT_OFFSET    0.0745329 // result = +10deg
#define KNEE_MAX_SOFT_OFFSET    0.279254  // result = +110deg

#define HIP_MIN_SOFT_OFFSET       0.429204  // result = -90deg
#define HIP_MAX_SOFT_OFFSET       0         // result = +20deg

#define VMAX                    1000    // max joint velocity rad/s
#define AMAX                    10000    // max joint acceleration rad/s2
#define FMAX                    2300    // max actuator force N  (1779.289 N = 400 lb; 2000 N = 450lb )
#define FMAXSOFT                1850    // max force used to decide when to stop integrating force error in JointCtl.c
#define FTX_MAX                    495        // max torso sensor x-force
#define FTY_MAX                    165        // max torso sensor y-force
#define TMAX                    15        // max torso sensor torque

#define LONG_PISTON_POS_MIN       -0.05044584189365  // min and max piston positions for the ankle and hip actuators (m)
#define LONG_PISTON_POS_MAX       0.07655415810635

#define SHORT_PISTON_POS_MIN    -0.04835667351492  // min and max piston positions for the knee actuator (m)
#define SHORT_PISTON_POS_MAX    0.06124332648508

#define ANKLE_MOMENT_ARM_MIN    0.04518882740326
#define ANKLE_MOMENT_ARM_MAX    0.09046879619931

#define KNEE_MOMENT_ARM_MIN       -0.04780871462969
#define KNEE_MOMENT_ARM_MAX       -0.00182785957733

#define HIP_MOMENT_ARM_MIN       0.01076198253385
#define HIP_MOMENT_ARM_MAX       0.06386202913989

#define FOOT_DISTANCE_MAX        1.5  // max foot distance from CG in the transverse plane (m)
#define SGMAX                    4    // max straingauge output (V)
#define THERM_MAX                4    // max thermister output value
#define THERM_MIN                4    // max thermister output value

#define QROT0                    0.78  // Absolute angle difference between the two hip rotations at which rotation
                                 //    horizontal force factor is zero
#define QROT1                    0.52  // Absolute angle difference between the two hip rotations at which rotation
                                 //    horizontal force factor starts to decrease
#define KROT_OFFSET               2.99
#define KROT_SLOPE               (-3.8197)  // = 1/(QROT1-QROT0)

#define LABD                    0.1397  // (m) lateral distance from hip flexion/rotation axis to hip abduction axis
#define DGUB                    0.1016  // (m) lateral distance from hip abduction axis to upper body CG


// DEFINE INDICES
#define JUMP            0 // define dynamic mode (i.e. state) indices
#define    SSTANCE           1
#define DSTANCE           2
#define ONE_REDUNDANCY    3
#define TWO_REDUNDANCY    4

#define LANKLE_T    0 // actuator torque indices
#define LKNEE_T       1
#define LHIP_T       2
#define RANKLE_T    3
#define RKNEE_T       4
#define RHIP_T       5

#define ANKLE_T       0
#define KNEE_T       1
#define HIP_T       2

#define TORSO_FX    0
#define TORSO_FY    1
#define TORSO_T     2

#define LTOE        0 // kinematic data array indices
#define LANKLE        1
#define LKNEE        2
#define LHIP        3
#define RTOE        4
#define RANKLE        5
#define RKNEE        6
#define RHIP        7
#define LHIP_ROT    8
#define LHIP_ABD    9
#define RHIP_ROT    10
#define RHIP_ABD    11

#define LFOOT        0 // define body segment array indices
#define LSHANK       1
#define LTHIGH       2
#define RFOOT        3
#define RSHANK       4
#define RTHIGH       5
#define UPPERBODY    6
```

```
#define    TIP            0 // define footswitch array indices
#define TOE            1
#define BALL        2
#define MIDFOOT        3
#define HEEL        4

#define DL                        0 // define load distribution data array indices
#define DR                        1
#define HIP_ROTATION_FACTOR        2
#define WEIGHT_DISTR_FACTOR        3

// DEFINE INDICES FOR TRIG ARRAY
#define C2            0 // non-redundant leg trig array indices
#define S2            1
#define C23            2
#define S23            3
#define C234        4
#define S234        5
#define C3            6
#define S3            7
#define C4            8
#define S4            9

#define C1RD        0 // redundant leg trig array indices
#define S1RD        1
#define C12RD        2
#define S12RD        3
#define C123RD        4
#define S123RD        5
#define C1234RD        6
#define S1234RD        7
#define C2RD        8
#define S3RD        9
#define C23RD        10
#define C3RD        11
#define C4RD        12
#define S4RD        13

#define FOOT_ACC_DIST_INV        9.544 // (new rotation foot, 01-27-04) 6.85 (old foot)  // inverse of the distance between
accelerometers on each body segment
#define SHANK_ACC_DIST_INV        4.46  // (1/m)
#define THIGH_ACC_DIST_INV        4.61
#define UPPERBODY_ACC_DIST_INV    4.63

// DEFINE INDICES FOR HYDRAULIC PARAMETER ARRAY
#define XV            0
#define H2            1
#define O2            2
#define P2            3

// DEFINE ARRAY INDEICES FOR filtercoeffs.
#define A1            0
#define A2            1
#define A3            2
#define B2            3
#define B3            4


// DEFINE PHYSICAL CONSTANTS
// see hydraulicParameters.m
//
#define Ap1            2.8502e-4 // cylinder bore area (m2)
#define Ap2            2.1377e-4 // EX02, modified 2004-01-07 for 3/8" rod, old value 2.3554e-4, cylinder rod area (m2)
#define LA_ANKLE    0.31777003914252        // LA = sqrt(a^2 + b^2), geometric parameters for cylinder
#define LA_KNEE        0.31044301899146
#define LA_HIP        0.07438905527978
#define LB_2_ANKLE    0.00922587723853        // LB^2 = c^2 + d^2
#define LB_2_KNEE    0.00403376949374
#define LA_2_HIP    0.005533731584
#define M_ANKLE        -0.11020367501517        // M = -LA^2 - LB^2
#define M_KNEE        -0.10048863753427
#define M_HIP        -0.12533624472366
#define N_ANKLE        -0.06184453345237        // N = -2 LA LB
#define N_KNEE        -0.03943369131414
#define N_HIP        -0.05149582299157
#define LC_ANKLE    -0.09175192053811        // LC = LB^2 - LA^2
#define LC_KNEE        -0.09234109854680
#define LC_HIP        0.11426878163282
#define PHIA_ANKLE  0.20118446034160        // PHIA = atan a/b
#define PHIA_KNEE    0.13418383184850
#define PHIA_HIP    0.87306885403296
#define PHIB_ANKLE    1.21552307644741        // PHIB = atan d/c
#define PHIB_KNEE    0.78539816339745
#define PHIB_HIP    0.14729987726891
#define V0_ANKLE    1.55133923734404e-5 //EX02, rod dia change, chamber volume when areas on both sides of the piston are
equal (m3)
#define V0_KNEE        1.24107138987523e-5 //EX02, rod dia change,
#define V0_HIP        1.55133923734404e-5 //EX02, rod dia change,
#define IMAX        25e-3                // max input current to valve (A)
#define UMAX        4.5                    // max input voltage to the valve (V) note: max possible is 5V
#define XVMAX        328.5e-6            // 219e-6 for 3V; 328.5e-6 with 4.5;    373e-6  for 5V;  max spool travel on one
side (m)
#define R_VALVE        200                    // valve coil + series resistance on RIOM (2x68ohms on RIOM + 80ohms on
valve)

#define PS            6.8948e6    // supply pressure (Pa) = 1000 PSI
//#define PS        6.5500e6    // supply pressure (Pa) = 950 PSI (never tested)
//#define PS        6.2053e6    // supply pressure (Pa) = 900 PSI (never tested)
```

```
//#define PS        5.8606e6    // supply pressure (Pa) = 850 PSI (never tested)
//#define PS        5.5158e6    // supply pressure (Pa) = 800 PSI (never tested)
//#define PS        5.1712e6    // supply pressure (Pa) = 750 PSI (walking ok/sluggish, but knee forces are close to
saturation)
//#define PS        4.8263e6    // supply pressure (Pa) = 700 PSI (never tested)
//#define PS        4.4816e6    // supply pressure (Pa) = 650 PSI (never tested)
//#define PS        4.1369e6    // supply pressure (Pa) = 600 PSI (never tested)
//#define PS        3.7921e6    // supply pressure (Pa) = 550 PSI (never tested)
//#define PS        3.4474e6    // supply pressure (Pa) = 500 PSI (valves saturate for knee and hip every step, exo cannot
support its own weight, walking does not feel significantly better than unpowered)
//#define PS        1.7237e6    // supply pressure (Pa) = 250 PSI (never tested)

#define BETA        6.9e8        // old beta was 1.517e9  // effective fluid bulk modulus (Pa) (other sources: meritt 6.9e8
,1e9, 1.517e9 Pa)
                                 // smaller BETA gives better tracking at high amplitude (1e8 works well w/ smaller lambda
in FLin)

#define TAU           0.0015        // valve mechanical time constant (sec)
#define KS            0.0146        // valve DC gain (m/A)
#define KS_INV        68.493        // 1/KS  (A/m)
#define G3            9.7333        // KS / TAU
#define G3_INV        0.10274        // 1/G3
#define F3            -666.67        // - 1/ TAU
#define GAMMA        (2.16282e-4*BETA)    // with old beta was 327981   // Cd W Beta /sqrt(RO)
                                          // Cd = valve discharge coefficient
                                          // W = valve orifice area gradient (m)
                                          // RO =  fluid density (kg/m2)

#define SI            (4e-14*BETA) // value with old Beta 6.068e-5 // 2 Beta Ct
                                   // Ct = cylinder leakage coefficient ( m3/sec/Pa)

#define L0_PLUS_XP0_ANKLE    0.32981537832700 // (m)
#define L0_PLUS_XP0_KNEE    0.29292260266160 //    XP0 = piston position when area on both sides is equal (m)
#define L0_PLUS_XP0_HIP        0.32981537832700 //    L0  = cylinder dead length (m)


// define maximum additive and multiplicative uncertainty parameters for robust valve controllers
#define G2_TILDA_MIN        0.8
#define G2_TILDA_MAX        1.2
#define DELTA_BETA_MAX        (0.2*BETA)
#define DELTA_SI_MAX        (0.2*SI)
#define G3_TILDA_MIN        0.8
#define G3_TILDA_MAX        1.2
#define G3_TILDA_MAX_INV    0.83333  // 1/G3_TILDA_MAX
#define DELTA_F3_MAX        (-0.1*F3)


// define weight, mass and geometry of hydraulic components
#define W_ANKLE_ACT_SENSOR    14.1264  // = M_act g  (N)
#define W_KNEE_ACT_SENSOR    22.7592
#define W_HIP_ACT_SENSOR    6.867
#define RCG_ANKLE            0.239595
#define RCG_KNEE            0.241548
#define RCG_HIP                0.19605

// define torso F/T sensor constants
#define CFT11        134.1971323        // sensor calibration matrix
#define CFT12        -4.000547243    //  Fz row
#define CFT13        135.3017459
#define CFT14        -1.560794287
#define CFT15        136.2343085
#define CFT16        -3.581443361

#define CFT21        -0.568199187    // Fx row
#define CFT22        -0.986065476
#define CFT23        1.231591581
#define CFT24        -86.98476572
#define CFT25        -2.549988326
#define CFT26        85.88848683

#define CFT31        5.399278947        // Ty row
#define CFT32        -0.163992174
#define CFT33        -2.71626536
#define CFT34        1.079020558
#define CFT35        -2.648369989
#define CFT36        -0.939413716

#define BT1            0.13            // temperature compensated FT sensor voltage bias value vector (B x Vin in ATI
manual)
#define BT2            -0.165
#define BT3            0.387
#define BT4            0.1875
#define BT5            0.25
#define BT6            0.537

#define CT            5.781661486        // FT sensor thermister value at calibration (V, Amplified) twe. xls
#define GS            0.824382645        // FT sensor thermister gain slope (1/V, Amplified) twe.xls
```

Appendix A.2 – ExoMain.h

```
/* STRUCTURE DEFINITIONS */

typedef struct {
        double mass;            /* kg */
        double inertia;         /* about segment CG - kg.m2 */
        double length;          /* m */
        double Lcg;             /* axial distance from distal joint to segment CG - m */
        double hcg;             /* perpendicular distance from distal joint to segment CG - m */
} SegmentDataT;

typedef struct {
        SegmentDataT heel;
        SegmentDataT foot;
        SegmentDataT shank;
        SegmentDataT thigh;
        SegmentDataT upperBody;

        double torsoSensor_L;    // distance along upperbody to torso sensor
        double torsoSensor_h;    // distance orthogonal to upper body to torso sensor
} BodyDataT;

typedef struct {
        double sensorForce;     /* Cylinder force sensor reading - N */
        double position;        /* rad */
        double velocity;        /* rad/s */
        double acceleration;    /* rad/s2 */
        double momentArm;       // m
        double pistonPosition; // piston distance from xp0 reference position (m)
        double pistonVelocity; // m/s
        double torque;          // actuator torque N.m */
        double Tdes;            /* Desired Joint Torque vector */
        double Thm;             // Joint Torque of human on machine (N.m)
        double Tg;              // Joint Torque needed to compensate gravity (N.m)
        double Tcc;               // Joint torque needed to compensate centrifugal and coriolis forces (N.m)
        double Tf;               // Joint friction torque - includes hosing and cable stiffness (N.m)
        double Tlin;               // Linearizing torque Tlin = Tg + Tcc + Tf (N.m)
        double Tinertial;       // Torque due to inertial forces, JRS, 2004-06-24
        double Tvguard;         /* Virtual guard torque */
        double Tvlimit;         /* Virtual limit torque */
        double valveVoltage;    /* input voltage on the valve */
        double indexPulse;      // encoder index pulse
        int    againstStop;
} JointDataT;

typedef struct {      // unactuated joint angles
        double R_abduction;
        double L_abduction;
        double R_rotation;
        double L_rotation;
        double R_abduction_indexP;        // index Pulse
        double L_abduction_indexP;
        double R_rotation_indexP;
        double L_rotation_indexP;
} HipDataT;


typedef struct{
        double angular_accel;   // ang acceleration of body
        double lin_accel1;              // lin accelerometer 1 data
        double lin_accel2;              // lin accelerometer 2  data
        double offset1;                 // lin accel 1 offset
        double offset2;                 // lin accel 2 offset
        double gain1;                   // lin accel 1 gain
        double gain2;                   // lin accel 2 gain
}BodyAccelT;

typedef struct{
        int Mode;       /* 0:jump; 1:sgl sup; 2:dbl sup; 3: dbl sup sgl redundancy; 4:dbl sup bl red */
        int redundantLeg;
        int prevRedundantLeg;
        int groundedLeg;
        int prevGroundedLeg;
        int leftHeelContact;
        int rightHeelContact;
        int LstanceSwingTransition; // 1 if transitioning left leg from stance to swing
        int RstanceSwingTransition; // 1 if transitioning right leg from stance to swing
        int LstateTransition; // 1 if left leg is transitioning b/w states
        int RstateTransition; // 1 if right leg is transitioning b/w states
        int prevDynMode;
        int prevLeftLegStance;
        int prevRightLegStance;
        int prevLeftKneeControlType;  // 0:STOP; 1:manual voltage; 2:manual torque; 3:auto torque (MSS); 4:position
control; 5:ERROR
        int prevRightKneeControlType; // 0:STOP; 1:manual voltage; 2:manual torque; 3:auto torque (MSS); 4:position
control; 5:ERROR
        int lockingKnee;
} DynamicModeT;

typedef struct{
        double SG[6];    // torso F/T sensor straingauge outputs (V)
        double SGt[6];    // torso F/T sensor temperature-compensated straingauge outputs (V)
        double thermister; // thermister output (V)
        double Fx; // horizontal force (N)
        double Fy; // vertical force (N)
        double T;  // sagittal plane moment (N.m)
```

```
        double SGt_bias[6];  // temperature compensated straingauge bias (V)
} TorsoForceT;

typedef struct{
        double LankleDistance; // transvers plane distance from CG to ankles
        double RankleDistance;
        double weightDistrFactor; //  force factor due to gravity (alpha)
        double filteredBetaFg[4];    // last 4 elements of the filtered load distribution factor Beta (A[0] = most recent)
        double unfilteredBetaFg[4];           // ... unfiltered ....
        double filteredBetaFHM[4];
        double unfilteredBetaFHM[4];
        double filteredKrot[4];        // last 4 elements of filtered horizontal force hip rotation factor Krot
        double unfilteredKrot[4];   // ... unfiltered ...
} ForceDistributionT;

typedef struct {
        JointDataT jointData[8];         /* arrays are as follows: [Ltoe Lankle Lknee Lhip Rtoe Rankle Rknee Rhip ] */
        HipDataT hipData;
        BodyAccelT bodyAccel[7];      // [ LFOOT LSHANK LTHIGH RFOOT RSHANK RTHIGH UPPERBODY]
        DynamicModeT dynamicMode;
        TorsoForceT torsoForce;  // torso force sensor data
        ForceDistributionT forceDistribution;

        double TorsoTilt;                /* rad */
        double torsoVelocity;
        double virtualGuardFx; // horizontal force caused by virtual guard at upper body CG
        double virtualGuardT; // hip moment cause by virtualGuardFx

        double lockingKneeDesiredPosition;

        int Rfootswitch[5];             /* 0: off; 1: on */
        int Lfootswitch[5];
        int error;      // error = 1: an incoming OMNIBUS sensor signal is out of range; error = 0: no error
        int lostCommunication; // number of times communication with PCI was lost
        int loopPeriod;        // Supervisor loop period
        int GUIping;    // GUI ping response
        int calibrationFlag;  // indicates current controller state of accelerometer calibration
        unsigned long  int CounterTicks;     // 32bit counter used to send time stamp to GUI
} SensorDataT;

typedef struct {
        int virtualGuard;   // toggle the virtual guard
        int virtualLimits;  // toggle the virtual limits
        int controlFHM;     // toggle the human-machine force control
} TorqueControlT;

typedef struct{
        int OperationMode;       // 0:idle; 1:on
        int jointControlType;    // 0:manual voltage, 1:position control, 2:manual torque control (MSS), 3:velocity
control, 4:VFC
        int prevJointControlType; // 0:manual voltage, 1:position control, 2:manual torque control (MSS), 3:velocity
control, 4:VFC
        double kp;                // proportional gain
        double kv;                // derivative gain
        double lambda1;           // joint controller gains
        double lambda2;
        double Ci;                 // Force tracking error integrator gain
        double CiSwitch;
        double eta1;                  // sliding surface robustness parameters
        double eta2;
        double phi1_inv;          //   1/(sliding surface boundary layers)
        double phi2_inv;
        double ro1;                 // Lyapunov function gains
        double ro2_inv;           // 1/ro2
        double manualTorque;      // User-specified joint torques when the controller is on Manual Torque Control
        double manualValveInput;  // User-specified valve input voltage when the controller is on Valve Input Control
        double desiredVelocity;   // (rad/s) used in velocity control for friction calibration
        double desiredPosition;   // (rad) used in position control for friction, stiffness and mass calibration
        double frequency;         // frequency for manual torque, valve input and desired velocity sinusoidal signals
        double amplitude;         // amplitude for manual torque, valve input and desired velocity sinusoidal signals
        double kpFHM;             // human-machine force amplification gain
} JointControlT;

typedef struct {
        double activation;
        double position;  /* horizontal position of the VGuard's activation zone */
        double saturation; /* max force */
        double stiffness;
        double damping;
} VirtualGuardT;

typedef struct{
        int GUIflag;                    // indicates current GUI state of accelerometer calibration
        int selection[7];  // binary values to select  accelerometers to be calibrated
                                                // [ LFOOT LSHANK LTHIGH RFOOT RSHANK RTHIGH UPPERBODY]
} CalibrationT;

typedef struct{
        int      activateAnkleSpring;
        double springRate;
        double centerAngle;
        int      activateKneeDamper;
        double flexionDampingCoeff;
        double extensionDampingCoeff;
        int   test3;
        int   test4;
} DebuggingControlsT;

typedef struct {
```

```
        TorqueControlT torqueControl;
        JointControlT  jointControl[6];  /* [ankleL kneeL hipL ankleR kneeR hipR] */
        VirtualGuardT  virtualGuard;
        VirtualGuardT  virtualLimit;
        CalibrationT   calibration;

        double          DynDistributionFactor; // load distribution factor for double support
        double          ditherAmplitude;  // amplitude of valve dither voltage (V)

        int            mainOperationMode;   /* 0:stop; 1:valve control; 2:manual torque control; 3:auto torque control; 4:
vel. control */
        int               GUIping;        // ping signal from GUI
        int            ditherFrequency;    // valve dither frequency (Hz)
        int            recordFlag;
        int               resetTimer;
        DebuggingControlsT debuggingControls;
} SysPropertiesT;

typedef struct{
    JointDataT           jointData[8];
    JointControlT       jointControl[6];
    unsigned long int    CounterTicks;
    DynamicModeT        dynamicMode;
    double               TorsoTilt; // rad
    int                  Rfootswitch[5]; // 0=OFF, 1=ON
    int                  Lfootswitch[5];
} RecordedDataArrayT;


/* ------------------------------------------------------------------------
 *  FUNCTION PROTOTYPES
 * ------------------------------------------------------------------------*/


/* Function: InitBodyData
 *----------------------------------
 * This function intializes the bodyData structure (body length and mass properties). These values cannot be
 * modified by the GUI.
 */
void InitBodyData(BodyDataT *bodyData);

/* Function: InitSysProps
 *----------------------
 * This function intializes the sysProperties structure. These values can be modified in the GUI.
 */
void InitSysProps(SysPropertiesT *sysProperties);

/* Function: InitSensorData
 *----------------------
 * This function intializes the sensorData structure. Most of these values are displayed in the GUI.
 */
void InitSensorData(SensorDataT *sensorData);

/* Function: Supervisor
 * --------------------
 *      Main system controller. Collects sensor data and implements the control of all machine joints according
 * to the main operation mode (0:stop; 1:valve control; 2: manual torque control; 3: auto torque control).
 */
void Supervisor(long           bufaddr,
                BodyDataT      *bodyData,
                SysPropertiesT *sysProperties,
                SensorDataT    *sensorData);

/* Function: PositionControl
 * -------------------------
 * Apply joint torque control for position control. Position for each joint is given in sysProperties
 * Required torque is obtained with proportional control and stored in 'desiredtorques'
 */
void PositionControl(double               desiredPosition,
                     SensorDataT          *sensorData,
                     const SysPropertiesT *sysProperties,
                     int                  i);

/* Function: SysTorqueController
 * -----------------------------
 * Apply joint torque control for human-machine force control, Virtual guard and virtual limit stop modes.
 * desiredTorques is a pointer to the 6 element array [Lankle Lknee Lhip Rankle Rknee Rhip] containing
 * the resulting desired control torques.
 */
void SysTorqueController(double           *desiredTorque,
                         const BodyDataT *bodyData,
                         SensorDataT     *sensorData,
                         SysPropertiesT  *sysProperties);

/* Function: GetVGuardForces
 * -------------------------
 * Computes the required machine operational force to produce the virtual guard potential field. This potential field
 * will redirect the machine outside of the virtual guard activation region and will prevent a possible
 * loss of stability.
 */
void GetVGuardForces(SensorDataT         *sensorData,
                     const BodyDataT     *bodyData,
                     const VirtualGuardT vguard,
                     const double         xR,
                     const double         xL,
                     const double         c1234,
                     const double         s1234);

/* Function: GetVLimitsTorques
```

```
 * ----------------------------
 * Computes the required machine torques to produce the virtual joint limit stops. The virtual joint limit
 * stop is a potential field which is activated when the machine enters a predefined region near the hardware
 * stops. The potential field is produced by a virtual spring-damper system and will prevent the joints from
 * reaching their physical limits.
 */
void GetVLimitsTorques(const SysPropertiesT *sysProperties,
                       SensorDataT          *sensorData);

/*      Function: GetDynMode
 * ------------------------------------------------------------------------
 *      Establishes the dynamic mode of the machine according to footswitch data.
 *      Mode accepts the following values:
 *              - jump mode:       0
 *              - single support:  1
 *              - double support:  2
 *              - double support single redundancy: 3
 *              - double support double redundancy: 4
 *  LeftIsGrounded is 0 or 1
 *  LeftIsRedundant is 0 or 1
 *  LeftHeelContact is 0 or 1
 *  RightHeelContact is 0 or 1
 */
//void GetDynMode(SensorDataT *sensorData);
void GetDynMode(SensorDataT          *sensorData,
                const SysPropertiesT *sysProperties);  //note: added sysProperties 08-13-03

/* Function: GetFootStatus
 * ---------------------------
 * Returns the foot status according the footswitch readings.
 * The returned value is as follows:  no contact:0; heel contact:1; toe contact:2; flat:3;
 */
int GetFootStatus(const int *footswitch);

/* Function: SinusoidalSignal
 * ----------------------
 * Makes a sinusoidal signal out of a nominal data, and offset, an amplitude and a frequency.
 */
double SinusoidalSignal(double         nominalSignal,
                        JointControlT jointControl,
                        int           i);
```

Appendix A.3 – ExoMain.c

```
#ifndef DJGPP
    #define UCLOCKS_PER_SEC    1 // Visual C does not define this value
    #define uclock_t clock_t  // Visual C does not support uclock_t
     #define uclock clock      // Visual C does not support uclock()
#endif

//#include <iostream.h>
#include <math.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "ExoMain.h"
#include "Defines.h"
#include "FHM.h"
#include "Sensors.h"
#include "JointCtl.h"
#include "PCI.h"
#include "DSup.h"
#include "Record.h"
#include "Filters.h"

// change filter coefficients based on control loop update
//#include "Filt1k.h"   //  1.0 kHz
//#include "Filt1.4k.h"    //  1.4 kHz
#include "Filt2k.h"     //  2.0 kHz
//#include "Filt5k.h"   //  5.0 kHz
//#include "Filt10k.h"  // 10.0 kHz

long loopCter                       = 0; // keep track of number of Supervisor task runs
int  counterResetFlag               = FALSE; // indicates when the counter is reset
int  recordedDataLoopCounter        = 1; // indicates when to record data on exo cpu by counting loop cycles, added by
JRS, 2004-06-10
int  totalRecordedDataPoints        = 0; // keeps track of the number of data points recorded locally on exo cpu, added by
JRS, 2004-06-10
int  numPointsWrittenToFlash        = 0; // used in loop to write data to compact flash, added by JRS, 2004-06-10
int  recordedDataLoopCounterRollover = 0; // used to set frequency for recording data locally, added by JRS, 2004-06-10
int  recordDoneMessageDisplayed     = FALSE; // added by JRS, 2004-06-10
FILE *FileArray[NUM_RECORDED_FILES]; // currently = 7, JRS, 2004-10-28


/* Function: main
* ------------------------------------------------------------------------------------------
*/
int main(void){

    uclock_t            startT;
    uclock_t            startTcounter;
    BodyDataT            bodyData;
    SysPropertiesT       sysProperties;
    SensorDataT          sensorData;
    int                  switch_drequestDone;
    unsigned int        bus          = 0; // constants for PCI communication
    unsigned int        devfun       = 0;
    unsigned int        vendor_id    = 0;
    unsigned int        device_id    = 0;
    int                    iobase        = 0;
    short                loopPeriod   = 0;
    long                bufaddr      = 0;
    int                    stopProgram  = 0;
    short                shortCounter = 0;
    int                    localCount   = 0;
    RecordedDataArrayT    *recordedDataArray;

    recordedDataArray = (RecordedDataArrayT *) calloc(MAX_RECORED_DATA_POINTS, sizeof(RecordedDataArrayT));
    printf("\n\nSize of RecordedDataArray in bytes = %d\n\n",sizeof(RecordedDataArrayT));

    recordedDataLoopCounterRollover = (int) (FREQ/RECORDED_DATA_FREQ);
    printf("\nrecordedDataLoopCounterRollover = %d\n\n",recordedDataLoopCounterRollover);

    InitBodyData(&bodyData);       // initialize bodyData structure
    InitSysProps(&sysProperties); // initialize sysProperties structure
    InitSensorData(&sensorData);  // initialize sensorData structure

    iobase = InitComm(&bufaddr, &sensorData); // wait for PCI communication to initialize and send address and size
information
                                          // = -1 if no PCI device was found

    startT          = uclock(); // Visual C uses "clock" and DJGPP uses "uclock"
    startTcounter = uclock(); // this counter is used for measuring the loop time (should = 500us)

    //    printf("\nBegin test loop...\n");
    //    while( !kbhit()  /*stopProgram */){
    //        // do something here...
    //    }
    //    printf("\nTest loop completed.\n");

    // ----------------------SUPERVISOR LOOP STARTS HERE--------------------------
    while( !kbhit()  /*stopProgram */){ // start the GUI communication loop

        startT = uclock();

        counterResetFlag = FALSE; // following 4 lines are done for the SinusoidalSignal function.
        if( loopCter == 10000 ){ // when count gets high
```

```
            loopCter = 0; // reset counter to avoid rollover
            counterResetFlag  = TRUE; // flag to indicate that the counter was reset
        }

        if (sysProperties.resetTimer == TRUE){
            sensorData.CounterTicks = 0; // reset data recording timer
        }

        // Wait for communication flag to run loop, this flag comes from the Supervisor I/O board
        if (WaitForCounterUpdate(bufaddr, &sensorData)){
            sensorData.CounterTicks++;
            Supervisor(bufaddr, &bodyData, &sysProperties, &sensorData); // Run the system supervisor controller
        }else{
            sysProperties.mainOperationMode = 99; // stop program if communication has been lost
        }

        loopCter++; // update counter

        // local data recording by exo cpu, added by JRS, 06-10-2004
        if (sysProperties.recordFlag == TRUE) {
            if ((recordedDataLoopCounter >= recordedDataLoopCounterRollover) && (totalRecordedDataPoints <=
MAX_RECORED_DATA_POINTS)){
                RecordDataToMem(&recordedDataArray[totalRecordedDataPoints], &sysProperties, &sensorData, &bodyData);
                recordedDataLoopCounter = 1;
                totalRecordedDataPoints++;
            }
            recordedDataLoopCounter++;
        }

        if (totalRecordedDataPoints >= MAX_RECORED_DATA_POINTS){
            if (sysProperties.recordFlag == TRUE && !recordDoneMessageDisplayed){
                recordDoneMessageDisplayed = TRUE;
                printf("\n\nRECORDING DONE!\n\n");
            }

            sysProperties.recordFlag = FALSE; // use this line to record for a fixed time period and then stop
            //totalRecordedDataPoints = 0; // uncomment this line to keep recording over old array values - stop rec. with
GUI button
            fflush(stdout);
        }

        shortCounter++; // use this code to display loop period at program exit
        if(shortCounter == 10000){
            loopPeriod    = (short) (((double) (uclock() - startTcounter))/11931.8); //(10000*UCLOCKS_PER_MICROSECOND)); //
for some reason it doesn't work with the #define'd values
            shortCounter  = 0;
            startTcounter = uclock();
        }

        // communicate to debug port GUI via PCI (receive commands and send display vairables)
        switch_drequestDone = GUIcommunication(bufaddr, &sysProperties, &sensorData, loopPeriod, &bodyData);

        UpdateRegControl(bufaddr, sysProperties.calibration.GUIflag, switch_drequestDone);

        stopProgram = (sysProperties.mainOperationMode == 99);

    } //while !kbhit() loop

    // local data recording by exo cpu, added by JRS, 2004-06-10
    printf("\n\nloopPeriod: %d, uclocks per second: %d",loopPeriod, UCLOCKS_PER_SEC);
    printf("\n\ntotalRecordedDataPoints = %d\n",totalRecordedDataPoints);
    printf("\n\nBEGINING WRITING DATA TO DISK (BE PATIENT, THIS TAKES 2-10 MINUTES)\n\n");
    localCount = 0;
    InitRecordDataToMem(FileArray);
    for(numPointsWrittenToFlash=0; numPointsWrittenToFlash<totalRecordedDataPoints; numPointsWrittenToFlash++){
        WriteDataToFlash(&recordedDataArray[numPointsWrittenToFlash],FileArray);
        if (localCount == 100){
            printf("%d \n",numPointsWrittenToFlash);
            fflush( stdout );
            localCount  = 1;
        }
        else
            localCount++;
    }
    printf("\n\nALL DATA LOGGED TO DISK\n\n");
    CloseFiles(FileArray);
    StopPCIcommunication(bufaddr);

    free(recordedDataArray); //free memory allocated for dataArray at end of program

    return 1;
}


/* Function: Supervisor
* ------------------------------------------------------------------------------------------
* Main system controller. Collects sensor data and implements the control of all machine joints according
* to the main operation mode (0:stop; 1:valve control; 2: manual torque control; 3: auto torque control;
*    4: position control, 5: exit).
*/
void Supervisor(long           bufaddr,
                BodyDataT      *bodyData,
                SysPropertiesT *sysProperties,
                SensorDataT    *sensorData){

    int    supervisorState;
    int    i;
    double desiredTorques[6]     = {0,0,0,0,0,0}; // desired torque on each actuated joint [Lankle Lknee Lhip Rankle Rknee
Rhip]
```

```
    double desiredPosition[6]    = {0,0,0,0,0,0}; // desired torque on each actuated joint [Lankle Lknee Lhip Rankle Rknee
Rhip]
    double desiredTorques_dot[6] = {0,0,0,0,0,0}; // derivatives of desired torque on each actuated joint
    double amplitude_deg         = 0;
    double kneeDesiredPosition   = 0;

    //set upper body mass = dither amplitude
    bodyData->upperBody.mass = sysProperties->ditherAmplitude; // 2003-10-16 by JRS

    if (GetSensorData(bufaddr, sensorData) == 0){ // Read sensor data and update sensorData structure
        //sysProperties->mainOperationMode = 0; // if FPGA data is zero twice in a row, try to warn user and go on stop
mode
        sensorData->error = 1100;
    }

    // if(sensorData->error != 0 ){ // if a signal is out of range, set supervisor state to stop
    if(0){ // if a signal is out of range, set supervisor state to stop
        supervisorState = 0;
    }else{
        supervisorState = sysProperties->mainOperationMode; // Find current state from sensor data, GUI
    }

    // get the VFC torques as long as the exo isn't in STOP mode
    if (supervisorState == VALVE_CTL ||
        supervisorState == MANUAL_TORQUE_CTL ||
        supervisorState == AUTO_TORQUE_CTL ||
        supervisorState == POSITION_CTL){

        for (i=0; i<6; i++){
            sysProperties->jointControl[i].prevJointControlType = sysProperties->jointControl[i].jointControlType; //
record previous
        }

        SysTorqueController(desiredTorques, bodyData, sensorData, sysProperties); // Update desired torques

        // loop through each joint and apply the appropriate torque controller.
        for (i=0; i<6; i++){

            if (sysProperties->jointControl[i].jointControlType == AUTO_TORQUE_CTL){
                JointController(desiredTorques[i], 0, 0, sensorData, sysProperties, i); //Generate torques at joints

            }else if (sysProperties->jointControl[i].jointControlType == MANUAL_TORQUE_CTL){
                desiredTorques[i] = SinusoidalSignal(sysProperties->jointControl[i].manualTorque, sysProperties-
>jointControl[i], i);
                JointController(desiredTorques[i], 0, 0, sensorData, sysProperties, i); //Generate torques at joints

            }else if (sysProperties->jointControl[i].jointControlType == POSITION_CTL){
                amplitude_deg = sysProperties->jointControl[i].amplitude; // get amplitude from GUI command
                sysProperties->jointControl[i].amplitude = amplitude_deg * Pi/180; // convert to rads
                desiredPosition[i] = SinusoidalSignal(sysProperties->jointControl[i].desiredPosition, sysProperties-
>jointControl[i], i);
                PositionControl(desiredPosition[i], sensorData, sysProperties, i);
                //sysProperties->jointControl[i].amplitude = amplitude_deg;

            }else if (sysProperties->jointControl[i].jointControlType == KNEEE_LOCK_POSITION_CTL){
                PositionControl(sensorData->lockingKneeDesiredPosition, sensorData, sysProperties, sensorData-
>dynamicMode.lockingKnee);

            }else if (sysProperties->jointControl[i].jointControlType == VALVE_CTL){
                if (i == LANKLE_T)
                    sensorData->jointData[LANKLE].valveVoltage = SinusoidalSignal(sysProperties-
>jointControl[LANKLE_T].manualValveInput, sysProperties->jointControl[LANKLE_T], LANKLE_T);
                else if (i == LKNEE_T)
                    sensorData->jointData[LKNEE].valveVoltage  = SinusoidalSignal(sysProperties-
>jointControl[LKNEE_T].manualValveInput,  sysProperties->jointControl[LKNEE_T],  LKNEE_T);
                else if (i == LHIP_T)
                    sensorData->jointData[LHIP].valveVoltage   = SinusoidalSignal(sysProperties-
>jointControl[LHIP_T].manualValveInput,   sysProperties->jointControl[LHIP_T],   LHIP_T);
                else if (i == RANKLE_T)
                    sensorData->jointData[RANKLE].valveVoltage = SinusoidalSignal(sysProperties-
>jointControl[RANKLE_T].manualValveInput, sysProperties->jointControl[RANKLE_T], RANKLE_T);
                else if (i == RKNEE_T)
                    sensorData->jointData[RKNEE].valveVoltage  = SinusoidalSignal(sysProperties-
>jointControl[RKNEE_T].manualValveInput,  sysProperties->jointControl[RKNEE_T],  RKNEE_T);
                else if (i == RHIP_T)
                    sensorData->jointData[RHIP].valveVoltage   = SinusoidalSignal(sysProperties-
>jointControl[RHIP_T].manualValveInput,   sysProperties->jointControl[RHIP_T],   RHIP_T);
            }
        }
    }else {
        for (i=0; i<6; i++){
            sensorData->jointData[i].valveVoltage = DEFAULT_VALVE_IN; // Set the valve input to the the default values
(i.e. closed valves)
        }
        for (i=0; i<6; i++){
            sysProperties->jointControl[i].Ci = 0; // zero all integral gains to avoid error accumulation
        }
    }

    if (sysProperties->calibration.GUIflag == 2
        || sysProperties->calibration.GUIflag == 3
        || sysProperties->calibration.GUIflag == 4){ // if user has issued the appropriate calibration request

        //CalibrateAccel(sysProperties, sensorData); // calibrate accelerometers (not tested yet)
    }

    if (sysProperties->calibration.GUIflag == 5
        || sysProperties->calibration.GUIflag == 6
        || sysProperties->calibration.GUIflag == 7
```

```
        || sysProperties->calibration.GUIflag == 8){ // if user has issued the appropriate calibration request

        //CalibrateLoad(bodyData, sysProperties, sensorData); // calibrate load (not tested yet)
    }

    if (sysProperties->calibration.GUIflag == 9){
        //CalibrateFTsensor(sensorData); // F/T backpack sensor bias adjustment
        sysProperties->calibration.GUIflag = 0; // reset calibration flag
    }

    sensorData->jointData[LANKLE].Tdes = desiredTorques[LANKLE_T]; // update structure for user display
    sensorData->jointData[LKNEE].Tdes  = desiredTorques[LKNEE_T];
    sensorData->jointData[LHIP].Tdes   = desiredTorques[LHIP_T];
    sensorData->jointData[RANKLE].Tdes = desiredTorques[RANKLE_T];
    sensorData->jointData[RKNEE].Tdes  = desiredTorques[RKNEE_T];
    sensorData->jointData[RHIP].Tdes   = desiredTorques[RHIP_T];

    // display upper body mass as SG[0] to verify that it is changing
    //sensorData->torsoForce.SG[0] = bodyData->upperBody.mass; // 10/16/03 by JRS

    UpdateDACs(bufaddr, sysProperties, sensorData); // Set the valve input to the user provided voltage
}

/* Function: PositionControl
* ------------------------------------------------------------------------------------------
* Apply joint torque control for position control. Position for each joint is given in sysProperties
* Required torque is obtained with proportional control and stored in 'desiredtorques'
*/
void PositionControl(double                desiredPosition,
                     SensorDataT          *sensorData,
                     const SysPropertiesT *sysProperties,
                     int                   i){

    double kp, q_des, q, dq, qe, dqe, u;
    int sign = 1; // changed from  sign = -1 on 2003-07-29
    int valveNumberInJointData;


    switch(i){ // change array index number so it matches convention in jointData array
    case LANKLE_T:
        valveNumberInJointData = LANKLE;
        break;
    case LKNEE_T:
        valveNumberInJointData = LKNEE;
        break;
    case LHIP_T:
        valveNumberInJointData = LHIP;
        break;
    case RANKLE_T:
        valveNumberInJointData = RANKLE;
        break;
    case RKNEE_T:
        valveNumberInJointData = RKNEE;
        break;
    case RHIP_T:
        valveNumberInJointData = RHIP;
        break;
    }

    if(i == RKNEE_T || i == LKNEE_T){ // adjust sign for knee (obsolete)
        sign = 1;
    }

    kp = sysProperties->jointControl[i].kp;    // get latest feedback gains

    q_des = desiredPosition; // get desired joint position (rad)
    q     = sensorData->jointData[valveNumberInJointData].position;    // get current joint position (rad)
    dq    = sensorData->jointData[valveNumberInJointData].velocity;    // get current joint velocity (rad/s)
    qe    = q_des - q; // error
    dqe   = -dq; // error time derivative (use in nonlinear controller)

    u= sign * kp * qe; // required voltage

    sensorData->jointData[valveNumberInJointData].valveVoltage = u; // valve input voltage (V)
}


/* Function: SysTorqueController
* ------------------------------------------------------------------------------------------
* Apply joint torque control for human-machine force control, Virtual guard and virtual limit stop modes.
* desiredTorques is a pointer to the 6 element array [Lankle Lknee Lhip Rankle Rknee Rhip] containing
* the resulting desired control torques.
*/
void SysTorqueController(double          *desiredTorques,
                         const BodyDataT *bodyData,
                         SensorDataT     *sensorData,
                         SysPropertiesT  *sysProperties){

    double FHMTorques[6]     = {0,0,0,0,0,0}; // torques to computed by the human-machine auto-control law
    double VGuardTorques[6]  = {0,0,0,0,0,0};
    double VLimitsTorques[6] = {0,0,0,0,0,0};

    GetDynMode(sensorData, sysProperties); // find and update the machine's dynamic mode (Jump, Single Support, etc...)

    if (sysProperties->mainOperationMode == AUTO_TORQUE_CTL){ // if the exo is in Virtual Force Control (auto torque) mode
        if (sysProperties->debuggingControls.test3){ // and if test3 is ON, (i.e turn on "lock-the-knee-at-heel-strike")
            if ((sensorData->dynamicMode.prevDynMode == SSTANCE) // and if the previous mode was single stance
                && ((sensorData->dynamicMode.Mode == DSTANCE) // and the current mode is double stance
                || (sensorData->dynamicMode.Mode == ONE_REDUNDANCY))){ // or the current mode is one redundancy
```

```
            if (sensorData->dynamicMode.groundedLeg == LEFT){ // and if the left leg is striking the ground
                sysProperties->jointControl[LKNEE_T].jointControlType = KNEEE_LOCK_POSITION_CTL; //lock the left
knee in its current position
                sensorData->dynamicMode.lockingKnee = LKNEE_T;
                sensorData->lockingKneeDesiredPosition = sensorData->jointData[LKNEE].position;
            }else if (sensorData->dynamicMode.groundedLeg == RIGHT){ // else if the right is striking the ground
                sysProperties->jointControl[RKNEE_T].jointControlType = KNEEE_LOCK_POSITION_CTL; // lock the right
knee in its current position
                sensorData->dynamicMode.lockingKnee = RKNEE_T;
                sensorData->lockingKneeDesiredPosition = sensorData->jointData[RKNEE].position;
            }
        }

        // unlock the knee when the other leg (in swing) strikes the ground
        if ((sysProperties->jointControl[LKNEE_T].prevJointControlType == KNEEE_LOCK_POSITION_CTL) // if the left knee
was previously locked
            && (sysProperties->jointControl[RKNEE_T].prevJointControlType == AUTO_TORQUE_CTL) // and the right leg was
previously free to swing
            && (sysProperties->jointControl[RKNEE_T].jointControlType == KNEEE_LOCK_POSITION_CTL)){ // and now the
right leg has become locked

            sysProperties->jointControl[LKNEE_T].jointControlType = AUTO_TORQUE_CTL; // then unlock the left knee

        }else if ((sysProperties->jointControl[RKNEE_T].prevJointControlType == KNEEE_LOCK_POSITION_CTL) // if the
right knee was previously locked
            && (sysProperties->jointControl[LKNEE_T].prevJointControlType == AUTO_TORQUE_CTL) // and the left leg was
previously free to swing
            && (sysProperties->jointControl[LKNEE_T].jointControlType == KNEEE_LOCK_POSITION_CTL)){ // and now the left
leg has become locked

            sysProperties->jointControl[RKNEE_T].jointControlType = AUTO_TORQUE_CTL; // then unlock the right knee
        }
    }else if ((sysProperties->jointControl[RKNEE_T].prevJointControlType == KNEEE_LOCK_POSITION_CTL) // else, if the
knee-lock is disabled and either knee was previously in a locked state
        || (sysProperties->jointControl[LKNEE_T].prevJointControlType == KNEEE_LOCK_POSITION_CTL)){ // make sure knee
is back in auto torque mode
        sysProperties->jointControl[RKNEE_T].jointControlType = AUTO_TORQUE_CTL;
        sysProperties->jointControl[LKNEE_T].jointControlType = AUTO_TORQUE_CTL;
    }
}

// enable virtual limits
//if (sysProperties->virtualLimit.activation){ // if the virtual limits have been enabled
//    GetVLimitsTorques(sysProperties, sensorData); // compute the Virtual Limits torques
//}else{ // stored in sensorData->jointData[i].Tvlimit
//    for (i=0; i<6; i++)  sensorData->jointData[i].Tvlimit = 0;
//}

GetFHMcontrolTorques(FHMTorques, bodyData, sensorData, sysProperties); // compute the required joint torques
// in FHMcontrol.c

// Compute total desired torque
desiredTorques[LANKLE_T] = FHMTorques[LANKLE_T] /** VGuardTorques[i] + sensorData->jointData[LANKLE].Tvlimit*/ ;
desiredTorques[LKNEE_T]  = FHMTorques[LKNEE_T]  /** VGuardTorques[i]  + sensorData->jointData[LKNEE].Tvlimit*/;
desiredTorques[LHIP_T]   = FHMTorques[LHIP_T]   /** VGuardTorques[i] + sensorData->jointData[LHIP].Tvlimit*/ ;
desiredTorques[RANKLE_T] = FHMTorques[RANKLE_T] /** VGuardTorques[i] + sensorData->jointData[RANKLE].Tvlimit*/ ;
desiredTorques[RKNEE_T]  = FHMTorques[RKNEE_T]  /** VGuardTorques[i]  + sensorData->jointData[RKNEE].Tvlimit*/;
desiredTorques[RHIP_T]   = FHMTorques[RHIP_T]   /** VGuardTorques[i] + sensorData->jointData[RHIP].Tvlimit*/ ;
}


/* Function: GetVGuardForces
* ------------------------------------------------------------------------------------------
* Computes the required machine operational force to produce the virtual guard potential field. This potential field
* will redirect the machine outside of the virtual guard activation region and will prevent a possible
* loss of stability.
*/
void GetVGuardForces(SensorDataT         *sensorData,
                     const BodyDataT     *bodyData,
                     const VirtualGuardT vguard,
                     const double         xR,
                     const double         xL,
                     const double         c1234,
                     const double         s1234){

    double x; // distance to back foot (m)
    double vGuardForce; // resistive force producd by the virtual guard (N)

    if (vguard.activation){ // If virtual guard control has been enabled

        if (xL > xR){ // find back foot and set distance from CG to back foot
            x = xR;
        }else{
            x = xL;
        }

        if (x > 0 && (-x) < vguard.position){ // Only apply Vguard if
            // 1. CG is in front of back foot; if CG in already behind
            //    the back foot the vguard will be useless and the exo will fall backwards
            // 2. CG is within Vguard actuatiopn zone

            vGuardForce = (vguard.position + x) * vguard.stiffness;

            // saturate the force
            if (vGuardForce > vguard.saturation){
                vGuardForce = vguard.saturation;
            }

            // horizontal force caused by virtual guard
```

```
            sensorData->virtualGuardFx = vGuardForce;

            // hip moment caused by vGuard Force at CG
            sensorData->virtualGuardT = - vGuardForce * (bodyData->upperBody.Lcg*c1234 - bodyData->upperBody.hcg*s1234);
        }
    }
}


/* Function: GetVLimitsTorques
* ------------------------------------------------------------------------------------------
* Computes the required machine torques to produce the virtual joint limit stops. The virtual joint limit
* stop is a potential field which is activated when the machine enters a predefined region near the hardware
* stops. The potential field is produced by a virtual spring-damper system and will prevent the joints from
* reaching their physical limits.
*/
void GetVLimitsTorques(const SysPropertiesT *sysProperties,
                       SensorDataT          *sensorData){

    double angles[8], velocities[8], width, Kj, Bj;
    int i;

    width = sysProperties->virtualLimit.position;  // angle width from joint limit to VLimit activation zone
    Kj    = sysProperties->virtualLimit.stiffness; // potential field stiffness
    Bj    = sysProperties->virtualLimit.damping;   // potential field damping

    for (i=0; i<8; i++){
        angles[i]     = sensorData->jointData[i].position; // joint angles [Ltoe Lankle Lknee Lhip Rtoe Rankle Rknee Rhip]
        velocities[i] = sensorData->jointData[i].velocity; // joint angular velocities
    }

    // check for each joint if it is in VLimit zone and compute appropriate torques.
    if (angles[LANKLE] < ANKLE_MIN + width ){ // if joint is within VLimit activation zone 1
        if( velocities[LANKLE] > 0) Bj = 0; // don't resist motion when moving away from the stops
        sensorData->jointData[LANKLE].Tvlimit = ((ANKLE_MIN + width) - angles[LANKLE] )*Kj - velocities[LANKLE]*Bj;
    }else if ( angles[LANKLE] > ANKLE_MAX - width){ // if joint is within VLimit activation zone 2
        if( velocities[LANKLE] < 0) Bj = 0;
        sensorData->jointData[LANKLE].Tvlimit = ((ANKLE_MAX - width) - angles[LANKLE] )*Kj - velocities[LANKLE]*Bj;
    }else{
        sensorData->jointData[LANKLE].Tvlimit = 0;
    }

    Bj = sysProperties->virtualLimit.damping; // reinitialize to original value
    if (angles[RANKLE] < ANKLE_MIN + width){
        //   if( velocities[RANKLE] > 0) Bj = 0; // don't resist motion when moving away from the stops
        sensorData->jointData[RANKLE].Tvlimit = ((ANKLE_MIN + width) - angles[RANKLE] )*Kj -
velocities[RANKLE]*velocities[RANKLE]*fabs(velocities[RANKLE])*Bj/velocities[RANKLE];
    }else if ( angles[RANKLE] > ANKLE_MAX - width){
        //       if( velocities[RANKLE] < 0) Bj = 0;
        sensorData->jointData[RANKLE].Tvlimit = ((ANKLE_MAX - width) - angles[RANKLE] )*Kj -
velocities[RANKLE]*velocities[RANKLE]*fabs(velocities[RANKLE])*Bj/velocities[RANKLE];
    }else{
        sensorData->jointData[RANKLE].Tvlimit = 0;
    }

    Bj = sysProperties->virtualLimit.damping;
    if (angles[LKNEE] < KNEE_MIN + width){
        if( velocities[LKNEE] > 0) Bj = 0;
        sensorData->jointData[LKNEE].Tvlimit = ((KNEE_MIN + width) - angles[LKNEE] )*Kj - velocities[LKNEE]*Bj;
    }else if ( angles[LKNEE] > KNEE_MAX- width){
        if( velocities[LKNEE] < 0) Bj = 0;
        sensorData->jointData[LKNEE].Tvlimit = ((KNEE_MAX - width) - angles[LKNEE] )*Kj - velocities[LKNEE]*Bj;
    }else{
        sensorData->jointData[LKNEE].Tvlimit = 0;
    }

    Bj = sysProperties->virtualLimit.damping;
    if (angles[RKNEE] < KNEE_MIN + width ){
        // if( velocities[RKNEE] > 0) Bj = 0;
        sensorData->jointData[RKNEE].Tvlimit = ((KNEE_MIN + width) - angles[RKNEE] )*Kj -
velocities[RKNEE]*velocities[RKNEE]*fabs(velocities[RKNEE])*Bj/velocities[RKNEE];
    }else if ( angles[RKNEE] > KNEE_MAX - width){
        // if( velocities[RKNEE] < 0) Bj = 0;
        sensorData->jointData[RKNEE].Tvlimit = ((KNEE_MAX - width) - angles[RKNEE] )*Kj -
velocities[RKNEE]*velocities[RKNEE]*fabs(velocities[RKNEE])*Bj/velocities[RKNEE];
    }else{
        sensorData->jointData[RKNEE].Tvlimit = 0;
    }

    Bj = sysProperties->virtualLimit.damping;
    if (angles[LHIP] < HIP_MIN + width){
        if( velocities[LHIP] > 0) Bj = 0;
        sensorData->jointData[LHIP].Tvlimit = ((HIP_MIN + width) - angles[LHIP] )*Kj - velocities[LHIP]*Bj;
    }else if ( angles[LHIP] > HIP_MAX - width){
        if( velocities[LHIP] < 0) Bj = 0;
        sensorData->jointData[LHIP].Tvlimit = ((HIP_MAX - width) - angles[LHIP] )*Kj - velocities[LHIP]*Bj;
    }else{
        sensorData->jointData[LHIP].Tvlimit = 0;
    }

    Bj = sysProperties->virtualLimit.damping;
    if (angles[RHIP] < HIP_MIN + width ){
        // if( velocities[RHIP] > 0) Bj = 0;
        sensorData->jointData[RHIP].Tvlimit = ((HIP_MIN + width) - angles[RHIP] )*Kj -
velocities[RHIP]*velocities[RHIP]*fabs(velocities[RHIP])*Bj/velocities[RHIP];
    }else if ( angles[RHIP] > HIP_MAX - width){
        // if( velocities[RHIP] < 0) Bj = 0;
        sensorData->jointData[RHIP].Tvlimit = ((HIP_MAX - width) - angles[RHIP] )*Kj -
velocities[RHIP]*velocities[RHIP]*fabs(velocities[RHIP])*Bj/velocities[RHIP];
```

```
    }else{
        sensorData->jointData[RHIP].Tvlimit = 0;
    }
}


/* Function: GetDynMode
* ------------------------------------------------------------------------------------------------
* Establishes the dynamic mode of the machine according to footswitch data.
* Mode accepts the following values:
*       - jump mode          =    0
*       - single support =    1
*       - double support =    2
*       - double support single redundancy = 3
*       - double support double redundancy = 4
*  LeftIsGrounded   is 0 or 1
*  LeftIsRedundant  is 0 or 1
*  LeftHeelContact  is 0 or 1
*  RightHeelContact is 0 or 1
*/
void GetDynMode(SensorDataT          *sensorData,
                const SysPropertiesT *sysProperties){ //note: added sysProperties 2003-08-13

    int RFootStatus, LFootStatus; // no contact:0; heel contact:1; toe contact:2; flat:3;
    int Mode;
    int leftHeelContact, rightHeelContact;
    int groundedLeg;
    int redundantLeg;

    groundedLeg      = NONE;
    redundantLeg     = NONE;

    leftHeelContact  = FALSE;
    rightHeelContact = FALSE;

    sensorData->dynamicMode.LstanceSwingTransition = FALSE; // initialize state transition indicators (used in
GetFHMcontrolTorques)
    sensorData->dynamicMode.RstanceSwingTransition = FALSE;

    sensorData->dynamicMode.LstateTransition = FALSE;
    sensorData->dynamicMode.RstateTransition = FALSE;

    // record previous dyn mode data
    sensorData->dynamicMode.prevDynMode        = sensorData->dynamicMode.Mode;
    sensorData->dynamicMode.prevGroundedLeg    = sensorData->dynamicMode.groundedLeg;

    // Obtain foot status from foot switches:
    //        no contact   = 0
    //        heel contact = 1
    //        toe contact  = 2
    //        flat         = 3
    RFootStatus = GetFootStatus(sensorData->Rfootswitch);
    LFootStatus = GetFootStatus(sensorData->Lfootswitch);

    // single support, right leg on ground
    if (RFootStatus != 0 && LFootStatus == 0){
        Mode = SSTANCE;
        groundedLeg = RIGHT;
        redundantLeg = NONE;

//        if((sensorData->dynamicMode.Mode == SSTANCE && (sensorData->dynamicMode.groundedLeg == LEFT)) // if left leg was
previously on the ground
//            || sensorData->dynamicMode.Mode == DSTANCE
//            || sensorData->dynamicMode.Mode == ONE_REDUNDANCY){
//
//            sensorData->dynamicMode.LstanceSwingTransition = TRUE; // signal transition (used in GetFHMcontrolTorques)
//        }
    }

    // single support, left leg on ground
    else if (LFootStatus != 0 && RFootStatus == 0){
        Mode = SSTANCE;
        groundedLeg = LEFT;
        redundantLeg = NONE;

//        if((sensorData->dynamicMode.Mode == SSTANCE && (sensorData->dynamicMode.groundedLeg == RIGHT))
//            || sensorData->dynamicMode.Mode == DSTANCE
//            || sensorData->dynamicMode.Mode == ONE_REDUNDANCY){ // if right leg was previously on the ground
//
//            sensorData->dynamicMode.RstanceSwingTransition = TRUE; // signal transition (used in GetFHMcontrolTorques)
//        }
    }

    // right foot is flat
    else if (RFootStatus == 3){
        rightHeelContact = TRUE;

        // left foot is flat
        // double support
        if (LFootStatus == 3){
            Mode = DSTANCE;
            redundantLeg = NONE;
            leftHeelContact = TRUE;
        }

        // left foot is in heel contact
        // double support single redundancy
        else if (LFootStatus == 1){
            //Mode = ONE_REDUNDANCY;
```

```
            //redundantLeg = LEFT;
            Mode = DSTANCE; // 2003-06-19
            redundantLeg = NONE; // 2003-06-19
            leftHeelContact = TRUE;
        }

        // left foot is in toe contact
        // double support single redundancy
        else if (LFootStatus == 2){
            Mode = ONE_REDUNDANCY;
            //Mode = SSTANCE; // test for 2004-10-04, knee locking out seems to happen only for sstance->1red->dstance
transition
            groundedLeg = RIGHT;
            redundantLeg = LEFT;
            //leftIsGrounded = 0;  // test for 2004-10-04, this line added to mimic sstance, right leg gnd case
            //leftIsRedundant = 0; // test for 2004-10-04, changed to 0 from 1
            leftHeelContact = FALSE;
        }
    }

    // left foot is flat (right foot must be redundant)
    else if (LFootStatus == 3){
        redundantLeg = RIGHT;
        leftHeelContact = TRUE;

        // right foot is in heel contact
        // double support single redundancy
        if (RFootStatus == 1){
            //Mode = ONE_REDUNDANCY;
            //redundantLeg = RIGHT;
            Mode = DSTANCE; // 06 19 03
            redundantLeg = NONE; // 2003-06-19
            rightHeelContact = TRUE;
        }

        // right foot is in toe contact
        // double support single redundancy
        else if (RFootStatus == 2){
            Mode = ONE_REDUNDANCY;
            //Mode = SSTANCE; // test for 2004-10-04, knee locking out seems to happen only for sstance->1red->dstance
transition
            groundedLeg = LEFT;
            redundantLeg = RIGHT;
            rightHeelContact = FALSE;
        }
    }

    // left foot is in heel contact
    // double support double redundancy (unstable)
    else if (LFootStatus == 1){
        leftHeelContact = TRUE;

        // right foot is in heel contact
        // double support (use this instead of [unstable] 2Red -06 02 03)
        if (RFootStatus == 1){
            //Mode = TWO_REDUNDANCY;
            Mode = DSTANCE; // commented 2004-09-22
            redundantLeg = NONE;
            //Mode = ONE_REDUNDANCY;
            //redundantLeg = RIGHT;
            rightHeelContact = TRUE;
        }

        // right foot is in toe contact
        // double support single redundancy w left red (use this instead of unstable 2Red 2003-06-02)
        else{
            //Mode = TWO_REDUNDANCY;
            Mode = ONE_REDUNDANCY;
            //Mode = SSTANCE; // test for 2004-10-04, knee locking out seems to happen only for sstance->1red->dstance
transition
            groundedLeg = LEFT;
            redundantLeg = RIGHT;
            rightHeelContact = FALSE;
        }
    }

    // right foot is in heel contact (left is toe contact)
    // double support single redundancy w left red (use this instead of unstable 2Red 2003-06-02)
    else if (RFootStatus == 1){
        //Mode = TWO_REDUNDANCY;
        Mode = ONE_REDUNDANCY;
        //Mode = SSTANCE; // test for 2004-10-04, knee locking out seems to happen only for sstance->1red->dstance
transition
        groundedLeg = RIGHT;
        redundantLeg = LEFT;
        //leftIsRedundant  = 1;
        rightHeelContact = TRUE;
        leftHeelContact  = FALSE;
    }

    // left and right feet are in toe contact
    // double support (use this instead of [unstable] 2Red 2003-06-02)
    else if (LFootStatus == 2 && RFootStatus == 2){
        //Mode = TWO_REDUNDANCY; // Double support double redundancy (unstable)
        Mode = DSTANCE;
        redundantLeg = NONE; // arbitrary?
        rightHeelContact = 0;
        leftHeelContact  = 0;
    }
```

```
    // jump mode
    else if (LFootStatus == 0 && RFootStatus == 0){
        Mode = JUMP;
        //Mode = sensorData->dynamicMode.Mode; // set dynamic mode to previous mode
        //groundedLeg = sensorData->dynamicMode.groundedLeg; // set grounded leg to previous grounded leg
        rightHeelContact = 0;
        leftHeelContact  = 0;
        groundedLeg  = NONE;
        redundantLeg = NONE;
    }

//    // record sstance-to-swing transitions if they occur, also note which leg is transitioning
//    if(sensorData->dynamicMode.Mode == SSTANCE && (Mode == DSTANCE || Mode == ONE_REDUNDANCY)){
//        if(sensorData->dynamicMode.groundedLeg == LEFT){ // if left leg was previously stance leg
//            sensorData->dynamicMode.RstanceSwingTransition = TRUE;  // signal transition (used in GetFHMcontrolTorques)
//            sensorData->dynamicMode.LstanceSwingTransition = FALSE; // signal transition (used in GetFHMcontrolTorques)
//        }else if(sensorData->dynamicMode.groundedLeg == RIGHT){ // if right leg was previously stance leg
//            sensorData->dynamicMode.LstanceSwingTransition = TRUE;  // signal transition (used in GetFHMcontrolTorques)
//            sensorData->dynamicMode.RstanceSwingTransition = FALSE; // signal transition (used in GetFHMcontrolTorques)
//        }
//    }
//
//    // record dstance-to-1red transitions if they occur, also note which leg is transitioning
//    if(sensorData->dynamicMode.Mode == DSTANCE && Mode == ONE_REDUNDANCY){
//        if(redundantLeg == LEFT){
//            sensorData->dynamicMode.LstateTransition = TRUE;  // signal transition (used in GetFHMcontrolTorques)
//            sensorData->dynamicMode.RstateTransition = FALSE; // signal transition (used in GetFHMcontrolTorques)
//        }else if(redundantLeg == RIGHT){
//            sensorData->dynamicMode.RstateTransition = TRUE;  // signal transition (used in GetFHMcontrolTorques)
//            sensorData->dynamicMode.LstateTransition = FALSE; // signal transition (used in GetFHMcontrolTorques)
//        }
//
//    // record 1red-to-dstance transitions if they occur, also note which leg is transitioning
//    }else if(sensorData->dynamicMode.Mode == ONE_REDUNDANCY && Mode == DSTANCE){
//        if(sensorData->dynamicMode.redundantLeg == LEFT){
//            sensorData->dynamicMode.LstateTransition = TRUE;  // signal transition (used in GetFHMcontrolTorques)
//            sensorData->dynamicMode.RstateTransition = FALSE; // signal transition (used in GetFHMcontrolTorques)
//        }else if (sensorData->dynamicMode.redundantLeg == RIGHT){
//            sensorData->dynamicMode.RstateTransition = TRUE;  // signal transition (used in GetFHMcontrolTorques)
//            sensorData->dynamicMode.LstateTransition = FALSE; // signal transition (used in GetFHMcontrolTorques)
//        }
//    }

    sensorData->dynamicMode.Mode              = Mode; // update values in original data structure
    sensorData->dynamicMode.groundedLeg       = groundedLeg;
    sensorData->dynamicMode.redundantLeg      = redundantLeg;
    sensorData->dynamicMode.leftHeelContact   = leftHeelContact;
    sensorData->dynamicMode.rightHeelContact  = rightHeelContact;
}


/* Function: GetFootStatus
* ------------------------------------------------------------------------------------------
* Returns the foot status according the footswitch readings.
* The returned value is as follows:
*       no contact   = 0
*       heel contact = 1
*       toe contact  = 2
*       flat         = 3
*/

int GetFootStatus(const int *footswitch){

    int sw[5]; // switch array: [TIP TOE BALL MIDFOOT HEEL]
    int i;

    // copy footswitch data to new array
    for (i=5; i; i--){
        sw[i-1] = footswitch[i-1];
    }

    // use switch states to give foot orientation wrt ground
    if ((sw[HEEL]          && (sw[MIDFOOT] || sw[BALL] || sw[TOE]) ) // if heel and (anything else)
        || ( sw[MIDFOOT]   && (sw[BALL] || sw[TOE]) ) // or (if midfoot and (ball or toe))
        ||   sw[MIDFOOT]){ // or (if just the midfoot)
        return 3; // flat foot

    }else if (sw[HEEL]) { // heel contact
        return 1;

    }else if (sw[BALL] || sw[TOE]) { // toe contact
        return 2;

    }else{
        return 0; // no contact

    }
}


/* Function: SinusoidalSignal
* ------------------------------------------------------------------------------------------
* Makes a sinusoidal signal out of a nominal data, an offset, an amplitude and a frequency.
*/
double SinusoidalSignal(double         nominalSignal,
                        JointControlT jointControl,
                        int           i){
```

```
    double signal, wt;
    double x; // temporary variable
    static double phase = 0; // phase induced from resetting counter

    wt = 2*Pi*jointControl.frequency*loopCter*TS;

    if (i == 0 && counterResetFlag == 1){ // if counter has been reset (do only once, not 6 times -for each joint)
        x = (wt + phase)/(2*Pi); // see counterReset.m file

        if( x<0 ){
            x = -x;
            phase = (wt + phase) + floor(x)*2*Pi; // compute the new phase
        }else{
            phase = (wt + phase) - floor(x)*2*Pi; // compute the new phase
        }

        wt = 0;     // reset time
    }

    signal  = nominalSignal + jointControl.amplitude * sin(wt + phase);

    return signal;
}


/* Function: InitBodyData
* ------------------------------------------------------------------------------------------
* This function intializes the bodyData structure (body length and mass properties). These values cannot be
* modified by the GUI.
*/
void InitBodyData(BodyDataT *bodyData){

    // Define these values as constants in define.h for improved speed

    bodyData->heel.length        = HEEL_LENGTH; // heel mass properties exclude ankle bearing or actuator
    bodyData->heel.Lcg            = HEEL_LCG; // dist from heel to foot CG
    bodyData->heel.hcg            = HEEL_HCG; // dist from heel to foot CG
    bodyData->heel.mass           = HEEL_MASS;
    bodyData->heel.inertia        = HEEL_INERTIA;

    bodyData->foot.mass           = FOOT_MASS; // foot mass properties exclude ankle bearing or actuator
    bodyData->foot.inertia        = FOOT_INERTIA;
    bodyData->foot.length        = FOOT_LENGTH;
    bodyData->foot.Lcg            = FOOT_LCG; // dist from ankle to foot CG
    bodyData->foot.hcg            = FOOT_HCG; // dist from ankle to foot CG

    bodyData->shank.mass         = SHANK_MASS; // shank mass properties includes ankle bearings and actuator
    bodyData->shank.inertia       = SHANK_INERTIA; // but excludes knee bearings and actuator
    bodyData->shank.length        = SHANK_LENGTH;
    bodyData->shank.Lcg           = SHANK_LCG;
    bodyData->shank.hcg           = SHANK_HCG    ;

    bodyData->thigh.mass         = THIGH_MASS; // thigh mass properties includes knee bearings, knee and hip actuators,
    bodyData->thigh.inertia       = THIGH_INERTIA; // thigh actuator, but excludes hip bearings.
    bodyData->thigh.length        = THIGH_LENGTH;
    bodyData->thigh.Lcg           = THIGH_LCG;
    bodyData->thigh.hcg           = THIGH_HCG;

    bodyData->upperBody.mass     = UPPERBODY_MASS; // upper body properties excludes hip bearings and actuators
    bodyData->upperBody.inertia = UPPERBODY_INERTIA;
    bodyData->upperBody.length  = 1; // irrelevant
    bodyData->upperBody.Lcg      = UPPERBODY_LCG;
    bodyData->upperBody.hcg      = UPPERBODY_HCG;

    bodyData->torsoSensor_L      = TORSOSENSOR_L; // vertical distance of ATI F/T sensor from hip axis
    bodyData->torsoSensor_h      = TORSOSENSOR_H; // horizontal distance ...
}


/* Function: InitSysProps
* ------------------------------------------------------------------------------------------
* This function initializes the sysProperties structure. These values can be modified in the GUI.
*/
void InitSysProps(SysPropertiesT *sysProperties){

    int i;

    sysProperties->mainOperationMode        = DEFAULT_CTRL_MODE; // 0:stop; 1:valve control; 2: manual torque control; 3:
auto torque control; 4: velocity control;
    sysProperties->torqueControl.controlFHM = DEFAULT_THM_STATUS; // default set in defines.h, toggle the human-machine
force control (i.e. turn on/off accelerometers)
    sysProperties->DynDistributionFactor    = DEFAULT_DYNDISTFACTOR; // default=0.80333, load distribution factor for
double support (use 0.80333 )

    for (i=0; i<6; i++) {
        sysProperties->jointControl[i].OperationMode    = RUNNING; // 0:idle; 1: on
        sysProperties->jointControl[i].jointControlType = 0; // 0:STOP 1:manual voltage, 2:manual torque, 3:auto torque
(MSS), 4:position control, 5:ERROR
        sysProperties->jointControl[i].prevJointControlType = 0; // 0:STOP 1:manual voltage, 2:manual torque, 3:auto torque
(MSS), 4:position control, 5:ERROR
        sysProperties->jointControl[i].kp               = 0; // joint controller gains
        sysProperties->jointControl[i].kv               = 0;
        sysProperties->jointControl[i].lambda1           = 0; // hydraulic joint controller gains
        sysProperties->jointControl[i].lambda2           = 0;
        sysProperties->jointControl[i].Ci               = 0;
        sysProperties->jointControl[i].CiSwitch          = 1; // multiplier for Ci, used to turn Ci on/off, 1=ON, 0=OFF
        sysProperties->jointControl[i].eta1              = 1;
        sysProperties->jointControl[i].eta2              = 0.001;
        sysProperties->jointControl[i].phi1_inv          = 0;
```

```
        sysProperties->jointControl[i].phi2_inv            = 0.001;
        sysProperties->jointControl[i].ro1                 = 0.001;
        sysProperties->jointControl[i].ro2_inv             = 0.001;
        sysProperties->jointControl[i].manualTorque     = DEFAULT_MANUAL_TORQUE; // joint torques when the controller is on
Manual Torque Control, 0.0Nm
        sysProperties->jointControl[i].manualValveInput = DEFAULT_VALVE_IN; // valve input voltage when the controller is
on Valve Input Control, 0.0V
        sysProperties->jointControl[i].desiredVelocity  = 0; // (rad/s) used in velocity control for friction calibration
        sysProperties->jointControl[i].desiredPosition  = 0; // (rad) used in position control for friction calibration
        sysProperties->jointControl[i].frequency        = 0; // frequency for manual torque, valve input and desired
velocity sinusoidal signals
        sysProperties->jointControl[i].amplitude        = 0; // amplitude for manual torque, valve input and desired
velocity sinusoidal signals
        sysProperties->jointControl[i].kpFHM            = 0; // human-machine force amplification gain
    }

    sysProperties->jointControl[LANKLE_T].lambda2 = 400; // LEFT ANKLE joint controller gains
    sysProperties->jointControl[LKNEE_T].lambda2  = 400; // LEFT KNEE joint controller gains
    sysProperties->jointControl[LHIP_T].lambda2   = 1000; // LEFT HIP joint controller gains

    sysProperties->jointControl[RANKLE_T].lambda2 = 400; // RIGHT ANKLE joint controller gains
    sysProperties->jointControl[RKNEE_T].lambda2  = 400; // RIGHT KNEE joint controller gains
    sysProperties->jointControl[RHIP_T].lambda2   = 1000; // RIGHT HIP joint controller gains

    sysProperties->virtualGuard.activation = 0; // toggle the virtual guard
    sysProperties->virtualGuard.position   = 0; // horizontal position of the VGuard's activation zone
    sysProperties->virtualGuard.saturation = 0; // max force of virtual guard
    sysProperties->virtualGuard.stiffness  = 0;
    sysProperties->virtualGuard.damping    = 0;
    sysProperties->virtualLimit.activation = 0; // toggle the virtual limits
    sysProperties->virtualLimit.position   = 0.26; // angular position of the joint's Vlimit activation zone (rad)
    sysProperties->virtualLimit.saturation = 0; // max force of virtual limit
    sysProperties->virtualLimit.stiffness  = 0;
    sysProperties->virtualLimit.damping    = 20;

    for(i=0; i<7; i++){
        sysProperties->calibration.selection[i] = 0;
        // binary values to select  accelerometers to be calibrated
        // [ LFOOT LSHANK LTHIGH RFOOT RSHANK RTHIGH UPPERBODY]
    }

    sysProperties->GUIping             = 0; // ping signal from GUI   -- NOT USED
    sysProperties->calibration.GUIflag = FALSE; // indicate current state of accelerometer calibration
    sysProperties->ditherAmplitude       = 0; // amplitude of valve dither voltage (V)
    sysProperties->ditherFrequency     = 0;    // valve dither frequency (Hz)
    sysProperties->recordFlag          = FALSE; // signal form GUI to start recording data locally on exo cpu, added by
JRS, 06-18-2004
    sysProperties->resetTimer           = FALSE;

    sysProperties->debuggingControls.activateAnkleSpring   = FALSE;
    sysProperties->debuggingControls.springRate              = 0;
    sysProperties->debuggingControls.centerAngle           = 0;
    sysProperties->debuggingControls.activateKneeDamper    = FALSE;
    sysProperties->debuggingControls.extensionDampingCoeff = 0;
    sysProperties->debuggingControls.flexionDampingCoeff   = 0;
    sysProperties->debuggingControls.test3                   = FALSE;
    sysProperties->debuggingControls.test4                   = FALSE;
}


/* Function: InitSensorData
* --------------------------------------------------------------------------------------------
* This function initializes the sensorData structure. Most of these values are displayed in the GUI.
*/
void InitSensorData(SensorDataT *sensorData){

    int i;

    sensorData->dynamicMode.Mode                    = JUMP;  // 0:jump; 1:sgl sup; 2:dbl sup; 3: dbl sup sgl red; 4:dbl sup
dbl red
    sensorData->dynamicMode.leftHeelContact           = FALSE;    // 1 if the left heel is in contact with the ground
    sensorData->dynamicMode.rightHeelContact        = FALSE;    // 1 if the right heel is in contact with the ground
    sensorData->dynamicMode.LstanceSwingTransition = FALSE;    // 1 if transitioning left leg from stance to swing
    sensorData->dynamicMode.RstanceSwingTransition = FALSE;    // 1 if transitioning right leg from stance to swing
    sensorData->dynamicMode.LstateTransition        = FALSE;    // 1 if left leg transitioning b/w states
    sensorData->dynamicMode.RstateTransition        = FALSE; // 1 if right leg transitioning b/w states
    sensorData->dynamicMode.groundedLeg               = NONE;
    sensorData->dynamicMode.redundantLeg            = NONE;

    sensorData->dynamicMode.prevGroundedLeg           = NONE;
    sensorData->dynamicMode.prevRedundantLeg        = NONE;
    sensorData->dynamicMode.prevDynMode             = 0;
    sensorData->dynamicMode.prevLeftLegStance       = 0; // 0=jump; 1:sgl sup; 2:dbl sup; 3: dbl sup sgl red; 4:dbl sup dbl
red
    sensorData->dynamicMode.prevRightLegStance      = 0; // 0=jump; 1:sgl sup; 2:dbl sup; 3: dbl sup sgl red; 4:dbl sup dbl
red
    sensorData->dynamicMode.prevLeftKneeControlType  = 0; // 0:STOP; 1:manual voltage; 2:manual torque; 3:auto torque
(MSS); 4:position control; 5:ERROR
    sensorData->dynamicMode.prevRightKneeControlType = 0; // 0:STOP; 1:manual voltage; 2:manual torque; 3:auto torque
(MSS); 4:position control; 5:ERROR

    for (i=0; i<8; i++) {
        // Define these values as constants in define.h for improved speed
        sensorData->jointData[i].sensorForce     = 0; // Cylinder force sensor reading (N)
        sensorData->jointData[i].position        = 0; // arrays are as follows: [Ltoe Lankle Lknee Lhip Rtoe Rankle Rknee
Rhip ]
        sensorData->jointData[i].velocity        = 0; // m/s
        sensorData->jointData[i].acceleration    = 0; // m/s/s
        sensorData->jointData[i].momentArm       = 1; // m
```

```
        sensorData->jointData[i].pistonPosition = 0; // piston distance from xp0 reference position (m)
        sensorData->jointData[i].pistonVelocity = 0; // m/s
        sensorData->jointData[i].torque          = 0; // torque caused by actuator of distal on proximal segment(N.m)
        sensorData->jointData[i].Thm            = 0; // Joint Torque of human on machine (N.m)
        sensorData->jointData[i].Tg              = 0; // Joint Torque needed to compensate gravity (N.m)
        sensorData->jointData[i].Tdes            = 0; // Desired Joint Torque vector
        sensorData->jointData[i].Tcc            = 0; // Joint torque needed to compensate centrifugal and coriolis forces
(N.m)
        sensorData->jointData[i].Tf              = 0; // Joint friction torque - includes hosing and cable stiffness
(N.m)
        sensorData->jointData[i].Tlin            = 0; // Linearizing torque Tlin = Tg + Tcc + Tf (N.m)
        sensorData->jointData[i].Tinertial       = 0; // Torque due to inertial forces, JRS, 2004-06-24
        sensorData->jointData[i].Tvguard        = 0; // Virtual guard torque
        sensorData->jointData[i].Tvlimit        = 0; // Virtual limit torque
        sensorData->jointData[i].valveVoltage   = DEFAULT_VALVE_IN; // input voltage on the valve, 0.0V
        sensorData->jointData[i].indexPulse     = 0; // encoder index pulse
        sensorData->jointData[i].againstStop    = FALSE; // FALSE=0, TRUE=1;
    }

    for (i=0; i<5; i++) {
        sensorData->Lfootswitch[i] = 0; // footswitch binary value
        sensorData->Rfootswitch[i] = 0;
    }

    // Define these values as constants in define.h for improved speed
    for (i=0; i<7; i++) {   // bodyAccel[ LFOOT LSHANK LTHIGH RFOOT RSHANK RTHIGH UPPERBODY]
        sensorData->bodyAccel[i].angular_accel = 0;
        sensorData->bodyAccel[i].lin_accel1    = 0; // linear accelerometer outputs
        sensorData->bodyAccel[i].lin_accel2    = 0;
        sensorData->bodyAccel[i].offset1       = 0; // linear accelerometer offsets
        sensorData->bodyAccel[i].offset2       = 0;
        sensorData->bodyAccel[i].gain1         = 1; // linear accelerometer gains
        sensorData->bodyAccel[i].gain2         = 1;
    }

    sensorData->TorsoTilt     = 0; // torso inertial sensor output.
    sensorData->torsoVelocity = 0;

    for (i=0; i<6; i++){
        sensorData->torsoForce.SG[i]  = 0; // torso FT sensor strain gauge outputs
        sensorData->torsoForce.SGt[i] = 0; // temperature-compensated strain gauge outputs
    }

    sensorData->torsoForce.thermister = 0;
    sensorData->torsoForce.Fx         = 0; // torso FT sensor strain gauge outputs
    sensorData->torsoForce.Fy         = 0;
    sensorData->torsoForce.T          = 0;

    sensorData->torsoForce.SGt_bias[0] = BT1; // temperature compensated straingauge bias (V)
    sensorData->torsoForce.SGt_bias[1] = BT2;
    sensorData->torsoForce.SGt_bias[2] = BT3;
    sensorData->torsoForce.SGt_bias[3] = BT4;
    sensorData->torsoForce.SGt_bias[4] = BT5;
    sensorData->torsoForce.SGt_bias[5] = BT6;

    sensorData->hipData.R_abduction           = 0; // unactuated hip joint angles
    sensorData->hipData.R_abduction_indexP = 0; // index pulse
    sensorData->hipData.R_rotation         = 0;
    sensorData->hipData.R_rotation_indexP  = 0;
    sensorData->hipData.L_abduction        = 0;
    sensorData->hipData.L_abduction_indexP = 0;
    sensorData->hipData.L_rotation         = 0;
    sensorData->hipData.L_rotation_indexP  = 0;

    sensorData->forceDistribution.LankleDistance    = 0; // transvers plane distance from CG to ankles
    sensorData->forceDistribution.RankleDistance    = 0;
    sensorData->forceDistribution.weightDistrFactor = 0.5; // force factor due to gravity (alpha)

    for (i=0; i<4; i++) {
        sensorData->forceDistribution.filteredBetaFg[i]    = 0.5; // last 4 elements of the filtered load distribution
factor Beta (A[0] = most recent)
        sensorData->forceDistribution.unfilteredBetaFg[i]  = 0.5; // ... unfiltered ...
        sensorData->forceDistribution.filteredBetaFHM[i]   = 0.5;
        sensorData->forceDistribution.unfilteredBetaFHM[i] = 0.5;
        sensorData->forceDistribution.filteredKrot[i]      = 1; // filtered horiz. force factor due to hip rotation
        sensorData->forceDistribution.unfilteredKrot[i]    = 1;
    }

    sensorData->error              = 0; // set to 'no error' as default
    sensorData->lostCommunication = 0; // number of times communication with PCI was lost
    sensorData->loopPeriod        = 0; // Supervisor loop period (usec)
    sensorData->GUIping             = 0; // GUI ping response (NOT USED)
    sensorData->calibrationFlag   = 0; // indicates current controller state of accelerometer calibration
    sensorData->virtualGuardFx    = 0; // horizontal force caused by virtual guard at upper body CG
    sensorData->virtualGuardT     = 0; // hip moment cause by virtualGuardFx
    sensorData->CounterTicks      = 0; // JRS, 2004-06-28
}
```

Appendix A.4 – Sensors.h

```
/* Function: GetSensorData
 * ---------------------
 * Reads data from the sensors and the FPGA and updates the variables of sensorData.
 * The data from the sensors and the FPGA is read from input pins and consists of: joint angles, joint
 * velocities, joint accelerations, actuator pressures, footswitch data, actuator force, torso tilt.
 */
int GetSensorData(long bufaddr, SensorDataT *sensorData);

/* Function: GetTorsoForce
 * ------------------------
 * Reads the torso interaction forces from the backpack sensors and updates the sensorData structure.
 */
void GetTorsoForce(short *dataArray, SensorDataT *sensorData);

/* Function: GetTorsoTilt
 * ------------------------
 * Reads the torso tilt angle from the Intersense sensor and updates the sensorData structure.
 */

void GetTorsoTilt(short *dataArray, SensorDataT *sensorData);

/* Function: GetJointAngles
 * ------------------------
 * Reads angle data from the FPGA and updates the sensorData structure.
 * Angles are for the proximal segment relative to the  distal segment.
 *
 *
 *                          .
 *             \      +ve .
 *              \  angle .
 * proximal      \<-----.
 * segment        \    .
 * (e.g. thigh)    \  .
 *                  \.
 *                   |
 *                   |  distal segment (e.g. shank)
 *                   |
 *                   |
 *
 */
void GetJointAngles(short *dataArray, SensorDataT *sensorData);

/* Function: GetEncoderIndexPulses
 * ---------------------------
 * Reads encoder index pulse data and updates the sensorData structure.
 */
void GetEncoderIndexPulses(short *dataArray, SensorDataT *sensorData);

/* Function: GetJointVelocities
 * ----------------------------
 * Reads angular velocity data from the FPGA and updates the sensorData structure.
 * Angular velocities are for the  proximal segment relative to the distal segment.
 */
void GetJointVelocities(short *dataArray, SensorDataT *sensorData);

/* Function: GetSensorForces
 * -----------------------------
 * Read cylinder force sensors.
 */
void GetSensorForces(short *dataArray, SensorDataT *sensorData);

/* Function: ActuatorKinematics
 * ----------------------------
 * compute kinematic actuator variables needed in the nonlinear control laws:
 *
 *        L : cylinder length (m)
 *        h : cylinder moment arm (m)
 *        xp : cylinder position (m)
 *        xp_dot : cylinder velocity (m/s)
 */
void ActuatorKinematics(SensorDataT *sensorData );

/* Function: GetTorques
 * ----------------------------
 * Computes joint torques from pressure and angle data.
 */
void GetTorques(SensorDataT *sensorData);

/* Function: GetFootSwitches
 * ----------------------------
 * Reads footswitch data and updates the sensorData structure.
 */
void GetFootSwitches(short *dataArray, SensorDataT *sensorData);

/* Function: GetEncoderIndexPulses
 * ----------------------------
 * Reads encoder index pulse data and updates the sensorData structure.
 */
void GetEncoderIndexPulses(short *dataArray, SensorDataT *sensorData);

/* Function: DetectErrors
 * ----------------------------
 * Set sensorData.error to 1 if a signal is out-of-range
 */
void DetectErrors(    SensorDataT *sensorData);
```

Appendix A.5 – Sensors.c

```
#include <math.h>
#include <stdio.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"
#include "Sensors.h"
#include "Accel.h"
#include "DSup.h"
#include "PCI.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];

extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];


/* Function: GetSensorData
* ------------------------------------------------------------------------------------------
* Reads data from the sensors and the FPGA and updates the variables of sensorData.
* The data from the sensors and the FPGA is read from input pins and consists of: joint angles, joint
* velocities, joint accelerations, actuator pressures, footswitch data, actuator force, torso tilt.
*/
int GetSensorData(long            bufaddr,
                  SensorDataT    *sensorData){

    int          status = 3;
    short       dataArray[84]; // stores read data from the PCI bus
    static int missedFPGA1onLastRun = 0;
    static int missedFPGA2onLastRun = 0;

    status = UpdateDataArray(bufaddr, dataArray);

    if (status != -1){ // if either FPGA is communicating
        GetTorsoTilt(dataArray, sensorData);              // torso tilt: from FAS-G orientation sensor
        GetJointAngles(dataArray, sensorData);             // encoders
        GetJointVelocities(dataArray, sensorData);         // differentiate encoders
        GetBodyAccelerations(dataArray, sensorData);      // ...from accelerometers
        GetJointAccelerations(sensorData);                 // use accelerometer pairs
        //GetJointAccelerationsVelocityBased(sensorData); // use double differentiated encoders for angular acceleration
        GetSensorForces(dataArray, sensorData);            // actuator force sensors
        GetTorsoForce(dataArray, sensorData);             // backpack six axis force sensor
        ActuatorKinematics(sensorData);                    // piston position, velocity, actuator moment arm
        GetTorques(sensorData);                            // joint torques
        GetFootSwitches(dataArray, sensorData);            // foot switches
        GetEncoderIndexPulses(dataArray, sensorData);     // index pulses
        DetectErrors(sensorData);                         // find out-of-range sensor errors
    } else{    // send an error message to the GUI that FPGA sensor data was not ready
        // ........
    }
    return 1;
}


/* Function: GetTorsoTilt
 * ------------------------------------------------------------------------------------------
 * Reads the torso tilt angle from the Intersense sensor and updates the sensorData structure.
 */
void GetTorsoTilt(short           *dataArray,
                  SensorDataT    *sensorData){

    static double unfilteredAngle[4] = {0,0,0,0}; // Current A[0] and Previous values
    static double filteredAngle[4]   = {0,0,0,0}; // used in lowpass filter

    sensorData->TorsoTilt = dataArray[27]*A_TO_D_CONVERSION*TORSO_INCL_SLOPE + TORSO_INCL_OFFSET;

    // filter angle
    LowpassFilter(unfilteredAngle, filteredAngle, &sensorData->TorsoTilt, filterCoeffs2nd10); // 10 Hz filter (5Hz)
}


/* Function: GetJointAngles
* ------------------------------------------------------------------------------------------
* Reads angle data from the FPGA and updates the sensorData structure.
* Angles are for the proximal segment relative to the  distal segment.
*
*
*
*                       \    +ve   .
```

```
*                          \  angle .
* proximal                 \<-----.
* segment                   \      .
* (e.g. thigh)               \  .
*                             \.
*                              |
*                              |  distal segment (e.g. shank)
*                              |
*                              |
*                              |
*/
void GetJointAngles(short *dataArray, SensorDataT *sensorData){

    sensorData->jointData[LANKLE].position = -dataArray[54] * RADIANS_PER_COUNT + LANKLE_ENC_OFFSET;
    sensorData->jointData[LKNEE].position  = -dataArray[48] * RADIANS_PER_COUNT + LKNEE_ENC_OFFSET;
    sensorData->jointData[LHIP].position   = -dataArray[42] * RADIANS_PER_COUNT + LHIP_ENC_OFFSET;
    sensorData->jointData[LTOE].position   = sensorData->TorsoTilt
                                           - sensorData->jointData[LHIP].position
                                           - sensorData->jointData[LKNEE].position
                                           - sensorData->jointData[LANKLE].position;
    sensorData->hipData.L_rotation  = 0; //  dataArray[60]*RADIANS_PER_COUNT + LHIP_ROT_ENC_OFFSET; // hip unactuated joint
angles
    sensorData->hipData.L_abduction = 0; // -dataArray[66]*RADIANS_PER_COUNT + LHIP_ABD_ENC_OFFSET;

    sensorData->jointData[RANKLE].position = dataArray[12] * RADIANS_PER_COUNT + RANKLE_ENC_OFFSET;
    sensorData->jointData[RKNEE].position  = dataArray[6]  * RADIANS_PER_COUNT + RKNEE_ENC_OFFSET;
    sensorData->jointData[RHIP].position   = dataArray[0]  * RADIANS_PER_COUNT + RHIP_ENC_OFFSET;
    sensorData->jointData[RTOE].position   = sensorData->TorsoTilt
                                           - sensorData->jointData[RHIP].position
                                           - sensorData->jointData[RKNEE].position
                                           - sensorData->jointData[RANKLE].position;
    sensorData->hipData.R_rotation  = 0; //  dataArray[18]*RADIANS_PER_COUNT + RHIP_ROT_ENC_OFFSET; // hip unactuated joint
angles
    sensorData->hipData.R_abduction = 0; // -dataArray[24]*RADIANS_PER_COUNT + RHIP_ABD_ENC_OFFSET;
}


/* Function: GetJointVelocities
* ------------------------------------------------------------------------------------------------
* Reads angular velocity data from the FPGA and updates the sensorData structure.
* Angular velocities are for the  proximal segment relative to the distal segment.
* Velocity saturated at +/- 30 rad/sec =
*/
void GetJointVelocities(short        *dataArray,
                        SensorDataT *sensorData){

    int              i, j, signBit;
    int              arrayIndex[7]    = {25,55,49,43,13,7,1}; // array of velocity index numbers in dataArray
    int              velocityArray[7] = {0,0,0,0,0,0,0}; // {torso,lankle,lknee,lhip,rankle,rknee,rhip}
    short          tempvel2, tempvel1;
    int              tempvel3;
    static double unfilteredVel[7][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}};// Current
A[0] and Previous values
    static double filteredVel[7][4]   = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter
    static double torsoTiltPrevious   = 0;

    for(j=0; j<7; j++){
        i = arrayIndex[j];
        signBit  = (dataArray[i] & 0x8000)>>15; // Bit 15 is 1 for -ve; 0=+ve
        tempvel1 = dataArray[i] & 0x7FFF; // get rid of the sign bit
//        tempvel2 = dataArray[i+4] & 0xFF80; // get rest of velocity bits in footswitch value
//        velocityArray[j] =  (tempvel2 >> 7) ^ (tempvel1 << 9);     // bits 7-15 are velocity bits

        tempvel2 = dataArray[i+4];
        tempvel3 = tempvel2;
        velocityArray[j] = ((tempvel3 & 0xFF80) >> 7) ^ (tempvel1 << 9);     // bits 7-15 are velocity bits
        if (signBit == 1)
            velocityArray[j] = -velocityArray[j]; // adjust the sign
    }

    // the time between counts is given in units of 20MHz

    sensorData->torsoVelocity = (sensorData->TorsoTilt - torsoTiltPrevious)*FREQ; // Upper body
    torsoTiltPrevious = sensorData->TorsoTilt;

    sensorData->jointData[LANKLE].velocity = -20e6 * RADIANS_PER_COUNT/velocityArray[1];
    sensorData->jointData[LKNEE].velocity  = -20e6 * RADIANS_PER_COUNT/velocityArray[2];
    sensorData->jointData[LHIP].velocity   = -20e6 * RADIANS_PER_COUNT/velocityArray[3];
    sensorData->jointData[LTOE].velocity   = sensorData->torsoVelocity  - sensorData->jointData[LHIP].velocity
                                                                        - sensorData->jointData[LKNEE].velocity
                                                                        - sensorData->jointData[LANKLE].velocity;

    sensorData->jointData[RANKLE].velocity = 20e6 * RADIANS_PER_COUNT/velocityArray[4];
    sensorData->jointData[RKNEE].velocity  = 20e6 * RADIANS_PER_COUNT/velocityArray[5];
    sensorData->jointData[RHIP].velocity   = 20e6 * RADIANS_PER_COUNT/velocityArray[6];
    sensorData->jointData[RTOE].velocity   = sensorData->torsoVelocity  - sensorData->jointData[RHIP].velocity
                                                                        - sensorData->jointData[RKNEE].velocity
                                                                        - sensorData->jointData[RANKLE].velocity;

    // saturate velocities in case values are off due to FPGA or communication errors
    for(j=0;j<8;j++){
        if(sensorData->jointData[j].velocity > MAX_VELOCITY_SAT){  // default max is 30 rad/s
            sensorData->jointData[j].velocity = MAX_VELOCITY_SAT;
        }else if(sensorData->jointData[j].velocity < MIN_VELOCITY_SAT){ // default min is -30 rad/s
            sensorData->jointData[j].velocity = -MIN_VELOCITY_SAT;
        }
    }
```

```
    // filter velocities
    LowpassFilter(unfilteredVel[0], filteredVel[0], &sensorData->jointData[LANKLE].velocity, filterCoeffs2nd100); //
default = 100Hz
    LowpassFilter(unfilteredVel[1], filteredVel[1], &sensorData->jointData[LKNEE].velocity, filterCoeffs2nd100);  //
default = 10Hz to test velocity damping on knee
    LowpassFilter(unfilteredVel[2], filteredVel[2], &sensorData->jointData[LHIP].velocity, filterCoeffs2nd100);   //
default = 100Hz
    LowpassFilter(unfilteredVel[3], filteredVel[3], &sensorData->jointData[RANKLE].velocity, filterCoeffs2nd100); //
default = 100Hz
    LowpassFilter(unfilteredVel[4], filteredVel[4], &sensorData->jointData[RKNEE].velocity, filterCoeffs2nd100);  //
default = 10Hz to test velocity damping on knee
    LowpassFilter(unfilteredVel[5], filteredVel[5], &sensorData->jointData[RHIP].velocity, filterCoeffs2nd100);   //
default = 100Hz
    LowpassFilter(unfilteredVel[6], filteredVel[6], &sensorData->torsoVelocity, filterCoeffs2nd20);                  //
default = 20Hz
}


/* Function: GetSensorForces
 * ------------------------------------------------------------------------------------------------
 * Read cylinder force sensors. Force should be positive when sensor is in tension.
 */
void GetSensorForces(short            *dataArray,
                     SensorDataT    *sensorData){

    static double unfilteredForce[7][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; //
current A[0] and Previous values
    static double filteredForce[7][4]   = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used
in lowpass filter

    sensorData->jointData[LANKLE].sensorForce = (dataArray[57] * A_TO_D_CONVERSION) * LANKLE_FSENSOR_GAIN +
LANKLE_FSENSOR_OFFSET;
    sensorData->jointData[LKNEE].sensorForce  = (dataArray[51] * A_TO_D_CONVERSION) * LKNEE_FSENSOR_GAIN  +
LKNEE_FSENSOR_OFFSET;
    sensorData->jointData[LHIP].sensorForce   = (dataArray[45] * A_TO_D_CONVERSION) * LHIP_FSENSOR_GAIN   +
LHIP_FSENSOR_OFFSET;
    sensorData->jointData[RANKLE].sensorForce = (dataArray[15] * A_TO_D_CONVERSION) * RANKLE_FSENSOR_GAIN +
RANKLE_FSENSOR_OFFSET;
    sensorData->jointData[RKNEE].sensorForce  = (dataArray[9]  * A_TO_D_CONVERSION) * RKNEE_FSENSOR_GAIN  +
RKNEE_FSENSOR_OFFSET;
    sensorData->jointData[RHIP].sensorForce   = (dataArray[3]  * A_TO_D_CONVERSION) * RHIP_FSENSOR_GAIN   +
RHIP_FSENSOR_OFFSET;

//    LowpassFilter(unfilteredForce[LANKLE_T], filteredForce[LANKLE_T], &sensorData->jointData[LANKLE].sensorForce ,
filterCoeffs2nd20); // do not do this.  This introduces phase lag that leads to instability very quickly!!
//    LowpassFilter(unfilteredForce[LKNEE_T],  filteredForce[LKNEE_T],  &sensorData->jointData[LKNEE]. sensorForce ,
filterCoeffs2nd20); // do not do this.  This introduces phase lag that leads to instability very quickly!!
//    LowpassFilter(unfilteredForce[LHIP_T],   filteredForce[LHIP_T],   &sensorData->jointData[LHIP].  sensorForce ,
filterCoeffs2nd20); // do not do this.  This introduces phase lag that leads to instability very quickly!!
//    LowpassFilter(unfilteredForce[RANKLE_T], filteredForce[RANKLE_T], &sensorData->jointData[RANKLE].sensorForce ,
filterCoeffs2nd20); // do not do this.  This introduces phase lag that leads to instability very quickly!!
//    LowpassFilter(unfilteredForce[RKNEE_T],  filteredForce[RKNEE_T],  &sensorData->jointData[RKNEE]. sensorForce ,
filterCoeffs2nd20); // do not do this.  This introduces phase lag that leads to instability very quickly!!
//    LowpassFilter(unfilteredForce[RHIP_T],   filteredForce[RHIP_T],   &sensorData->jointData[RHIP].  sensorForce ,
filterCoeffs2nd20); // do not do this.  This introduces phase lag that leads to instability very quickly!!
}


/* Function: GetTorsoForce
 * ------------------------------------------------------------------------------------------------
 * Reads the torso interaction forces from the backpack sensors straingauges and computes the equivalent forces and
 * torque using the calibration matrix, offset vector and temperature compensation. Filters the results and
 * updates the sensorData structure.
 * X force = horizontal in the sagittal plane, pointing forward.
 * Z force = vertical in the sagittal plane, pointing upward.
 * Moment acts in the sagittal plane.
 */
void GetTorsoForce(short        *dataArray,
                   SensorDataT    *sensorData){

    double C[3][6] = {{CFT11, CFT12, CFT13, CFT14, CFT15, CFT16},
                      {CFT21, CFT22, CFT23, CFT24, CFT25, CFT26},
                      {CFT31, CFT32, CFT33, CFT34, CFT35, CFT36}}; // adjusted calibration matrix = C / Vin
    double Bt[6]; // temperature adjusted bias value vector (V)
    double FT[3]; // force/torque vector [Fx Fy Tz]
    double SGt_minus_Bt[6], one_minus_GS_x_thermister_minus_Ct;// intermediate variables
    int i; //counter

    // Read straingauge and thermister data
    sensorData->torsoForce.SG[0] = dataArray[62] * A_TO_D_CONVERSION;
    sensorData->torsoForce.SG[1] = dataArray[63] * A_TO_D_CONVERSION;
    sensorData->torsoForce.SG[2] = dataArray[64] * A_TO_D_CONVERSION;
    sensorData->torsoForce.SG[3] = dataArray[68] * A_TO_D_CONVERSION;
    sensorData->torsoForce.SG[4] = dataArray[69] * A_TO_D_CONVERSION;
    sensorData->torsoForce.SG[5] = dataArray[70] * A_TO_D_CONVERSION;
    sensorData->torsoForce.thermister = dataArray[20] * A_TO_D_CONVERSION + 4.96; // add 4.96V because of analog circuit
design

    // compute intermediate variables
    //CT = thermister value at calibration (from TWE.xls) , GS = thermister gain slope
    one_minus_GS_x_thermister_minus_Ct = 1 - GS*(sensorData->torsoForce.thermister - CT);

    // apply offset and temperature compensation on straingauge signals
    for (i=0; i<6; i++){ // for each straingauge
        Bt[i] = sensorData->torsoForce.SGt_bias[i];// get bias value
        // Apply temperature compensation
        sensorData->torsoForce.SGt[i] =    sensorData->torsoForce.SG[i]/one_minus_GS_x_thermister_minus_Ct;
        SGt_minus_Bt[i] = sensorData->torsoForce.SGt[i] - Bt[i];     // subtract adjusted bias from straingauge data
```

```
    }

    // Apply ATI F/T sensor's transformation matrix
    MatVectMult(FT, &C[0][0], SGt_minus_Bt, 3, 6);

    // torso force sensor outputs
    sensorData->torsoForce.Fx = -FT[0]; // torso operational force vector
    sensorData->torsoForce.Fy = -FT[1]; // i.e. forces acting on the machine torso
    sensorData->torsoForce.T  = -FT[2];

//    // FILTER DEBUGGING CODE use with second_0_filter_zoh.m in MATLAB
//
//            if (k==0){
//                if((fpRead =fopen("inputDataFile.txt","r")) == NULL){
//                        printf("Cannot open input file.\n");
//                }
//                if((fpWrite =fopen("outputDataFile.txt","w")) == NULL){
//                        printf("Cannot open output file.\n");
//                }
//            }
//
//            k++;
//            fscanf(fpRead,"%f \n", &sensorData->torsoForce.T);
//
//           // insert lowpass filter here
//
//            LowpassFilternew(unfilteredT, filteredT, &sensorData->torsoForce.T, filterCoeffs2nd5);
//
//            fprintf(fpWrite,"%f \n", sensorData->torsoForce.T);
//
//            if (k==10000){
//                fclose(fpRead);
//                fclose(fpWrite);
//            }
}


/* Function: ActuatorKinematics
* ---------------------------------------------------------------------------------------------
* compute kinematic actuator variables needed in the nonlinear control laws:
*
*             L : cylinder length (m)
*             h : cylinder moment arm (m)
*             xp : cylinder position (m)
*             xp_dot : cylinder velocity (m/s)
*       see hydraulicParameters.m
*/
void ActuatorKinematics(SensorDataT *sensorData ){

    double N, M, LA, LC, PHIA, PHIB, L, q, dq, L0_PLUS_XP0;
    int i; // valve number

    for (i=0; i<8; i++){   // compute for each joint

        q  = sensorData->jointData[i].position;
        dq = sensorData->jointData[i].velocity;

        // actuator in front
        if (i == LANKLE || i == RANKLE){
            N            = N_ANKLE;
            M            = M_ANKLE;
            LA           = LA_ANKLE;
            LC           = LC_ANKLE;
            PHIA         = PHIA_ANKLE;
            PHIB         = PHIB_ANKLE;
            L0_PLUS_XP0 = L0_PLUS_XP0_ANKLE;
            L = sqrt(N*cos(-PHIA-PHIB+q-Pi)-M); // cylinder length (m);
            sensorData->jointData[i].momentArm      = LA*sin(acos( (LC-L*L)/(-2*L*LA) )); // cylinder moment arm (m)
            sensorData->jointData[i].pistonPosition = L-L0_PLUS_XP0; // piston position from xp0 ref. (m)
            sensorData->jointData[i].pistonVelocity = -N*dq*sin(-PHIA-PHIB+q-Pi)/(2*sqrt(N*cos(-PHIA-PHIB+q-Pi)-M)); //
piston velocity (m/s)

        // actuator behind (this is taken into account in the computation and sign of the variables: moment arm should be
negative)
        } else if (i == LKNEE || i == RKNEE) {
            N            = N_KNEE;
            M            = M_KNEE;
            LA           = LA_KNEE;
            LC           = LC_KNEE;
            PHIA         = PHIA_KNEE;
            PHIB         = PHIB_KNEE;
            L0_PLUS_XP0 = L0_PLUS_XP0_KNEE;
            L = sqrt(N*cos(-PHIA-PHIB+Pi-q)-M); // cylinder length (m);
            sensorData->jointData[i].momentArm      = -LA*sin(acos((LC-L*L)/(-2*L*LA))); // cylinder moment arm (m)
            sensorData->jointData[i].pistonPosition = L-L0_PLUS_XP0; // piston position from xp0 ref. (m)
            sensorData->jointData[i].pistonVelocity = N*dq*sin(-PHIA-PHIB+Pi-q)/(2*sqrt(N*cos(-PHIA-PHIB+Pi-q)-M)); //
piston velocity (m/s)

        // actuator in front
        } else if (i == LHIP || i == RHIP) {
            N = N_HIP;
            M = M_HIP;
            LA = LA_HIP;
            LC = LC_HIP;
            PHIA = PHIA_HIP;
            PHIB = PHIB_HIP;
            L0_PLUS_XP0 = L0_PLUS_XP0_HIP;
            L = sqrt(N*cos(-PHIA-PHIB+q-Pi)-M); // cylinder length (m);
            sensorData->jointData[i].momentArm      = LA*sin(acos((LC-L*L)/(-2*L*LA))); // cylinder moment arm (m)
```

```
            sensorData->jointData[i].pistonPosition = L-L0_PLUS_XP0; // piston position from xp0 ref. (m)
            sensorData->jointData[i].pistonVelocity = -N*dq*sin(-PHIA-PHIB+q-Pi)/(2*sqrt(N*cos(-PHIA-PHIB+q-Pi)-M)); //
piston velocity (m/s)
        }
    }
}


/* Function: GetTorques
 * --------------------------------------------------------------------------------------------
 * Computes joint torques from rod force sensor data. Joint torque is of distal segment on proximal
 * segment in the direction of e03. Includes the torque due to actuator weight component
 * perpendicular to actuator axis.
 */
void GetTorques(SensorDataT *sensorData){

    double cylLength_2[6]; // [Lankle, Lknee, Lhip Rankle Rknee Rhip] cyl length^2   (m^2)
    double cylLength[6]; // [Lankle, Lknee, Lhip Rankle Rknee Rhip] cyl length (m)
    double Rs[6]; // actuator sensor joint reaction forces
                  // perpendicular to sensor axis. (N)
    double h[6]; // moments arms of Rs with joint. (m)
    double Tsgn[6] = {1,1,1,1,1,1}; // sign of the torque due to the actuator reaction force
    double qa[6]; // actuator angle with respect to gravity (rad)
/*
    // actuator orientations (rad)
    qa[LHIP_T]   = sensorData->TorsoTilt - sensorData->jointData[LHIP].position;
    qa[LKNEE_T]  = qa[LHIP_T] - sensorData->jointData[LKNEE].position;
    qa[LANKLE_T] = qa[LKNEE_T] - sensorData->jointData[LANKLE].position;
    qa[RHIP_T]   = sensorData->TorsoTilt - sensorData->jointData[RHIP].position;
    qa[RKNEE_T]  = qa[RHIP_T] - sensorData->jointData[RKNEE].position;
    qa[RANKLE_T] = qa[RKNEE_T] - sensorData->jointData[RANKLE].position;

    // cylinder lengths
    cylLength[LANKLE_T] = sensorData->jointData[LANKLE].pistonPosition + L0_PLUS_XP0_ANKLE;
    cylLength[LKNEE_T]  = sensorData->jointData[LKNEE].pistonPosition + L0_PLUS_XP0_KNEE;
    cylLength[LHIP_T]   = sensorData->jointData[LHIP].pistonPosition + L0_PLUS_XP0_HIP;
    cylLength[RANKLE_T] = sensorData->jointData[RANKLE].pistonPosition + L0_PLUS_XP0_ANKLE;
    cylLength[RKNEE_T]  = sensorData->jointData[RKNEE].pistonPosition + L0_PLUS_XP0_KNEE;
    cylLength[RHIP_T]   = sensorData->jointData[RHIP].pistonPosition + L0_PLUS_XP0_HIP;

    // reaction forces
    Rs[LANKLE_T] = (RCG_ANKLE + sensorData->jointData[LANKLE].pistonPosition ) * W_ANKLE_ACT_SENSOR * sin(qa[LANKLE_T]) /
cylLength[LANKLE_T]; // W_ANKLE_ACT_SENSOR = ma * g
    Rs[LKNEE_T]  = (RCG_KNEE + sensorData->jointData[LKNEE].pistonPosition )   * W_KNEE_ACT_SENSOR  * sin(qa[LKNEE_T])  /
cylLength[LKNEE_T];
    Rs[LHIP_T]   = (RCG_HIP + sensorData->jointData[LHIP].pistonPosition )     * W_HIP_ACT_SENSOR   * sin(qa[LHIP_T])   /
cylLength[LHIP_T];
    Rs[RANKLE_T] = (RCG_ANKLE + sensorData->jointData[RANKLE].pistonPosition ) * W_ANKLE_ACT_SENSOR * sin(qa[RANKLE_T]) /
cylLength[RANKLE_T];
    Rs[RKNEE_T]  = (RCG_KNEE + sensorData->jointData[RKNEE].pistonPosition )   * W_KNEE_ACT_SENSOR  * sin(qa[RKNEE_T])  /
cylLength[RKNEE_T];
    Rs[RHIP_T]   = (RCG_HIP + sensorData->jointData[RHIP].pistonPosition )     * W_HIP_ACT_SENSOR   * sin(qa[RHIP_T])   /
cylLength[RHIP_T];

    // reaction force moment arms
    h[LANKLE_T] = sqrt(LB_2_ANKLE - sensorData->jointData[LANKLE].momentArm * sensorData->jointData[LANKLE].momentArm);
    h[LKNEE_T]  = sqrt(LB_2_KNEE  - sensorData->jointData[LKNEE].momentArm  * sensorData->jointData[LKNEE].momentArm);
    h[LHIP_T]   = sqrt(LA_2_HIP   - sensorData->jointData[LHIP].momentArm   * sensorData->jointData[LHIP].momentArm);
    h[RANKLE_T] = sqrt(LB_2_ANKLE - sensorData->jointData[RANKLE].momentArm * sensorData->jointData[RANKLE].momentArm);
    h[RKNEE_T]  = sqrt(LB_2_KNEE  - sensorData->jointData[RKNEE].momentArm  * sensorData->jointData[RKNEE].momentArm);
    h[RHIP_T]   = sqrt(LA_2_HIP   - sensorData->jointData[RHIP].momentArm   * sensorData->jointData[RHIP].momentArm);

    // cylinder lengths^2
    cylLength_2[LANKLE_T] = cylLength[LANKLE_T] * cylLength[LANKLE_T]; // ankle cylinder length^2 (m)
    cylLength_2[LKNEE_T]  = cylLength[LKNEE_T]  * cylLength[LKNEE_T];  // knee  cylinder length^2 (m)
    cylLength_2[LHIP_T]   = cylLength[LHIP_T]   * cylLength[LHIP_T];   // hip   cylinder length^2 (m)
    cylLength_2[RANKLE_T] = cylLength[RANKLE_T] * cylLength[RANKLE_T]; // ankle cylinder length^2 (m)
    cylLength_2[RKNEE_T]  = cylLength[RKNEE_T]  * cylLength[RKNEE_T];  // knee  cylinder length^2 (m)
    cylLength_2[RHIP_T]   = cylLength[RHIP_T]   * cylLength[RHIP_T];   // hip   cylinder length^2 (m)

    // reaction force torque directions
    if(cylLength_2[LANKLE_T] > -LC_ANKLE)
        Tsgn[LANKLE_T] = -1; // LC = LB^2 - LA^2
    if(cylLength_2[LKNEE_T]  > -LC_KNEE)
        Tsgn[LKNEE_T]  = -1;
    if(cylLength_2[LHIP_T]   >  LC_HIP)
        Tsgn[LHIP_T]   = -1;
    if(cylLength_2[RANKLE_T] > -LC_ANKLE)
        Tsgn[RANKLE_T] = -1;
    if(cylLength_2[RKNEE_T]  > -LC_KNEE)
        Tsgn[RKNEE_T]  = -1;
    if(cylLength_2[RHIP_T]   >  LC_HIP)
        Tsgn[RHIP_T]   = -1;
*/
    sensorData->jointData[LANKLE].torque = -sensorData->jointData[LANKLE].sensorForce * sensorData-
>jointData[LANKLE].momentArm; // + Tsgn[LANKLE_T] * h[LANKLE_T] * Rs[LANKLE_T]; // this causes Torque to peak to inf
sometimes (?)
    sensorData->jointData[LKNEE].torque  = -sensorData->jointData[LKNEE].sensorForce  * sensorData-
>jointData[LKNEE].momentArm;  // + Tsgn[LKNEE_T]  * h[LKNEE_T]  * Rs[LKNEE_T];
    sensorData->jointData[LHIP].torque   = -sensorData->jointData[LHIP].sensorForce   * sensorData-
>jointData[LHIP].momentArm;   // + Tsgn[LHIP_T]   * h[LHIP_T]   * Rs[LHIP_T];
    sensorData->jointData[RANKLE].torque = -sensorData->jointData[RANKLE].sensorForce * sensorData-
>jointData[RANKLE].momentArm; // + Tsgn[RANKLE_T] * h[RANKLE_T] * Rs[RANKLE_T];
    sensorData->jointData[RKNEE].torque  = -sensorData->jointData[RKNEE].sensorForce  * sensorData-
>jointData[RKNEE].momentArm;  // + Tsgn[RKNEE_T]  * h[RKNEE_T]  * Rs[RKNEE_T];
    sensorData->jointData[RHIP].torque   = -sensorData->jointData[RHIP].sensorForce   * sensorData-
>jointData[RHIP].momentArm;   // + Tsgn[RHIP_T]   * h[RHIP_T]   * Rs[RHIP_T];
}
```

```
/* Function: GetFootSwitches
 * ---------------------------
 * Reads footswitch data and updates the sensorData structure.
 */
void GetFootSwitches(short *dataArray, SensorDataT *sensorData){

     static double filteredFoot[10][4]   =
{{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in lowpass
filter { Lball Lmidfoot Lheel  Rball Rmidfoot Rheel}
    static double unfilteredFoot[10][4] =
{{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Previous values
    double Lihuafootswitch[10];
    int j;

    // switch array: [TIP TOE BALL MIDFOOT HEEL]

    // right foot
    Lihuafootswitch[0] = (dataArray[17] & 1);                 // tip
    Lihuafootswitch[1] = (dataArray[17] & 2)  ==  2 ? 1 : 0; // toe
    Lihuafootswitch[2] = (dataArray[17] & 4)  ==  4 ? 1 : 0; // ball
    Lihuafootswitch[3] = (dataArray[17] & 8)  ==  8 ? 1 : 0; // midfoot
    Lihuafootswitch[4] = (dataArray[17] & 16) == 16 ? 1 : 0; // heel

    // left foot
    Lihuafootswitch[5] = (dataArray[59] & 1);                 // tip
    Lihuafootswitch[6] = (dataArray[59] & 2)  ==  2 ? 1 : 0; // toe
    Lihuafootswitch[7] = (dataArray[59] & 4)  ==  4 ? 1 : 0; // ball
    Lihuafootswitch[8] = (dataArray[59] & 8)  ==  8 ? 1 : 0; // midfoot
    Lihuafootswitch[9] = (dataArray[59] & 16) == 16 ? 1 : 0; // heel

    // filter footswitches to de-bounce switches
    for(j=0;j<10; j++) {
        LowpassFilter(unfilteredFoot[j], filteredFoot[j], &Lihuafootswitch[j], filterCoeffs1st5); // default = 10Hz

        // apply threshold to convert analog outut of filter to digital value
        if (Lihuafootswitch[j]>0.5)    {
            Lihuafootswitch[j] = 1;
        }
        else {
            Lihuafootswitch[j] = 0;
        }
    }

    // save filter values back into structure
    sensorData->Rfootswitch[TIP]     = (int) Lihuafootswitch[0];
    sensorData->Rfootswitch[TOE]     = (int) Lihuafootswitch[1];
    sensorData->Rfootswitch[BALL]    = (int) Lihuafootswitch[2];
    sensorData->Rfootswitch[MIDFOOT] = (int) Lihuafootswitch[3];
    sensorData->Rfootswitch[HEEL]    = (int) Lihuafootswitch[4];
    sensorData->Lfootswitch[TIP]     = (int) Lihuafootswitch[5];
    sensorData->Lfootswitch[TOE]     = (int) Lihuafootswitch[6];
    sensorData->Lfootswitch[BALL]    = (int) Lihuafootswitch[7];
    sensorData->Lfootswitch[MIDFOOT] = (int) Lihuafootswitch[8];
    sensorData->Lfootswitch[HEEL]    = (int) Lihuafootswitch[9];

    // use this section to fix the stance of the exo (eg: for use on stance plate)
    // ON  = 1
    // OFF = 0

//    // right foot
//    sensorData->Rfootswitch[TIP]     = 1;    // TIP
//    sensorData->Rfootswitch[TOE]     = 1;    // TOE
//    sensorData->Rfootswitch[BALL]    = 1;    // BALL
//    sensorData->Rfootswitch[MIDFOOT] = 1;    // MIDFOOT
//    sensorData->Rfootswitch[HEEL]    = 1;    // HEEL

//    // left foot
//    sensorData->Lfootswitch[TIP]     = 1;    // TIP
//    sensorData->Lfootswitch[TOE]     = 1;    // TOE
//    sensorData->Lfootswitch[BALL]    = 1;    // BALL
//    sensorData->Lfootswitch[MIDFOOT] = 1;    // MIDFOOT
//    sensorData->Lfootswitch[HEEL]    = 1;    // HEEL
}


/* Function: GetEncoderIndexPulses
 * ---------------------------------------------------------------------------------------------
 * Reads encoder index pulse data and updates the sensorData structure.
 */
void GetEncoderIndexPulses(short        *dataArray,
                           SensorDataT    *sensorData){

    // encoder pulse is on Bit 6
    // 0x40 = 64 = 1000000 in binary
    // & 0x40 means mask all but first bit
    // shift left by 6 means pick of the last bit (0 or 1)

    sensorData->jointData[LANKLE].indexPulse = (dataArray[59] & 0x40) >> 6;
    sensorData->jointData[LKNEE].indexPulse  = (dataArray[53] & 0x40) >> 6;
    sensorData->jointData[LHIP].indexPulse   = (dataArray[47] & 0x40) >> 6;

    sensorData->jointData[RANKLE].indexPulse = (dataArray[17] & 0x40) >> 6;
    sensorData->jointData[RKNEE].indexPulse  = (dataArray[11] & 0x40) >> 6;
    sensorData->jointData[RHIP].indexPulse   = (dataArray[5]  & 0x40) >> 6;

    sensorData->hipData.R_abduction_indexP = (dataArray[29] & 0x40) >> 6;
    sensorData->hipData.R_rotation_indexP  = (dataArray[23] & 0x40) >> 6;
    sensorData->hipData.L_abduction_indexP = (dataArray[71] & 0x40) >> 6;
```

```
    sensorData->hipData.L_rotation_indexP  = (dataArray[65] & 0x40) >> 6;
}

/* Function: DetectErrors
* ------------------------------------------------------------------------------------------
* Set sensorData.error to 1 if a signal is out-of-range
*/
void DetectErrors(SensorDataT *sensorData){

    // Check if any data is out-of-range

    if ( sensorData->TorsoTilt < TORSO_MIN
        || sensorData->TorsoTilt > TORSO_MAX) sensorData->error = 126;  // torso tilt

    else if ( sensorData->jointData[LANKLE].position < ANKLE_MIN -0.2 ) sensorData->error = 101;// joint positions
    else if ( sensorData->jointData[LANKLE].position > ANKLE_MAX +0.2 ) sensorData->error = 101;

    else if ( sensorData->jointData[LKNEE].position  < KNEE_MIN -0.2 ) sensorData->error = 102;
    else if ( sensorData->jointData[LKNEE].position  > KNEE_MAX +0.2 ) sensorData->error = 102;

    else if ( sensorData->jointData[LHIP].position   < HIP_MIN -0.2   ) sensorData->error = 103;
    else if ( sensorData->jointData[LHIP].position   > HIP_MAX +0.2 ) sensorData->error   = 103;

    else if ( sensorData->jointData[RANKLE].position < ANKLE_MIN -0.2 ) sensorData->error = 107;
    else if ( sensorData->jointData[RANKLE].position > ANKLE_MAX +0.2 ) sensorData->error = 107;

    else if ( sensorData->jointData[RKNEE].position  < KNEE_MIN -0.2 ) sensorData->error = 108;
    else if ( sensorData->jointData[RKNEE].position  > KNEE_MAX +0.2 ) sensorData->error = 108;

    else if ( sensorData->jointData[RHIP].position   < HIP_MIN -0.2) sensorData->error   = 109;
    else if ( sensorData->jointData[RHIP].position   > HIP_MAX +0.2  ) sensorData->error = 109;

    else if ( sensorData->hipData.R_abduction        < HIP_ABD_MIN -0.2) sensorData->error  = 110;
    else if ( sensorData->hipData.R_abduction        > HIP_ABD_MAX +0.2 ) sensorData->error = 110;

    else if ( sensorData->hipData.L_abduction        < HIP_ABD_MIN -0.2) sensorData->error  = 104;
    else if ( sensorData->hipData.L_abduction        > HIP_ABD_MAX +0.2 ) sensorData->error = 104;

    else if ( sensorData->hipData.R_rotation         < HIP_ROT_MIN -0.2) sensorData->error  = 111;
    else if ( sensorData->hipData.R_rotation         > HIP_ROT_MAX +0.2 ) sensorData->error = 111;

    else if ( sensorData->hipData.L_rotation         < HIP_ROT_MIN -0.2) sensorData->error  = 105;
    else if ( sensorData->hipData.L_rotation         > HIP_ROT_MAX +0.2 ) sensorData->error = 105;

    else if ( sensorData->jointData[LANKLE].velocity < -VMAX  ) sensorData->error = 201;  // joint velocities
    else if ( sensorData->jointData[LANKLE].velocity > VMAX ) sensorData->error   = 201;

    else if (   sensorData->jointData[LKNEE].velocity< -VMAX ) sensorData->error = 202;
    else if ( sensorData->jointData[LKNEE].velocity  > VMAX ) sensorData->error  = 202;

    else if ( sensorData->jointData[LHIP].velocity   < -VMAX ) sensorData->error = 203;
    else if ( sensorData->jointData[LHIP].velocity   > VMAX ) sensorData->error  = 203;

    else if ( sensorData->jointData[RANKLE].velocity < -VMAX ) sensorData->error = 207;
    else if ( sensorData->jointData[RANKLE].velocity > VMAX ) sensorData->error  = 207;

    else if (   sensorData->jointData[RKNEE].velocity  < -VMAX ) sensorData->error = 208;
    else if ( sensorData->jointData[RKNEE].velocity  > VMAX ) sensorData->error    = 208;

    else if ( sensorData->jointData[RHIP].velocity   < -VMAX ) sensorData->error = 209;
    else if ( sensorData->jointData[RHIP].velocity   > VMAX  ) sensorData->error = 209;

    else if ( sensorData->bodyAccel[LFOOT].angular_accel  < -AMAX ) sensorData->error = 320;// joint accelerations
    else if ( sensorData->bodyAccel[LFOOT].angular_accel  > AMAX ) sensorData->error  = 320;

    else if ( sensorData->bodyAccel[LSHANK].angular_accel < -AMAX ) sensorData->error = 321;
    else if ( sensorData->bodyAccel[LSHANK].angular_accel > AMAX  ) sensorData->error = 302;

    else if ( sensorData->bodyAccel[LTHIGH].angular_accel < -AMAX ) sensorData->error = 322;
    else if ( sensorData->bodyAccel[LTHIGH].angular_accel > AMAX ) sensorData->error  = 322;

    else if ( sensorData->bodyAccel[RFOOT].angular_accel  < -AMAX ) sensorData->error = 323;
    else if ( sensorData->bodyAccel[RFOOT].angular_accel  > AMAX ) sensorData->error  = 323;

    else if ( sensorData->bodyAccel[RSHANK].angular_accel < -AMAX ) sensorData->error = 324;
    else if ( sensorData->bodyAccel[RSHANK].angular_accel > AMAX  ) sensorData->error = 324;

    else if ( sensorData->bodyAccel[RTHIGH].angular_accel < -AMAX ) sensorData->error = 325;
    else if ( sensorData->bodyAccel[RTHIGH].angular_accel > AMAX ) sensorData->error  = 325;

    else if ( sensorData->bodyAccel[UPPERBODY].angular_accel < -AMAX ) sensorData->error = 326;
    else if ( sensorData->bodyAccel[UPPERBODY].angular_accel > AMAX  ) sensorData->error = 326;

    else if (   sensorData->jointData[LANKLE].sensorForce < -FMAX ) sensorData->error = 401;   // actuator sensor forces
    else if (   sensorData->jointData[LANKLE].sensorForce > FMAX ) sensorData->error  = 401;

    else if (   sensorData->jointData[LKNEE].sensorForce  < -FMAX ) sensorData->error = 402;
    else if (   sensorData->jointData[LKNEE].sensorForce  > FMAX ) sensorData->error  = 402;

    else if (   sensorData->jointData[LHIP].sensorForce   < -FMAX ) sensorData->error = 403;
    else if (   sensorData->jointData[LHIP].sensorForce   > FMAX ) sensorData->error  = 403;

    else if (   sensorData->jointData[RANKLE].sensorForce < -FMAX ) sensorData->error = 407;
    else if (   sensorData->jointData[RANKLE].sensorForce > FMAX ) sensorData->error  = 407;

    else if ( sensorData->jointData[RKNEE].sensorForce       < -FMAX ) sensorData->error = 408;
    else if ( sensorData->jointData[RKNEE].sensorForce    > FMAX ) sensorData->error  = 408;
```

```
    else if (   sensorData->jointData[RHIP].sensorForce   < -FMAX ) sensorData->error = 409;
    else if (   sensorData->jointData[RHIP].sensorForce   > FMAX ) sensorData->error  = 409;

    /*                else if ( sensorData->torsoForce.Fx < -FTX_MAX ) sensorData->error = 530;              // torso
force sensor
    else if ( sensorData->torsoForce.Fx > FTX_MAX  ) sensorData->error = 530;

      else if ( sensorData->torsoForce.Fy < -FTY_MAX ) sensorData->error = 531;
      else if ( sensorData->torsoForce.Fy > FTY_MAX  ) sensorData->error = 531;

        else if ( sensorData->torsoForce.T  < -TMAX    ) sensorData->error = 532;
        else if ( sensorData->torsoForce.T  > TMAX     ) sensorData->error = 532;

          else if (sensorData->torsoForce.SG[0]  > SGMAX
          || sensorData->torsoForce.SG[0]  < -SGMAX
          || sensorData->torsoForce.SG[1]  > SGMAX
          || sensorData->torsoForce.SG[1]  < -SGMAX
          || sensorData->torsoForce.SG[2]  > SGMAX
          || sensorData->torsoForce.SG[2]  < -SGMAX
          || sensorData->torsoForce.SG[3]  > SGMAX
          || sensorData->torsoForce.SG[3]  < -SGMAX
          || sensorData->torsoForce.SG[4]  > SGMAX
          || sensorData->torsoForce.SG[4]  < -SGMAX
          || sensorData->torsoForce.SG[5]  > SGMAX
          || sensorData->torsoForce.SG[5]  < -SGMAX ) sensorData->error = 533;

            else if (sensorData->torsoForce.thermister > THERM_MAX
            || sensorData->torsoForce.thermister < THERM_MIN ) sensorData->error = 534;

              else if ( (sensorData->Rfootswitch[TIP]     != 0  &&  sensorData->Rfootswitch[TIP] != 1) ) sensorData->error
= 645;
              else if ( (sensorData->Rfootswitch[TOE]     != 0  &&  sensorData->Rfootswitch[TOE] != 1) ) sensorData->error
= 646;   // footswitches
              else if ( (sensorData->Rfootswitch[BALL]    != 0  &&  sensorData->Rfootswitch[BALL] != 1) ) sensorData->error
= 647;
              else if ( (sensorData->Rfootswitch[MIDFOOT] != 0  &&  sensorData->Rfootswitch[MIDFOOT] != 1) ) sensorData-
>error = 648;
              else if ( (sensorData->Rfootswitch[HEEL]    != 0  &&  sensorData->Rfootswitch[HEEL] != 1) ) sensorData->error
= 649;

                else if ( (sensorData->Lfootswitch[TIP]     != 0  &&  sensorData->Lfootswitch[TIP] != 1) ) sensorData-
>error = 640;
                else if ( (sensorData->Lfootswitch[TOE]     != 0  &&  sensorData->Lfootswitch[TOE] != 1) ) sensorData-
>error = 641;
                else if ( (sensorData->Lfootswitch[BALL]    != 0  &&  sensorData->Lfootswitch[BALL] != 1) ) sensorData-
>error = 642;
                else if ( (sensorData->Lfootswitch[MIDFOOT] != 0  &&  sensorData->Lfootswitch[MIDFOOT] != 1) ) sensorData-
>error = 643;
                else if ( (sensorData->Lfootswitch[HEEL]    != 0  &&  sensorData->Lfootswitch[HEEL] != 1) ) sensorData-
>error = 644;

                  else if (  sensorData->jointData[LANKLE].pistonPosition < LONG_PISTON_POS_MIN // piston position
                  || sensorData->jointData[LANKLE].pistonPosition > LONG_PISTON_POS_MAX ) sensorData->error = 701;
                  else if (  sensorData->jointData[LKNEE].pistonPosition < SHORT_PISTON_POS_MIN
                  || sensorData->jointData[LKNEE].pistonPosition > SHORT_PISTON_POS_MAX ) sensorData->error = 702;
                  else if (  sensorData->jointData[LHIP].pistonPosition < LONG_PISTON_POS_MIN
                  || sensorData->jointData[LHIP].pistonPosition > LONG_PISTON_POS_MAX ) sensorData->error = 703;
                  else if (  sensorData->jointData[RANKLE].pistonPosition < LONG_PISTON_POS_MIN
                  || sensorData->jointData[RANKLE].pistonPosition > LONG_PISTON_POS_MAX ) sensorData->error = 707;
                  else if (  sensorData->jointData[RKNEE].pistonPosition < SHORT_PISTON_POS_MIN
                  || sensorData->jointData[RKNEE].pistonPosition > SHORT_PISTON_POS_MAX ) sensorData->error = 708;
                  else if (  sensorData->jointData[RHIP].pistonPosition < LONG_PISTON_POS_MIN
                  || sensorData->jointData[RHIP].pistonPosition > LONG_PISTON_POS_MAX ) sensorData->error = 709;

                    else if (  sensorData->jointData[LANKLE].momentArm <  ANKLE_MOMENT_ARM_MIN     // actuator moment arms
                    || sensorData->jointData[LANKLE].momentArm >  ANKLE_MOMENT_ARM_MAX ) sensorData->error = 801;
                    else if (  sensorData->jointData[LKNEE].momentArm <  KNEE_MOMENT_ARM_MIN
                    || sensorData->jointData[LKNEE].momentArm >  KNEE_MOMENT_ARM_MAX ) sensorData->error = 802;
                    else if (  sensorData->jointData[LHIP].momentArm <  HIP_MOMENT_ARM_MIN
                    || sensorData->jointData[LHIP].momentArm >  HIP_MOMENT_ARM_MAX ) sensorData->error = 803;
                    else if (  sensorData->jointData[RANKLE].momentArm <  ANKLE_MOMENT_ARM_MIN
                    || sensorData->jointData[RANKLE].momentArm >  ANKLE_MOMENT_ARM_MAX ) sensorData->error = 807;
                    else if (  sensorData->jointData[RKNEE].momentArm <  KNEE_MOMENT_ARM_MIN
                    || sensorData->jointData[RKNEE].momentArm >  KNEE_MOMENT_ARM_MAX ) sensorData->error = 808;
                    else if (  sensorData->jointData[RHIP].momentArm <  HIP_MOMENT_ARM_MIN
                    || sensorData->jointData[RHIP].momentArm >  HIP_MOMENT_ARM_MAX ) sensorData->error = 809;

                      else if ( sensorData->forceDistribution.LankleDistance > FOOT_DISTANCE_MAX ) sensorData->error = 901;
// transverse plane distance from CG to ankles
                        else if ( sensorData->forceDistribution.RankleDistance > FOOT_DISTANCE_MAX ) sensorData->error = 907;
    */
    else (sensorData->error = 0);

}

/* ERROR CODE DEFINITIONS

000 no data
100    position
200 velocity
300 ang acceleration
400 actuator force
500 torso force sensor
600    footswitches
700 piston position
800 actuator moment arm
900 ankle distance to CG
1000 communication with FPGA lost  ( this is set by the Supervisor I/O for the GUI)
1100 FPGA data missed twice in a row
```

```
0       Ltoe
1       Lankle
2       Lknee
3       Lhip
4       Lhip abduction
5       Lhip rotation
6       Rtoe
7       Rankle
8       Rknee
9       Rhip
10       Rhip abduction
11       Rhip rotation

20       Lfoot
21       Lshank
22       Lthigh
23       Rfoot
24       Rshank
25       Rthigh
26       Upper body

30       Fx
31       Fy
32       Tz
33       straingage
34       thermister

40       Ltip
41       Ltoe
42       Lball
43       Lmidfoot
44       Lheel
45       Rtip
46       Rtoe
47       Rball
48       Rmidfoot
49       Rheel

90       no specifier

*/
```

Appendix A.6 – Accel.h

```
/* Function: GetBodyAccelerations
 * ------------------------------
 * Reads angular acceleration data from the accelerometers and updates the sensorData structure.
 */
void GetBodyAccelerations(short *dataArray, SensorDataT *sensorData);


/* Function: GetJointAccelerations
 * -------------------------------
 * Computes joint angular acclerations and updates the sensorData structure.
 */
void GetJointAccelerations(SensorDataT *sensorData);


/* Function: GetJointAccelerationsVelocityBased
 * ------------------------------------------------------------------------------------------
 * Computes joint angular acclerations and updates the sensorData structure.
 * Accelerations are for the distal segment relative to the proximal segment
 */
void GetJointAccelerationsVelocityBased(SensorDataT *sensorData);
```

Appendix A.7 – Accel.c

```
#include <math.h>
#include <stdio.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"
#include "Accel.h"
#include "DSup.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];

extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];


/* Function: GetBodyAccelerations
 * --------------------------------------------------------------------------------------------
 * Reads angular acceleration data from the accelerometers and updates the sensorData structure.
 */
void GetBodyAccelerations(short         *dataArray,
                          SensorDataT *sensorData){

    double lin1[7], lin2[7]; // accelerometer output [LFOOT LSHANK LTHIGH RFOOT RSHANK RTHIGH UPPERBODY]
    static int k = 0; // for debugging
    static FILE *fpWrite; // file pointers    for debugging

    //  using two linear accelerometers to find body angular acceleration
    //read accelerometers( note: left foot accelerometers are not in the same orientation as the rest)
    lin1[LFOOT]     = -dataArray[58] * A_TO_D_CONVERSION; // top accelerometer (ADC2)
    lin1[LSHANK]    =  dataArray[52] * A_TO_D_CONVERSION;
    lin1[LTHIGH]    =  dataArray[46] * A_TO_D_CONVERSION;
    lin1[RFOOT]     =  dataArray[16] * A_TO_D_CONVERSION;
    lin1[RSHANK]    =  dataArray[10] * A_TO_D_CONVERSION;
    lin1[RTHIGH]    =  dataArray[4]  * A_TO_D_CONVERSION;
    lin1[UPPERBODY] =  dataArray[28] * A_TO_D_CONVERSION;

    lin2[LFOOT]     = -dataArray[56] * A_TO_D_CONVERSION; // bottom accelerometers (ADC8)
    lin2[LSHANK]    =  dataArray[50] * A_TO_D_CONVERSION;
    lin2[LTHIGH]    =  dataArray[44] * A_TO_D_CONVERSION;
    lin2[RFOOT]     =  dataArray[14] * A_TO_D_CONVERSION;
    lin2[RSHANK]    =  dataArray[8]  * A_TO_D_CONVERSION;
    lin2[RTHIGH]    =  dataArray[2]  * A_TO_D_CONVERSION;
    lin2[UPPERBODY] =  dataArray[26] * A_TO_D_CONVERSION;

    // USING GAIN FROM SPECS and computed offsets Ap 15
    sensorData->bodyAccel[RFOOT].gain1      = RFOOT_ACC_GAIN1;          //12.85526686;    // top     R1   RIOM 28
    sensorData->bodyAccel[RFOOT].offset1  = RFOOT_ACC_OFFSET1;          //-0.888567933;
    sensorData->bodyAccel[RFOOT].gain2      = RFOOT_ACC_GAIN2;          //13.37642633;    // bottom
    sensorData->bodyAccel[RFOOT].offset2  = RFOOT_ACC_OFFSET2;          //-0.868156195;

    // Using computed gains & offset
    sensorData->bodyAccel[RSHANK].gain1      = RSHANK_ACC_GAIN1;          // 13.38874545;    // top // RIOM 13
    sensorData->bodyAccel[RSHANK].offset1 = RSHANK_ACC_OFFSET1;          //0.888483984;
    sensorData->bodyAccel[RSHANK].gain2      = RSHANK_ACC_GAIN2;          //13.37889706;    // bottom
    sensorData->bodyAccel[RSHANK].offset2 = RSHANK_ACC_OFFSET2;          //-0.835178656;

    sensorData->bodyAccel[RTHIGH].gain1      = RTHIGH_ACC_GAIN1;          //5.876894451;    // top
    sensorData->bodyAccel[RTHIGH].offset1 = RTHIGH_ACC_OFFSET1;          // 0.848491186;
    sensorData->bodyAccel[RTHIGH].gain2      = RTHIGH_ACC_GAIN2;          //12.5546848;     // bottom
    sensorData->bodyAccel[RTHIGH].offset2 = RTHIGH_ACC_OFFSET2;          //0.820738499;

    sensorData->bodyAccel[LFOOT].gain1      = LFOOT_ACC_GAIN1;          //-13.61289758;    // top // L3 and RIOM#28
    sensorData->bodyAccel[LFOOT].offset1  = LFOOT_ACC_OFFSET1;          //-0.882320267;
    sensorData->bodyAccel[LFOOT].gain2      = LFOOT_ACC_GAIN2;          //-13.83991511;    // bottom
    sensorData->bodyAccel[LFOOT].offset2  = LFOOT_ACC_OFFSET2;          //-0.818183387;

    sensorData->bodyAccel[LSHANK].gain1      = LSHANK_ACC_GAIN1;          //12.99961686;    // top
    sensorData->bodyAccel[LSHANK].offset1 = LSHANK_ACC_OFFSET1;          //-0.832516644;
    sensorData->bodyAccel[LSHANK].gain2      = LSHANK_ACC_GAIN2;          //12.95494464;    // bottom
    sensorData->bodyAccel[LSHANK].offset2 = LSHANK_ACC_OFFSET2;          // -0.832714978;

    sensorData->bodyAccel[LTHIGH].gain1      = LTHIGH_ACC_GAIN1;          //5.051387685;// 5.81827332;    // top
    sensorData->bodyAccel[LTHIGH].offset1 = LTHIGH_ACC_OFFSET1;          //-0.866331814; // -0.856421669;
    sensorData->bodyAccel[LTHIGH].gain2      = LTHIGH_ACC_GAIN2;          //12.81899544; //12.71754197;    // bottom
    sensorData->bodyAccel[LTHIGH].offset2 = LTHIGH_ACC_OFFSET2;       //-0.814218852; //-0.888885387;

    sensorData->bodyAccel[UPPERBODY].gain1     = UPPERBODY_ACC_GAIN1;   //-5.888841371;    // top
    sensorData->bodyAccel[UPPERBODY].offset1 = UPPERBODY_ACC_OFFSET1; // -0.852321259;
    sensorData->bodyAccel[UPPERBODY].gain2     = UPPERBODY_ACC_GAIN2;   //-5.120388993;    // bottom
```

```
    sensorData->bodyAccel[UPPERBODY].offset2 = UPPERBODY_ACC_OFFSET2; //-0.054735767;

    // calculate linear accelerations

    sensorData->bodyAccel[LFOOT].lin_accel1     = (lin1[LFOOT] + sensorData->bodyAccel[LFOOT].offset1)
                                                * sensorData->bodyAccel[LFOOT].gain1;

    sensorData->bodyAccel[LSHANK].lin_accel1    = (lin1[LSHANK] + sensorData->bodyAccel[LSHANK].offset1)
                                                * sensorData->bodyAccel[LSHANK].gain1;

    sensorData->bodyAccel[LTHIGH].lin_accel1    = (lin1[LTHIGH] + sensorData->bodyAccel[LTHIGH].offset1)
                                                * sensorData->bodyAccel[LTHIGH].gain1 ;

    sensorData->bodyAccel[RFOOT].lin_accel1     = (lin1[RFOOT] + sensorData->bodyAccel[RFOOT].offset1)
                                                * sensorData->bodyAccel[RFOOT].gain1;

    sensorData->bodyAccel[RSHANK].lin_accel1    = (lin1[RSHANK] + sensorData->bodyAccel[RSHANK].offset1)
                                                * sensorData->bodyAccel[RSHANK].gain1;

    sensorData->bodyAccel[RTHIGH].lin_accel1    = (lin1[RTHIGH] + sensorData->bodyAccel[RTHIGH].offset1)
                                                * sensorData->bodyAccel[RTHIGH].gain1;

    sensorData->bodyAccel[UPPERBODY].lin_accel1 = (lin1[UPPERBODY] + sensorData->bodyAccel[UPPERBODY].offset1)
                                                * sensorData->bodyAccel[UPPERBODY].gain1;

    sensorData->bodyAccel[LFOOT].lin_accel2        = (lin2[LFOOT] + sensorData->bodyAccel[LFOOT].offset2)
                                                * sensorData->bodyAccel[LFOOT].gain2;

    sensorData->bodyAccel[LSHANK].lin_accel2    = (lin2[LSHANK] + sensorData->bodyAccel[LSHANK].offset2)
                                                * sensorData->bodyAccel[LSHANK].gain2;

    sensorData->bodyAccel[LTHIGH].lin_accel2    = (lin2[LTHIGH] + sensorData->bodyAccel[LTHIGH].offset2)
                                                * sensorData->bodyAccel[LTHIGH].gain2;

    sensorData->bodyAccel[RFOOT].lin_accel2        = (lin2[RFOOT] + sensorData->bodyAccel[RFOOT].offset2)
                                                * sensorData->bodyAccel[RFOOT].gain2;

    sensorData->bodyAccel[RSHANK].lin_accel2    = (lin2[RSHANK] + sensorData->bodyAccel[RSHANK].offset2)
                                                * sensorData->bodyAccel[RSHANK].gain2;

    sensorData->bodyAccel[RTHIGH].lin_accel2    = (lin2[RTHIGH] + sensorData->bodyAccel[RTHIGH].offset2)
                                                * sensorData->bodyAccel[RTHIGH].gain2;

    sensorData->bodyAccel[UPPERBODY].lin_accel2 = (lin2[UPPERBODY] + sensorData->bodyAccel[UPPERBODY].offset2)
                                                * sensorData->bodyAccel[UPPERBODY].gain2;

    sensorData->bodyAccel[RFOOT].angular_accel     = (sensorData->bodyAccel[RFOOT].lin_accel2
                                                   - sensorData->bodyAccel[RFOOT].lin_accel1 )
                                                   * FOOT_ACC_DIST_INV;

    sensorData->bodyAccel[RSHANK].angular_accel       = (sensorData->bodyAccel[RSHANK].lin_accel2
                                                   - sensorData->bodyAccel[RSHANK].lin_accel1)
                                                   * SHANK_ACC_DIST_INV;

    sensorData->bodyAccel[RTHIGH].angular_accel       = (sensorData->bodyAccel[RTHIGH].lin_accel2
                                                   - sensorData->bodyAccel[RTHIGH].lin_accel1)
                                                   * THIGH_ACC_DIST_INV;

    sensorData->bodyAccel[UPPERBODY].angular_accel = 0; // add 120503

//    sensorData->bodyAccel[UPPERBODY].angular_accel = (sensorData->bodyAccel[UPPERBODY].lin_accel2
//                                                   - sensorData->bodyAccel[UPPERBODY].lin_accel1)
//                                                   * UPPERBODY_ACC_DIST_INV;

    sensorData->bodyAccel[LFOOT].angular_accel       = (sensorData->bodyAccel[LFOOT].lin_accel2
                                                   - sensorData->bodyAccel[LFOOT].lin_accel1)
                                                   * FOOT_ACC_DIST_INV;

    sensorData->bodyAccel[LSHANK].angular_accel    = (sensorData->bodyAccel[LSHANK].lin_accel2
                                                   - sensorData->bodyAccel[LSHANK].lin_accel1)
                                                   * SHANK_ACC_DIST_INV;

    sensorData->bodyAccel[LTHIGH].angular_accel       = (sensorData->bodyAccel[LTHIGH].lin_accel2
                                                   - sensorData->bodyAccel[LTHIGH].lin_accel1)
                                                   * THIGH_ACC_DIST_INV;

}

/* Function: GetJointAccelerations
 * ----------------------------------------------------------------------------------------------
 * Computes joint angular accelerations and updates the sensorData structure.
 * Accelerations are for the distal segment relative to the proximal segment
 */
void GetJointAccelerations(SensorDataT *sensorData){

    static double unfilteredAccel[8][4] =
{{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Current, A[j][0] and Previous,
A[j][1] values
    static double filteredAccel[8][4]   =
{{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in lowpass filter
    // all body and point linear accelerations, ang acc, ang vel, are wrt Frame 0 (inertial reference frame)

    sensorData->jointData[LTOE].acceleration   = sensorData->bodyAccel[LFOOT].angular_accel;    // Ltoe   (i.e. Lfoot wrt
ground)

    sensorData->jointData[LANKLE].acceleration = sensorData->bodyAccel[LSHANK].angular_accel
                                               - sensorData->bodyAccel[LFOOT].angular_accel;    // Lankle (i.e. Lshank wrt
Lfoot)
    sensorData->jointData[LKNEE].acceleration  = sensorData->bodyAccel[LTHIGH].angular_accel
```

```
                                        - sensorData->bodyAccel[LSHANK].angular_accel;   // Lknee  (i.e. Lthigh wrt
Lshank)
    sensorData->jointData[LHIP].acceleration   = sensorData->bodyAccel[UPPERBODY].angular_accel
                                               - sensorData->bodyAccel[LTHIGH].angular_accel;   // Lhip   (i.e. upper body
wrt Lthigh)
    sensorData->jointData[RTOE].acceleration   = sensorData->bodyAccel[RFOOT].angular_accel;     // Rtoe   (i.e. Rfoot wrt
ground)

    sensorData->jointData[RANKLE].acceleration = sensorData->bodyAccel[RSHANK].angular_accel
                                                 - sensorData->bodyAccel[RFOOT].angular_accel;    // Rankle (i.e. Rshank
wrt Rfoot)
    sensorData->jointData[RKNEE].acceleration  = sensorData->bodyAccel[RTHIGH].angular_accel
                                               - sensorData->bodyAccel[RSHANK].angular_accel;   // Rknee  (i.e. Rthigh wrt
Rshank)
     sensorData->jointData[RHIP].acceleration   = sensorData->bodyAccel[UPPERBODY].angular_accel
                                                  - sensorData->bodyAccel[RTHIGH].angular_accel;   // Rhip   (i.e. upper
body wrt Rthigh)

//    LowpassFilter(unfilteredAccel[LTOE],   filteredAccel[LTOE],   &sensorData->jointData[LTOE].acceleration ,
filterCoeffs1st2); // added by JRS, 2004-06-24, default 5Hz
//    LowpassFilter(unfilteredAccel[LANKLE], filteredAccel[LANKLE], &sensorData->jointData[LANKLE].acceleration,
filterCoeffs1st2); // added by JRS, 2004-06-24, default 5Hz
//    LowpassFilter(unfilteredAccel[LKNEE],  filteredAccel[LKNEE],  &sensorData->jointData[LKNEE].acceleration,
filterCoeffs1st2); // added by JRS, 2004-06-24, default 5Hz
//    LowpassFilter(unfilteredAccel[LHIP],   filteredAccel[LHIP],   &sensorData->jointData[LHIP].acceleration,
filterCoeffs1st2); // added by JRS, 2004-06-24, default 5Hz
//
//    LowpassFilter(unfilteredAccel[RTOE],   filteredAccel[RTOE],   &sensorData->jointData[RTOE].acceleration,
filterCoeffs1st2); // added by JRS, 2004-06-24, default 5Hz
//    LowpassFilter(unfilteredAccel[RANKLE], filteredAccel[RANKLE], &sensorData->jointData[RANKLE].acceleration,
filterCoeffs1st2); // added by JRS, 2004-06-24, default 5Hz
//    LowpassFilter(unfilteredAccel[RKNEE],  filteredAccel[RKNEE],  &sensorData->jointData[RKNEE].acceleration,
filterCoeffs1st2); // added by JRS, 2004-06-24, default 5Hz
//    LowpassFilter(unfilteredAccel[RHIP],   filteredAccel[RHIP],   &sensorData->jointData[RHIP].acceleration,
filterCoeffs1st2); // added by JRS, 2004-06-24, default 5Hz
}


/* Function: GetJointAccelerationsVelocityBased
 * ------------------------------------------------------------------------------------------
 * Computes joint angular accelerations and updates the sensorData structure.
 * Accelerations are based on differentiating velocity
 * Added by JRS, 2004-07-23
 */
void GetJointAccelerationsVelocityBased(SensorDataT *sensorData){

    static double unfilteredAcceleration[8][4] =
{{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Current, A[j][0] and Previous,
A[j][1] values
    static double filteredAcceleration[8][4]   =
{{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in lowpass filter

    static double unfilteredVelocity[8][4] =
{{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Current, A[j][0] and Previous,
A[j][1] values
    static double filteredVelocity[8][4]   =
{{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in lowpass filter

    LowpassFilter(unfilteredVelocity[LTOE],   filteredVelocity[LTOE],   &sensorData->jointData[LTOE].acceleration ,
filterCoeffsOFF); // added by JRS, 2004-06-24, default OFF, keep history of old velocities
    LowpassFilter(unfilteredVelocity[LANKLE], filteredVelocity[LANKLE], &sensorData->jointData[LANKLE].acceleration,
filterCoeffsOFF); // added by JRS, 2004-06-24, default OFF, keep history of old velocities
    LowpassFilter(unfilteredVelocity[LKNEE],  filteredVelocity[LKNEE],  &sensorData->jointData[LKNEE].acceleration,
filterCoeffsOFF); // added by JRS, 2004-06-24, default OFF, keep history of old velocities
    LowpassFilter(unfilteredVelocity[LHIP],   filteredVelocity[LHIP],   &sensorData->jointData[LHIP].acceleration,
filterCoeffsOFF); // added by JRS, 2004-06-24, default OFF, keep history of old velocities

    LowpassFilter(unfilteredVelocity[RTOE],   filteredVelocity[RTOE],   &sensorData->jointData[RTOE].acceleration,
filterCoeffsOFF); // added by JRS, 2004-06-24, default OFF, keep history of old velocities
    LowpassFilter(unfilteredVelocity[RANKLE], filteredVelocity[RANKLE], &sensorData->jointData[RANKLE].acceleration,
filterCoeffsOFF); // added by JRS, 2004-06-24, default OFF, keep history of old velocities
    LowpassFilter(unfilteredVelocity[RKNEE],  filteredVelocity[RKNEE],  &sensorData->jointData[RKNEE].acceleration,
filterCoeffsOFF); // added by JRS, 2004-06-24, default OFF, keep history of old velocities
    LowpassFilter(unfilteredVelocity[RHIP],   filteredVelocity[RHIP],   &sensorData->jointData[RHIP].acceleration,
filterCoeffsOFF); // added by JRS, 2004-06-24, default OFF, keep history of old velocities

    // all body and point linear accelerations, ang acc, ang vel, are wrt Frame 0 (inertial reference frame)

    sensorData->jointData[LTOE].acceleration   = (filteredVelocity[LTOE][0]   - filteredVelocity[LTOE][1])   * FREQ; //
Ltoe   (i.e. Lfoot wrt ground)

    sensorData->jointData[LANKLE].acceleration = (filteredVelocity[LANKLE][0] - filteredVelocity[LANKLE][1]) * FREQ; //
Lankle (i.e. Lshank wrt Lfoot)

    sensorData->jointData[LKNEE].acceleration  = (filteredVelocity[LKNEE][0]  - filteredVelocity[LKNEE][1])  * FREQ; //
Lknee  (i.e. Lthigh wrt Lshank)

    sensorData->jointData[LHIP].acceleration   = (filteredVelocity[LHIP][0]   - filteredVelocity[LHIP][1])   * FREQ; //
Lhip   (i.e. upper body wrt Lthigh)

    sensorData->jointData[RTOE].acceleration   = (filteredVelocity[RTOE][0]   - filteredVelocity[RTOE][1])   * FREQ; //
Rtoe   (i.e. Rfoot wrt ground)

    sensorData->jointData[RANKLE].acceleration = (filteredVelocity[RANKLE][0] - filteredVelocity[RANKLE][1]) * FREQ; //
Rankle (i.e. Rshank wrt Rfoot)

    sensorData->jointData[RKNEE].acceleration  = (filteredVelocity[RKNEE][0]  - filteredVelocity[RKNEE][1])  * FREQ; //
Rknee  (i.e. Rthigh wrt Rshank)
```

```
     sensorData->jointData[RHIP].acceleration   = (filteredVelocity[LTOE][0]   - filteredVelocity[LTOE][1])   * FREQ; //
Rhip   (i.e. upper body wrt Rthigh)

    LowpassFilter(unfilteredAcceleration[LTOE],   filteredAcceleration[LTOE],   &sensorData->jointData[LTOE].acceleration ,
filterCoeffs1st5); // added by JRS, 2004-06-24, default 5Hz
    LowpassFilter(unfilteredAcceleration[LANKLE], filteredAcceleration[LANKLE], &sensorData-
>jointData[LANKLE].acceleration, filterCoeffs1st5); // added by JRS, 2004-06-24, default 5Hz
    LowpassFilter(unfilteredAcceleration[LKNEE],  filteredAcceleration[LKNEE],  &sensorData->jointData[LKNEE].acceleration,
filterCoeffs1st5); // added by JRS, 2004-06-24, default 5Hz
    LowpassFilter(unfilteredAcceleration[LHIP],   filteredAcceleration[LHIP],   &sensorData->jointData[LHIP].acceleration,
filterCoeffs1st5); // added by JRS, 2004-06-24, default 5Hz

    LowpassFilter(unfilteredAcceleration[RTOE],   filteredAcceleration[RTOE],   &sensorData->jointData[RTOE].acceleration,
filterCoeffs1st5); // added by JRS, 2004-06-24, default 5Hz
    LowpassFilter(unfilteredAcceleration[RANKLE], filteredAcceleration[RANKLE], &sensorData-
>jointData[RANKLE].acceleration, filterCoeffs1st5); // added by JRS, 2004-06-24, default 5Hz
    LowpassFilter(unfilteredAcceleration[RKNEE],  filteredAcceleration[RKNEE],  &sensorData->jointData[RKNEE].acceleration,
filterCoeffs1st5); // added by JRS, 2004-06-24, default 5Hz
    LowpassFilter(unfilteredAcceleration[RHIP],   filteredAcceleration[RHIP],   &sensorData->jointData[RHIP].acceleration,
filterCoeffs1st5); // added by JRS, 2004-06-24, default 5Hz
}
```

Appendix A.8 – JointCtl.h

```
/* Function: JointController
 * --------------------------
 * control the valves using a desired torque
 */
void JointController(const double          desiredTorque,
                     const double          desiredTorque_dot,
                     const int             useDesiredTorque_dot,
                     SensorDataT           *sensorData,
                     const SysPropertiesT  *sysProperties,
                     const int             valveNumber);


/* Function: ComputeHydraulicParameters
 * -----------------------------
 * returns some model parameters for valve nonlinear controllers
 *
 *  hydraulicParameters = [ xv h2 o2 p2]
 *
 */
void ComputeHydraulicParameters(SensorDataT          *sensorData,
                                const int            valveNumber,
                                const int            valveNumberInJointData,
                                double               *hydraulicParameters,
                                const SysPropertiesT *sysProperties);


/* Function: SimpleMSS
 * ---------------------------------------------------------------------------------------------
 *  Computes and sets the required valve voltage input using an Adaptive Multiple Sliding Surface control law.
 *  valveNumber = [LTOE LANKLE LKNEE LHIP RTOE RANKLE RKNEE RHIP]
 *  hydraulicParameters = [xv h2 o2 p2]
 *  Added by JRS, 06-24-2004
 */
double SimpleMSS(const double          desiredForce,
                 const double          desiredForce_dot,
                 const int             useDesiredForce_dot,
                 SensorDataT           *sensorData,
                 const int             valveNumber,
                 const int             valveNumberInJointData,
                 double                *hydraulicParameters,
                 const SysPropertiesT *sysProperties);


/* Function: Dither
 * -------------------------
 * Adds dither to the valve voltage. According to Moog, dither peak to peak amplitude should be less than 10%  max voltage
(5V)
 * and dither frequency should be 1.5 times the natural frequency of the valve. for the Series 31 valve: wn = 150Hz @ 1000
psi,
 * 120Hz @ 500psi.
 * Dither is a square wave (according to Meritt signal shape makes no difference)
 */
void Dither(double                *voltage,
            const int             valveNumber,
            const SysPropertiesT *sysProperties);
```

Appendix A.9 – JointCtl.c

```
#include <math.h>
#include <stdio.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"
#include "JointCtl.h"
#include "DSup.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];

extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];

/* Function: JointController
 * -------------------------
 * control the valves using a desired torque
 */
void JointController(const double          desiredTorque,
                     const double          desiredTorque_dot,
                     const int             useDesiredTorque_dot,
                     SensorDataT           *sensorData,
                     const SysPropertiesT  *sysProperties,
                     const int             valveNumber){

    double voltage               = DEFAULT_VALVE_IN; // default is +/- 4.5 volts
    double desiredForce;
    double desiredForce_dot;
    double hydraulicParameters[4] = {0,0,0,0};
    int    useDesiredForce_dot    = useDesiredTorque_dot;
    int    valveNumberInJointData;

    switch(valveNumber){ // change array index number so it matches convention in jointData array
    case LANKLE_T:
        valveNumberInJointData = LANKLE;
        break;
    case LKNEE_T:
        valveNumberInJointData = LKNEE;
        break;
    case LHIP_T:
        valveNumberInJointData = LHIP;
        break;
    case RANKLE_T:
        valveNumberInJointData = RANKLE;
        break;
    case RKNEE_T:
        valveNumberInJointData = RKNEE;
        break;
    case RHIP_T:
        valveNumberInJointData = RHIP;
        break;
    }

    ComputeHydraulicParameters(sensorData, valveNumber, valveNumberInJointData, hydraulicParameters, sysProperties);

    desiredForce     = (-1) * desiredTorque     / sensorData->jointData[valveNumberInJointData].momentArm;
    desiredForce_dot = (-1) * desiredTorque_dot / sensorData->jointData[valveNumberInJointData].momentArm;

    voltage = SimpleMSS(desiredForce,
                        desiredForce_dot,
                        useDesiredForce_dot,
                        sensorData,
                        valveNumber,
                        valveNumberInJointData,
                        hydraulicParameters,
                        sysProperties); // added by JRS, 2004-06-25

    // Copy required voltages to sensorData structure which is accessible from the PCI DAC function
    sensorData->jointData[valveNumberInJointData].valveVoltage = voltage; // valve input voltage (V)
}


/* Function: ComputeHydraulicParameters
 * ------------------------------
 * returns some model parameters for valve nonlinear controllers
 *
 *  hydraulicParameters = [ xv h2 o2 p2]
 *
 */
```

```
void ComputeHydraulicParameters(SensorDataT          *sensorData,
                                const int            valveNumber,
                                const int            valveNumberInJointData,
                                double               *hydraulicParameters,
                                const SysPropertiesT *sysProperties){

    double FL, fx, xv, h2, h2_hat, o2, p2, i, denom_inv, xp, xp_dot, V0, FL_tilda, p2_hat, f4, Ps;
    double alpha1 = 3000; // rknee 3000                          tip: tune alpha1, k1 for s.s. first
    double alpha2 = 0;    // rknee 0.0005; rhip 0.0
    double k1     = 500;  // rknee 500
    double k2     = 0;    // rknee 0.0001; rhip 0.0
    static double FL_hat[6]     = {0,0,0,0,0,0};
    static double FL_hat_dot[6] = {0,0,0,0,0,0};
    static double xv_hat[6]     = {0,0,0,0,0,0};
    static double xv_hat_dot[6] = {0,0,0,0,0,0};
    int sign = 1;

    if(valveNumberInJointData == RKNEE || valveNumberInJointData == LKNEE ){ // adjust sign for knee valve
        sign = -1;
    }

    FL     = -sensorData->jointData[valveNumberInJointData].sensorForce;
    xp     =  sensorData->jointData[valveNumberInJointData].pistonPosition;
    xp_dot =  sensorData->jointData[valveNumberInJointData].pistonVelocity;
    i      =  sensorData->jointData[valveNumberInJointData].valveVoltage / R_VALVE; // valve input current (A)

    Ps = PS; // get the supply pressure from the Defines.h file

    // get V0 volume according to joint
    if (valveNumberInJointData == LANKLE || valveNumberInJointData == RANKLE){
        V0 = V0_ANKLE;
    }
    else if (valveNumberInJointData == LKNEE || valveNumberInJointData == RKNEE){
        V0 = V0_KNEE;
    }
    else{
        V0 = V0_HIP;
    }

    // for spool observer
    //FL_hat[valveNumber] = FL_hat[valveNumber] + FL_hat_dot[valveNumber]*TS; // observer load force

    xv_hat[valveNumber] = xv_hat[valveNumber] + xv_hat_dot[valveNumber]*TS; // observer spool position

    if(xv_hat[valveNumber] > XVMAX){ // saturate spool position
        xv_hat[valveNumber] = XVMAX;
    }
    else if(xv_hat[valveNumber] < -XVMAX){
        xv_hat[valveNumber] = -XVMAX;
    }

    xv = xv_hat[valveNumber]; // use observer value of valve spool position (m)

    //-----------------------------------------------
    // this approximation of the spool position can be used instead of the valve observer
    //    xv = sign * i*KS; // spool position ignoring valve dynamics (m)
    //-----------------------------------------------

    if (xv > 0){
        fx = Ap1; // piston area (m2)
    }
    else if (xv < 0){
        fx = Ap2;
    }
    else{
        fx = 0.5*(Ap1 + Ap2);
    }

    denom_inv = 1 /(xp*(Ap1-Ap2)+2*V0);

    if (xv==0){
        h2 = 2*sqrt(2*(Ap1+Ap2)*(fx*Ps))*denom_inv; // flow pressure differential factor
    }else if( (xv/fabs(xv))*FL < fx*Ps){
        h2 = 2*sqrt(2*(Ap1+Ap2)*(fx*Ps-(xv/fabs(xv))*FL))*denom_inv; // flow pressure differential factor
    }else{
        h2 = 0;
        //h2 = 2*sqrt(2*(Ap1+Ap2)*(fx*Ps))*denom_inv; // (was h2 = 0 changed on 2003-04-13); flow is saturated
    }

    o2 = (Ap1+Ap2)*(Ap1+Ap2) * xp_dot * denom_inv; // flow compressibility factor

    p2 = (2*FL + Ps*(Ap2-Ap1))*denom_inv; // cylinder leakage factor based on measured force

    hydraulicParameters[XV] = xv; // save computed values to array accessible from calling function
    hydraulicParameters[H2] = h2;
    hydraulicParameters[O2] = o2;
    hydraulicParameters[P2] = p2;

      // use open loop valve spool model without observer gains
    xv_hat_dot[valveNumber] = sign*G3*i + F3*xv_hat[valveNumber]; // spool velocity

}


/* Function: SimpleMSS
 * ------------------------------------------------------------------------------------------
 *  Computes and sets the required valve voltage input using an Adaptive Multiple Sliding Surface control law.
 *  valveNumber = [LTOE LANKLE LKNEE LHIP RTOE RANKLE RKNEE RHIP]
 *  hydraulicParameters = [xv h2 o2 p2]
```

```
 *  Added by JRS, 06-24-2004
 */
double SimpleMSS(const double         desiredForce,
                 const double         desiredForce_dot,
                 const int            useDesiredForce_dot,
                 SensorDataT          *sensorData,
                 const int            valveNumber,
                 const int            valveNumberInJointData,
                 double               *hydraulicParameters,
                 const SysPropertiesT *sysProperties){

    double xvd;
    double xv;
    double h2;
    double o2;
    double p2;
    double FL;
    double u=0;
    double lambda1,lambda2;
    double Fd;
    double Fd_dot;
    double eta2;
    double phi_inv;
    double K;
    double Fe;
    double gamma_hat;
    double beta_hat;
    double si_hat;
    double ro1;
    double ro2_inv;
    double s1, s2;
    double xvd_dot;
    double i;
    double Ci;
    int    sign = 1;
    int    j;

    static double xvd_prev[6]    = {0,0,0,0,0,0}; // previous value of valve spool position
    static double Fd_prev[6]     = {0,0,0,0,0,0}; // previous value of desired force Fd
    static double Fe_integral[6] = {0,0,0,0,0,0}; // integral of force tracking error (Fe = Fd - FL) over time
    static double unfilteredFe_integral[6][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used to
smooth out sharp changes in fe_integral, seems to eliminate hip jerking
    static double filteredFe_integral[6][4]   = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; //
    static double unfilteredFd_dot[6][4]      = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Current
A[0] and Previous values
    static double filteredFd_dot[6][4]        = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter
    static double unfilteredxvd_dot[6][4]     = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Current
A[0] and Previous values
    static double filteredxvd_dot[6][4]       = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter
    static double unfilteredLambda1[6][4]     = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Current
A[0] and Previous values
    static double filteredLambda1[6][4]       = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter

    if(valveNumber == RKNEE_T || valveNumber == LKNEE_T){  // adjust sign of voltage output
        sign = -1;
    }

    xv =  hydraulicParameters[XV]; // get computed hydraulic model parameters
    h2 =  hydraulicParameters[H2];
    o2 =  hydraulicParameters[O2];
    p2 =  hydraulicParameters[P2];
    FL = -sensorData->jointData[valveNumberInJointData].sensorForce; //where is the vibration coming from when Fhm is
turned on? Fdesired is now smooth.
    Fd = -desiredForce;

    // get default or GUI control gains
       lambda1 = sysProperties->jointControl[valveNumber].lambda1*10;   // gain (700 in sim)
    lambda2 = sysProperties->jointControl[valveNumber].lambda2;      // gain (700 in sim)
    eta2    = sysProperties->jointControl[valveNumber].eta2*1e-6;    // robustness term for spool dyn.(1e-9 in sim)
    phi_inv = sysProperties->jointControl[valveNumber].phi2_inv*10;  // boundary layer for s2    (1e3 in sim)
    ro1     = sysProperties->jointControl[valveNumber].ro1*1e-6;     // (1e-8 in sim)
    ro2_inv = sysProperties->jointControl[valveNumber].ro2_inv*1e-6; // (1e-9 in sim)
    Ci      = sysProperties->jointControl[valveNumber].Ci;           // integral gain for s1 (100 in sim)

    Ci = Ci*sysProperties->jointControl[valveNumber].CiSwitch; // switch to turn Ci on/off from fhm.c, added by JRS, 2004-
07-19
    sensorData->torsoForce.SGt[valveNumber] = Ci; // output Ci to torso force sensor SGt channels on GUI

    // Get optimal gains for each state when in Auto torque Mode
    if (sysProperties->debuggingControls.test4){
          lambda1 = sysProperties->jointControl[valveNumber].lambda1*10; // gain (700 in sim)
        lambda2 = sysProperties->jointControl[valveNumber].lambda2;     // gain (700 in sim)
    }else{
        if(sysProperties->mainOperationMode == AUTO_TORQUE_CTL){

            if((valveNumber == LANKLE_T) || (valveNumber == RANKLE_T)){
                lambda1 = 2000;
                lambda2 = 400;
            }else if((valveNumber == LKNEE_T) || (valveNumber == RKNEE_T)){
                lambda1 = 500;
                lambda2 = 400;
            }else if((valveNumber == LHIP_T) || (valveNumber == RHIP_T)){
                lambda1 = 1000;
                lambda2 = 1000;
            }
        }
```

```
    }


//    // Get optimal gains for each state when in Auto torque Mode
//    if(sysProperties->mainOperationMode == AUTO_TORQUE_CTL){ // 3 = auto torque mode
//        // this code will reset the ci gain only for the 1red case and only for the left leg when it is redundant
//        //if (sensorData->dynamicMode.Mode == ONE_REDUNDANCY && sensorData->dynamicMode.leftIsRedundant && valveNumber==
LKNEE_T ){
//        //        Ci = sysProperties->ditherAmplitude; // CAUTION: ditherAmplitude used for torso mass now
//        //}
//
//        // left leg
//        if( (sensorData->dynamicMode.Mode == SSTANCE && !sensorData->dynamicMode.leftIsGrounded)
//            || sensorData->dynamicMode.Mode == JUMP){                                                    // left is in swing
//            if(valveNumber == LANKLE_T){
//                lambda1 = 2000;
//                if(sensorData->jointData[valveNumberInJointData+1].position > 1.4){    // decrease ankle gain when knee
is close to fully flexed
//                    lambda1 = 0.8*lambda1;
//                }else if (sensorData->jointData[valveNumberInJointData].position > 0.53){     // if ankle is against the
back stop decrease its gain linearly
//                    lambda1 = 0.775*lambda1;
//                }
//            }else if(valveNumber == LKNEE_T){
//                lambda1 = 500;
//            }else if(valveNumber == LHIP_T){
//                lambda1 = 1000;
//            }
//        }
//        else if (sensorData->dynamicMode.Mode == ONE_REDUNDANCY && sensorData->dynamicMode.leftIsRedundant && valveNumber
== LKNEE_T){
//            lambda1=500;
//        }
//        else{    // left is in stance
//            if(valveNumber == LANKLE_T){
//                lambda1 = 100;
//            }else if(valveNumber == LKNEE_T){
//                lambda1 = 50;
//            }else if(valveNumber == LHIP_T){
//                lambda1 = 200;
//            }
//        }
//
//        // right leg
//        if( (sensorData->dynamicMode.Mode == SSTANCE && sensorData->dynamicMode.leftIsGrounded)
//            || sensorData->dynamicMode.Mode == JUMP){                                                    // right is in swing
//            if(valveNumber == RANKLE_T){
//                lambda1 = 2000;     //7000
//                if(sensorData->jointData[valveNumberInJointData+1].position > 1.4){    // decrease ankle gain when knee
is close to fully flexed
//                    lambda1 = 0.8*lambda1;
//                }else if (sensorData->jointData[valveNumberInJointData].position > 0.53){ // if ankle is against the back
stop decrease its gain linearly
//                    lambda1 = 0.775*lambda1;
//                }
//            }else if(valveNumber == RKNEE_T){
//                lambda1 = 500;
//            }else if(valveNumber == RHIP_T){
//                lambda1 = 1000;
//            }
//        }
//        else if (sensorData->dynamicMode.Mode == ONE_REDUNDANCY && !sensorData->dynamicMode.leftIsRedundant &&
valveNumber == RKNEE_T){
//            lambda1=500;
//        }
//        else{    // right is in stance
//            if(valveNumber == RANKLE_T){
//                lambda1 = 100;
//            }else if(valveNumber == RKNEE_T){
//                lambda1 = 50;
//            }else if(valveNumber == RHIP_T){
//                lambda1 = 200;
//            }
//        }
//
//    // Filter lambda1 (when needed) to prevent sudden gain jumps and jerky response
//        if(lambda1 > filteredLambda1[valveNumber][0]){  // if the current lambda1 is larger than the previous, filter it
(1Hz)
//
//            // Filters for lambda1 during SSTANCE
//            if( (sensorData->dynamicMode.Mode == SSTANCE) &&
//                (  (valveNumber == LANKLE_T && !sensorData->dynamicMode.leftIsGrounded)
//                || (valveNumber == RANKLE_T && sensorData->dynamicMode.leftIsGrounded)  )){ // swing leg, ankle
//
//                LowpassFilter(unfilteredLambda1[valveNumber], filteredLambda1[valveNumber], &lambda1,
filterCoeffs2nd025);//ankle, 0.25Hz default
//            }
//            else if( (sensorData->dynamicMode.Mode == SSTANCE) &&
//                (  (valveNumber == LKNEE_T && !sensorData->dynamicMode.leftIsGrounded)
//                || (valveNumber == RKNEE_T && sensorData->dynamicMode.leftIsGrounded)  )){ // swing leg, knee
//
//                LowpassFilter(unfilteredLambda1[valveNumber], filteredLambda1[valveNumber], &lambda1, filterCoeffs2nd10);
//knee, 1Hz default (sticky), 10Hz is not sticky but causes knee jerk
//            }
//            else if( (sensorData->dynamicMode.Mode == SSTANCE) &&
//                (  (valveNumber == LHIP_T && !sensorData->dynamicMode.leftIsGrounded)
//                || (valveNumber == RHIP_T && sensorData->dynamicMode.leftIsGrounded)  )){ // swing leg, hip
//
```

```
//              LowpassFilter(unfilteredLambda1[valveNumber], filteredLambda1[valveNumber], &lambda1, filterCoeffs2nd10);
//hip, 1Hz default (sticky), 10Hz is not sticky but causes knee jerk
//          }
//
//          // Filters for lambda1 during ONE_REDUNDANCY
//          else if( (sensorData->dynamicMode.Mode == ONE_REDUNDANCY) &&
//              (  (valveNumber == LANKLE_T && sensorData->dynamicMode.leftIsRedundant)
//              || (valveNumber == RANKLE_T && !sensorData->dynamicMode.leftIsRedundant)  )){ // redundant leg, ankle
//
//              LowpassFilter(unfilteredLambda1[valveNumber], filteredLambda1[valveNumber], &lambda1,
filterCoeffs2nd10);//ankle, 10Hz default
//          }
//          else if( (sensorData->dynamicMode.Mode == ONE_REDUNDANCY) &&
//              (  (valveNumber == LKNEE_T && sensorData->dynamicMode.leftIsRedundant)
//              || (valveNumber == RKNEE_T && !sensorData->dynamicMode.leftIsRedundant)  )){ //  redundant  leg, knee
//
//              LowpassFilter(unfilteredLambda1[valveNumber], filteredLambda1[valveNumber], &lambda1, filterCoeffs2nd10);
//knee, 10Hz default
//          }
//          else if( (sensorData->dynamicMode.Mode == ONE_REDUNDANCY) &&
//              (  (valveNumber == LHIP_T && sensorData->dynamicMode.leftIsRedundant)
//              || (valveNumber == RHIP_T && !sensorData->dynamicMode.leftIsRedundant)  )){ // redundant leg, hip
//
//              LowpassFilter(unfilteredLambda1[valveNumber], filteredLambda1[valveNumber], &lambda1, filterCoeffs2nd10);
//hip, 10Hz default
//          }
//
//          else{
//              LowpassFilter(unfilteredLambda1[valveNumber], filteredLambda1[valveNumber], &lambda1, filterCoeffs2nd10);
//every other state, 1Hz default, test 10Hz=choppy (best out of 1,10,5), and 5Hz=sluggish and hard to lift feet
//          }
//
//      }
//      else{
//          for(j=0;j<4;j++){ // reinitialize filter
//              unfilteredLambda1[valveNumber][j] = lambda1;
//              filteredLambda1[valveNumber][j] = lambda1;
//          }
//      }
//  } // end of "if(sysProperties->mainOperationMode == AUTO_TORQUE_CTL){...."

    o2 = 0; // tracking works best with this value

    if (useDesiredForce_dot){ // if the desired force derivative has previously been computed use that value
        Fd_dot = desiredForce_dot;
    }
    else{
        Fd_dot = (Fd - Fd_prev[valveNumber])*FREQ; // compute Fd derivative
    }

    Fd_prev[valveNumber] = Fd; // save current desired force to previous value of next iteration

    Fe = FL - Fd; // compute force error

    if ((fabs(Ci) < 0.001)

        || (Ci == 0) // if Ci == 0

        || (sensorData->dynamicMode.LstanceSwingTransition || sensorData->dynamicMode.LstateTransition) // the following
two lines override the above two cases
        || (sensorData->dynamicMode.RstanceSwingTransition || sensorData->dynamicMode.RstateTransition)

        || (sensorData->jointData[valveNumberInJointData].againstStop) // added by JRS, 2004-07-20

        ){
        Fe_integral[valveNumber] = 0; // reset integral term
    }
    else {
        u = sensorData->jointData[valveNumberInJointData].valveVoltage;

        // integrate Fe if voltage and force are not saturated ( max force is 1800 @1000psi )
        if( (u<LMAX && u>-LMAX) && (Fd < FMAXSOFT && Fd > -FMAXSOFT) ){ // changed from || to &&, JRS, 2004-10-27
            Fe_integral[valveNumber] = Fe_integral[valveNumber] + Fe*TS; // compute force tracking error integral
        }
    }

//    // filtering Fe_integral is just a guess as to what might be causing the "jerking" in the knee
//    // --testing confirms that increasing filtering reduces jerking in knee/hip
//    if (sensorData->dynamicMode.Mode == ONE_REDUNDANCY
//        && ((!sensorData->dynamicMode.leftIsRedundant && valveNumber == RKNEE_T)
//            || (sensorData->dynamicMode.leftIsRedundant && valveNumber == LKNEE_T ))){
//
//        LowpassFilter(unfilteredFe_integral[valveNumber], filteredFe_integral[valveNumber], &Fe_integral[valveNumber],
filterCoeffs2nd50);
//    }

    s1 = Fe + Ci*Fe_integral[valveNumber]; // 1st sliding surface

    gamma_hat = GAMMA;
    beta_hat  = BETA;
    si_hat    = SI;

    // desired spool position (m)
    xvd = -(1/(h2*gamma_hat))*(lambda1*s1 - beta_hat*o2 - si_hat*p2 - Fd_dot + Ci*Fe);

    //saturate desired spool position to what is physically possible
    if(xvd > XVMAX){
        xvd = XVMAX;
    }
```

```
    else if(xvd < -XVMAX){
        xvd = -XVMAX;
    }

    s2 = xv - xvd; // 2nd sliding surface (spool position error)

     xvd_dot = (xvd - xvd_prev[valveNumber])*FREQ ; // compute desired valve spool position (xvd) derivative

    xvd_prev[valveNumber] = xvd; // save desired valve position for next iteration

    K = (DELTA_F3_MAX*fabs(xv)+(G3_TILDA_MAX - 1)*fabs(xvd_dot - F3*xv) + eta2*ro2_inv )*G3_TILDA_MAX_INV; // compute gain
K

    i = sign * G3_INV*(xvd_dot - F3*xv -lambda2*s2 - ro1*ro2_inv*gamma_hat*h2*s1 - K*s2*phi_inv ); // compute current

    u = R_VALVE * i; //    u = (Rseries + Rcoil) x i

    return u;
}


/* Function: Dither
 * ----------------------------
 * Adds dither to the valve voltage. According to Moog, dither peak to peak amplitude should be less than 10%  max voltage
(5V)
 * and dither frequency should be 1.5 times the natural frequency of the valve. for the Series 31 valve: wn = 150Hz @ 1000
psi,
 * 120Hz @ 500psi.
 * Dither is a square wave (according to Meritt signal shape makes no difference)
 */
void Dither(double               *voltage,
            const int            valveNumber,
            const SysPropertiesT *sysProperties){

    double     P    = 2.22e-3; // 1/2 dither period (sec)
    double     A    = 0.2; // dither amplitude (V)
    static int k    = 0; // iteration counter
    int        kmax = 0; // max number of iteration for 1/2 period
    static int sign = 1; // direction of dither signal

    if(sysProperties->ditherFrequency != 0){ // don't add dither to the voltage if no frequency is specified
        P = 0.5/(sysProperties->ditherFrequency); // 1/2 dither period
        A = sysProperties->ditherAmplitude;

        kmax = (int) (P/TS); // divide 1/2 period by sampling time to find number of iterations needed for one 1/2 period

        if(valveNumber == 0){ // this function runs 6 times so don't keep on switching the sign 6 times
            k++; // update the counter
            if (k>kmax){ // if the counter has gone beyond the 1/2 period, change the sign and zero the counter
                k    =  0;
                sign = -sign;
            }
        }

        *voltage = *voltage + A * sign;
    }
}
```

Appendix A.10 – Fhm.h

```
/* Function: GetFHMcontrolTorques
 * ----------------------------
 *  Computes the required machine torques to minimize human-machine forces.
 */
void GetFHMcontrolTorques(double           *FHMTorques,
                          const BodyDataT *bodyData,
                          SensorDataT     *sensorData,
                          SysPropertiesT  *sysProperties);

/* Function: GetTgAndTHM
 * --------------------------------
 * Calculates and updates the required gravity compensator torque vector.
 * Calculates and ouputs the joint torques due to human on the machine.
 * The vectors are as follows:- [TankleL TkneeL ThipL TankleR TkneeR ThipR]
 */
void GetTgAndTHM(const BodyDataT    *bodyData,
                 SensorDataT        *sensorData,
                 SysPropertiesT     *sysProperties,
                 ForceDistributionT *distrData);
```

Appendix A.11 – Fhm.c

```
#include <math.h>
#include <time.h>
#include <stdio.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"
#include "Fhm.h"
#include "SSup.h"
#include "DSup.h"
#include "Jump.h"
#include "1Red.h"
#include "2Red.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];
extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];


/* Function: GetFHMcontrolTorques
* ------------------------------------------------------------------------------------------------
* Computes the required machine command torques FHMTorques to minimize human-machine forces.
* Command Torque can then be sent to valve controller.
*/
void GetFHMcontrolTorques(double           *FHMTorques,
                          const BodyDataT   *bodyData,
                          SensorDataT       *sensorData,
                          SysPropertiesT    *sysProperties){

    int    j, i;
    double Tlin[6]          = {0,0,0,0,0,0}; // torque for feedback linearization Tlin = Tg + Tf
    double dTlin[6]         = {0,0,0,0,0,0}; // human-machine torque time derivative
    double THM[6]           = {0,0,0,0,0,0}; // human-machine torque
    double dTHM[6]          = {0,0,0,0,0,0}; // human-machine torque time derivative
    static double THM_int[6] = {0,0,0,0,0,0}; // integral of THM
    double Tfriction[6]     = {0,0,0,0,0,0}; // debug, added by JRS 2004-07-20
    static double unfilteredTHM[6][4]    = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Previous
values
    static double filteredTHM[6][4]      = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter
    static double unfiltered_dTHM[6][4]  = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Previous
values
    static double filtered_dTHM[6][4]    = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter
    static double unfilteredTlin[6][4]   = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Previous
values
    static double filteredTlin[6][4]     = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter
    static double unfiltered_dTlin[6][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Previous
values
    static double filtered_dTlin[6][4]   = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter
    double VLimitsTorques[6]   = {0,0,0,0,0,0}; // torques to implement a virtual stop
    double kpFHM               = 0; // PD gains
    double kv                  = 0;
    double kp                  = 0;
    double THMmax              = 10;
    double THMmin              = 0.5; // max and min values of THM;
    double THMReductionFactor  = 0; // JRS default is ZERO!
    int    frequency           = 2;
    double accel[6]            = {0,0,0,0,0,0};
    double *DfilterCoeffsTemp;
    double SWING_FhmGains[6]   = {SWING_LANKLE_KPFHM ,   SWING_LKNEE_KPFHM ,   SWING_LHIP_KPFHM ,   SWING_RANKLE_KPFHM ,
SWING_RKNEE_KPFHM ,   SWING_RHIP_KPFHM};
    double DSTANCE_FhmGains[6] = {DSTANCE_LANKLE_KPFHM , DSTANCE_LKNEE_KPFHM , DSTANCE_LHIP_KPFHM , DSTANCE_RANKLE_KPFHM ,
DSTANCE_RKNEE_KPFHM , DSTANCE_RHIP_KPFHM};

    GetTgAndTHM(bodyData, sensorData, sysProperties, &sensorData->forceDistribution); // update THM, torque due to human on
machine

    for (j=0; j<6; j++){ // j is the valve number

        switch(j){   // change array index number so it matches convention in jointData array
        case LANKLE_T:   // index j is used in 6x1 arrays
            i = LANKLE;  // index i is used in 8x1 arrays
            break;
        case LKNEE_T:
            i = LKNEE;
            break;
```

```
        case LHIP_T:
            i = LHIP;
            break;
        case RANKLE_T:
            i = RANKLE;
            break;
        case RKNEE_T:
            i = RKNEE;
            break;
        case RHIP_T:
            i = RHIP;
            break;
        }

        if(sysProperties->torqueControl.controlFHM){ // if FHM control is enabled
            if (sysProperties->jointControl[j].kpFHM <= 0){ //if kpFHM is NEGATIVE or ZERO in GUI, use pre-set gains found
in Defines.h
                kpFHM = SWING_FhmGains[j];
                kv    = 0;
            }
            else{ // else use GUI gains
                kpFHM = sysProperties->jointControl[j].kpFHM;
                kv    = sysProperties->jointControl[j].kv;
            }
        }else{ // else, Fhm switch is off, set gains to 0
            kpFHM = 0;
            kv    = 0;
        }


//        if(sysProperties->torqueControl.controlFHM){ // if FHM control is enabled
//            if( sensorData->dynamicMode.Mode == SSTANCE &&
//                ( ( j < RANKLE_T && !sensorData->dynamicMode.leftIsGrounded)// if this is a swing leg
//                || ( j > LHIP_T  && sensorData->dynamicMode.leftIsGrounded) ) ){
//                //kv = sysProperties->jointControl[j].kv;
//                kpFHM = SWING_FhmGains[j];
//            }else if (sensorData->dynamicMode.Mode == DSTANCE){
//                kpFHM = DSTANCE_FhmGains[j];
//                //kv = sysProperties->jointControl[j].kv;
//            }else{
//                kpFHM = sysProperties->jointControl[j].kpFHM; // get latest feedback gains
//                if(kpFHM != 0){
//                    kv = sysProperties->jointControl[j].kv;      // make sure I gain is not activated when P gain is 0
//                }else{
//                    kv = 0;
//                }
//            }
//        }else{            // if FHM control set gains to 0
//            kpFHM = 0;
//            kv = 0;
//        }


        // ignore FHM if it's forcing against the stops
        if(i == RANKLE || i == LANKLE){
            if(sensorData->jointData[i].position < (ANKLE_MIN + ANKLE_MIN_SOFT_OFFSET) /* && sensorData->jointData[i].Thm
< 0 */){ // old ankle soft min = 0.17

                sensorData->jointData[i].Thm            = 0; //sensorData->jointData[i].Thm*THMReductionFactor;
                sensorData->jointData[i].againstStop    = TRUE; // when against stops only compensate for gravity
                //sysProperties->jointControl[j].CiSwitch = 0; // Ci=0 is OFF

            }else if(sensorData->jointData[i].position > (ANKLE_MAX - ANKLE_MAX_SOFT_OFFSET) /*&& sensorData-
>jointData[i].Thm  > 0*/){ // old ankle soft max = 0.23

                sensorData->jointData[i].Thm            = 0; //sensorData->jointData[i].Thm*THMReductionFactor;
                //sensorData->jointData[i].againstStop    = TRUE;
                //sysProperties->jointControl[j].CiSwitch = 0; // Ci=0 is OFF

            }else{

                sensorData->jointData[i].againstStop    = FALSE;
                sysProperties->jointControl[j].CiSwitch    = 1; // Ci=1 is ON

            }
        }else if(i == RKNEE || i == LKNEE){
            if(sensorData->jointData[i].position < (KNEE_MIN + KNEE_MIN_SOFT_OFFSET)  /*&& sensorData->jointData[i].Thm  <
0*/){ // old knee soft min = 0.14

                sensorData->jointData[i].Thm            = 0; //sensorData->jointData[i].Thm*THMReductionFactor;
                sensorData->jointData[i].againstStop    = TRUE;
                //sensorData->jointData[i].Tlin         = sensorDa+0.17ta->jointData[i].Tlin + (sysProperties-
>jointControl[i-1].manualTorque*10*((KNEE_MIN + 0.14)-sensorData->jointData[i].position));
                //sysProperties->jointControl[j].CiSwitch = 0; // Ci=0 is OFF

            }else if(sensorData->jointData[i].position > (KNEE_MAX - KNEE_MAX_SOFT_OFFSET) /* && sensorData-
>jointData[i].Thm  > 0*/){ // old knee soft max = 0.30

                sensorData->jointData[i].Thm            = 0; //sensorData->jointData[i].Thm*THMReductionFactor;
                sensorData->jointData[i].againstStop    = TRUE;
                //sysProperties->jointControl[j].CiSwitch = 0; // Ci=0 is OFF

            }else{

                sensorData->jointData[i].againstStop    = FALSE;
                sysProperties->jointControl[j].CiSwitch    = 1; // Ci=1 is ON

            }
        }else if(i == RHIP || i == LHIP){
```

```
            if(sensorData->jointData[i].position < (HIP_MIN + HIP_MIN_SOFT_OFFSET)  /*&& sensorData->jointData[i].Thm  <
0*/){ // old hip soft min = 0.17

                sensorData->jointData[i].Thm            = 0; //sensorData->jointData[i].Thm*THMReductionFactor;
                sensorData->jointData[i].againstStop    = TRUE;
                //sysProperties->jointControl[j].CiSwitch = 0; // Ci=0 is OFF

            }else{

                sensorData->jointData[i].againstStop    = FALSE;
                sysProperties->jointControl[j].CiSwitch    = 1; // Ci=1 is ON

            }
        }

        THM[j]  = sensorData->jointData[i].Thm; // get human machine torque
        Tlin[j] = sensorData->jointData[i].Tlin;

        ////////////////////////////////////////////////////////////////////////////////
        // temporary -- these values were determined experimentally on 2004-09-10 for EX02
        Tfriction[j] = 0;
//        if (j == LANKLE_T)
//            Tfriction[j] = -4.25;
//        else if (j == LKNEE_T)
//            Tfriction[j] =  2.00;
//        else if (j == LHIP_T)
//            Tfriction[j] =  2.00;
//        else if (j == RANKLE_T)
//            Tfriction[j] = -9.00;
//        else if (j == RKNEE_T)
//            Tfriction[j] =  2.00;
//        else if (j == LHIP_T)
//            Tfriction[j] =  1.00;
        Tfriction[j] = Tfriction[j] + sysProperties->jointControl[j].manualTorque; // get friction from GUI, ignore stance
files
        ////////////////////////////////////////////////////////////////////////////////

//        // take derivative of THM (dTHM)
//        DfilterCoeffsTemp = DfilterCoeffs100;
//        dTHM[j] = THM[j];
//        //dTHM[j] = (filteredTHM[j][0] - filteredTHM[j][1])*FREQ;    // Calculate T_error time derivative
//        DifferentiatingFilter(unfiltered_dTHM[j], filtered_dTHM[j], &dTHM[j], DfilterCoeffsTemp); // filter dTHM
//        //LowpassFilter(unfiltered_dTHM[j], filtered_dTHM[j], &dTHM[j], filterCoeffs2nd100); // filter dTHM


//        // Compute the integral gain THM_int
//        if(THM[j] == 0 // if THM is zero
//            || sysProperties->torqueControl.controlFHM == 0    // or if THM control is switched OFF
//            || sysProperties->jointControl[j].kv==0 // or if the integral gain is zero
//            || ( j < RANKLE_T && sensorData->dynamicMode.LstanceSwingTransition) // of if this is a left leg joint going
into swing
//            || ( j > LHIP_T   && sensorData->dynamicMode.RstanceSwingTransition)){ // or a right leg joint going into
swing
//
//            THM_int[j] = 0; // reset integral to 0
//        }else{
//
//            THM_int[j] = THM_int[j] + THM[j]*TS; // otherwise integrate THM error
//        }


//        // filter THM
//
//        // jump or swing leg of s.stance mode
//        if ( sensorData->dynamicMode.Mode == JUMP  // jump
//            || ( sensorData->dynamicMode.Mode == SSTANCE  // s stance // this is a swing leg
//                  && ( (  (j == RHIP_T || j == RKNEE_T || j == RANKLE_T) && sensorData->dynamicMode.leftIsGrounded)
//                     || ((j == LHIP_T || j == LKNEE_T || j == LANKLE_T) && !sensorData->dynamicMode.leftIsGrounded) ) )
){
//
//            //filter_hack_working = 1; //use as a flag to see of we ever enter this
//
//
//            if(j == RHIP_T || j == LHIP_T){ // swing hip
//                if(THM[j] > THMmax){
//                    THM[j] = THMmax;   // saturate THM
//                }else if(THM[j] < -THMmax){
//                    THM[j] = -THMmax;
//                }else if(fabs(THM[j]) < THMmin){
//                    THM[j] = 0; // ignore any |THM| < THMmin
//                }
//                LowpassFilter(unfilteredTHM[j], filteredTHM[j], &THM[j], filterCoeffs2nd2);     // filter THM 5Hz (2Hz)
//            }else{ // swing knee and ankle
//                if( (fabs(Tlin[j] - filteredTlin[j][0]) > 1)){ // if there's a step input type difference it's a change
of state
//                    LowpassFilter(unfilteredTHM[j], filteredTHM[j], &THM[j], filterCoeffs2nd2);    // filter knee and
ankle THM 2Hz
//                }else{
//                    LowpassFilter(unfilteredTHM[j], filteredTHM[j], &THM[j], filterCoeffs2nd5);    // filter knee and
ankle THM 5Hz
//                }
//            }
//        }else{     // for stance
//
//            //#ifdef USE_BACKPACK_SENSOR
//            //            LowpassFilter(unfilteredTHM[j], filteredTHM[j], &THM[j], filterCoeffs2nd10);     // filter THM
before differentiation
//            //#else
//
```

```
//            //THM[j] = 0; // 10130308.exe, make sure THM is only used for swing leg
//
//            if(j == RHIP_T || j == LHIP_T){
//                if( (fabs(Tlin[j] - filteredTlin[j][0]) > 1)){ // if there's a step input type difference it's a change
of state
//                    LowpassFilter(unfilteredTHM[j], filteredTHM[j], &THM[j], filterCoeffs2nd2);     // filter hip THM 2Hz
//                }else{
//                    LowpassFilter(unfilteredTHM[j], filteredTHM[j], &THM[j], filterCoeffs2nd5);     // filter hip THM 5Hz
//                }
//            }else if(j == RKNEE_T || j == LKNEE_T){
//                LowpassFilter(unfilteredTHM[j], filteredTHM[j], &THM[j], filterCoeffs2nd2); // filter knee THM 2Hz
//            }else{ // ankle
//                LowpassFilter(unfilteredTHM[j], filteredTHM[j], &THM[j], filterCoeffs2nd2);  // filter ankle THM 2Hz
//            }
//            //#endif
//        }
//
//
////        // get derivative of Tlin
////        dTlin[j] = Tlin[j];
////        DifferentiatingFilter(unfiltered_dTlin[j], filtered_dTlin[j], &dTlin[j], DfilterCoeffs10);
////        kp = sysProperties->jointControl[j].kp;    // get latest feedback gains
////        sensorData->jointData[j].Tvlimit = dTlin[j];
//
//        // filter Tlin (linearization torque)
//        //if(sensorData->dynamicMode.Mode == SSTANCE && (sensorData->dynamicMode.LstanceSwingTransition || sensorData-
>dynamicMode.RstanceSwingTransition))
//        //    knee_Tlin_filter_counter = 0;
//
//        if(j==LANKLE_T || j==RANKLE_T){ // for the left or right ankle
//
//            if( (sensorData->dynamicMode.Mode == ONE_REDUNDANCY &&( (j==LANKLE_T && sensorData-
>dynamicMode.leftIsRedundant)
//                    || (j==RANKLE_T && !sensorData->dynamicMode.leftIsRedundant)))
//                /*
//                    ||( sensorData->dynamicMode.Mode == SSTANCE && ( (j==LANKLE_T && !sensorData-
>dynamicMode.leftIsGrounded)
//                    || (j==RANKLE_T && sensorData->dynamicMode.leftIsGrounded) ) )*/){
//
//                LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd2);
//            }
//            else if( fabs(Tlin[j] - filteredTlin[j][0]) > 1){// if there's a step input difference it's a change of state
//
//                /*if (sensorData->dynamicMode.Mode == ONE_REDUNDANCY &&( (j==LANKLE_T && sensorData-
>dynamicMode.leftIsRedundant)
//                    || (j==RANKLE_T && !sensorData->dynamicMode.leftIsRedundant))) {
//                    LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd1);
//                }
//
//                else */
//                if( sensorData->dynamicMode.Mode == SSTANCE && ( (j==LANKLE_T && !sensorData->dynamicMode.leftIsGrounded)
//                    || (j==RANKLE_T && sensorData->dynamicMode.leftIsGrounded) ) ){    // if this is a swing leg
//
//                    LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd10); // filter Tlin @1Hz
(5Hz better 07 1 03), set back to 1Hz 08-11-03 good?, then set to 10Hz...why?
//                                                                                                    // filter more because
foot oscillates after toe off
//                }
//                else if( !( ( (Tlin[j] > 0 && unfilteredTlin[j][0] > 0)  && (Tlin[j] < unfilteredTlin[j][0]) )
//                        ||  ( (Tlin[j] < 0 && unfilteredTlin[j][0] < 0)  && (Tlin[j] > unfilteredTlin[j][0])) )) {
//                                                    // Stance leg with above transition
//                    LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd2); // filter Tlin
@2Hz.
//                }else{                                                                              // Stance leg with
transition
//                    LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd5); // filter Tlin @5Hz
//                }
//            }
//            else{ // if there's no change of state, filter normally
//                LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd10);     // filter Tlin @10Hz
//            }
//
//        }
//        else { // for the knee or the hip
//            /*
//            if( sensorData->dynamicMode.Mode == SSTANCE && ( (j==LHIP_T && !sensorData->dynamicMode.leftIsGrounded)
//                || (j==RHIP_T && sensorData->dynamicMode.leftIsGrounded))
//                ||(sensorData->dynamicMode.Mode == ONE_REDUNDANCY &&( (j==LHIP_T && sensorData-
>dynamicMode.leftIsRedundant)
//                || (j==RHIP_T && !sensorData->dynamicMode.leftIsRedundant)))){
//
//                LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs1);
//
//            //    if (knee_Tlin_filter_counter < (sysProperties->ditherAmplitude*2)){
//            //        LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd025); // filter Tlin
@0.25Hz
//            //        knee_Tlin_filter_counter++;
//            //        filter_hack_working = 1;
//            //    }
//
//            }
//            else*/
//            if( (fabs(Tlin[j] - filteredTlin[j][0]) > 1)){ // if there's a step input type difference it's a change of
state
//                /*
//                if (sensorData->dynamicMode.Mode == ONE_REDUNDANCY && (j==LKNEE_T && sensorData-
>dynamicMode.leftIsRedundant)
//                    || (j==RKNEE_T && !sensorData->dynamicMode.leftIsRedundant)){
//                    LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd1);
```

```
//                }
//                else */
//                if( sensorData->dynamicMode.Mode == SSTANCE && ( (j==LKNEE_T && !sensorData->dynamicMode.leftIsGrounded)
//                    || (j==RKNEE_T && sensorData->dynamicMode.leftIsGrounded) ) ){    // if this is a swing leg, KNEE
//
//                    LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd2); // filter Tlin @2Hz
added to prevent jerkiness during swing
//                }
//                else if( sensorData->dynamicMode.Mode == SSTANCE && ( (j==LHIP_T && !sensorData-
>dynamicMode.leftIsGrounded)
//                    || (j==RHIP_T && sensorData->dynamicMode.leftIsGrounded) ) ){    // if this is a swing leg, HIP
//
//                    LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd2); // filter Tlin @0.5Hz
added to prevent jerkiness during swing, changed to 2Hz by JLR "a long time ago"
//                }
//                else {
//                    LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd2); // filter Tlin @2Hz
//                }
//            }
//            else{
//                LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd10);     // filter Tlin @10Hz
//            }
//        }


        // turn OFF filters for Thm and Tlin when the exo is against the hard stops (i.e. we don't want it to "stick" to
stops)
        if (!sensorData->jointData[i].againstStop){ // exo is NOT against stops
            LowpassFilter(unfilteredTHM[j],  filteredTHM[j],  &THM[j],  filterCoeffs2nd20); // filter THM @ 10Hz
            LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffs2nd100); // filter Tlin @ 100Hz
        }else{ // exo IS against stops, turn off filters
            LowpassFilter(unfilteredTHM[j],  filteredTHM[j],  &THM[j],  filterCoeffsOFF); // filter is OFF
            LowpassFilter(unfilteredTlin[j], filteredTlin[j], &Tlin[j], filterCoeffsOFF); // filter is OFF
        }

        // Simulates spring at ankle.  Added by JRS, 2004-07-28
        if (sysProperties->debuggingControls.activateAnkleSpring == TRUE){
            if (j == LANKLE_T){
                Tlin[j] = (sensorData->jointData[LANKLE].position - (sysProperties->debuggingControls.centerAngle*Pi/180))
* sysProperties->debuggingControls.springRate;
            }else if(j == RANKLE_T){
                Tlin[j] = (sensorData->jointData[RANKLE].position - (sysProperties->jointControl[RANKLE_T].kv*Pi/180)) *
sysProperties->debuggingControls.springRate;
            }
        }

        // Simulates damper at knee during stance.  Becomes pivot during swing.
        // Added by JWR, 2004-08-03
        if (sysProperties->debuggingControls.activateKneeDamper == TRUE){
            if (j == LKNEE_T){
                if (sensorData->dynamicMode.Mode == SSTANCE && (sensorData->dynamicMode.groundedLeg == RIGHT)){
                    //Tlin[j] = 0; // VFC mode - no human machine torque
                }else{
                    Tlin[j] = Tlin[j]-(sensorData->jointData[LKNEE].velocity * sysProperties-
>debuggingControls.flexionDampingCoeff);
                }
            }else if (j == RKNEE_T){
                if (sensorData->dynamicMode.Mode == SSTANCE && (sensorData->dynamicMode.groundedLeg == LEFT)){
                    //Tlin[j] = 0;
                }else{
                    Tlin[j] = Tlin[j]-(sensorData->jointData[RKNEE].velocity * sysProperties-
>debuggingControls.flexionDampingCoeff);
                }
            }
        }

        // calculate FHM torques (torques due to interaction between human and machine)
        FHMTorques[j] = (kpFHM * THM[j]) + Tlin[j] + Tfriction[j]; // new control scheme, JRS, 2004-06-25

        if(j==LANKLE_T || j==RANKLE_T){ // saturate ankle negative torque in double stance or single stance leg
            if(sensorData->dynamicMode.Mode == DSTANCE
                || sensorData->dynamicMode.Mode == ONE_REDUNDANCY
                || (sensorData->dynamicMode.Mode == SSTANCE
                && ((j==LANKLE_T && (sensorData->dynamicMode.groundedLeg == LEFT))
                || (j==RANKLE_T && (sensorData->dynamicMode.groundedLeg == RIGHT))))){

                if(FHMTorques[j] < MINIMUM_ANKLE_TORQUE)
                    FHMTorques[j] = MINIMUM_ANKLE_TORQUE; // default is -5.0 Nm
            }
        }

        // using more feedback linearization  ( Tdes = K*THM + Tg + M*ddq - ddq  gives very linear eq of motion (1+K)THM =
ddq )
        //accel[j] = sensorData->jointData[i].acceleration;
        //LowpassFilter(unfiltered_dTHM[j], filtered_dTHM[j], &accel[j] , filterCoeffs2nd5); // filter acceleration
        //FHMTorques[i] = (kpFHM + 1)*THM[j] + sensorData->jointData[i].torque - accel[j];
    } // for (j=0; j<6; j++)
}


/* Function: GetTgAndTHM
* ------------------------------------------------------------------------------------------
* Calculates and updates the required gravity compensator torque vector.
* Calculates and outputs the joint torques due to human on the machine.
* The vectors are as follows:- [TankleL TkneeL ThipL TankleR TkneeR ThipR]
*/
void GetTgAndTHM(const BodyDataT    *bodyData,
                 SensorDataT        *sensorData,
                 SysPropertiesT        *sysProperties,
```

```
                ForceDistributionT *distrData)
{
    double angles[12], velocities[8], accelerations[8], torques[6];
    double torsoForces[3] = {0,0,0}; // backpack sensor forces
    int i;
    VirtualGuardT vguard;

    for (i=0; i<8; i++){
        angles[i]        = sensorData->jointData[i].position; // get joint angles
        velocities[i]    = sensorData->jointData[i].velocity;
        accelerations[i] = sensorData->jointData[i].acceleration;
    }

    angles[LHIP_ROT] = sensorData->hipData.L_rotation; // hip unactuated joint angles
    angles[LHIP_ABD] = sensorData->hipData.L_abduction;
    angles[RHIP_ROT] = sensorData->hipData.R_rotation;
    angles[RHIP_ABD] = sensorData->hipData.R_abduction;

    torques[LANKLE_T] = sensorData->jointData[LANKLE].torque; // get actuator joint torques
    torques[LKNEE_T]  = sensorData->jointData[LKNEE].torque;
    torques[LHIP_T]   = sensorData->jointData[LHIP].torque;
    torques[RANKLE_T] = sensorData->jointData[RANKLE].torque;
    torques[RKNEE_T]  = sensorData->jointData[RKNEE].torque;
    torques[RHIP_T]   = sensorData->jointData[RHIP].torque;

    torsoForces[TORSO_FX] = sensorData->torsoForce.Fx; // make torso sensor force/torque vector
    torsoForces[TORSO_FY] = sensorData->torsoForce.Fy;
    torsoForces[TORSO_T]  = sensorData->torsoForce.T;

    vguard = sysProperties->virtualGuard;

    //sensorData->dynamicMode.Mode = DSTANCE; // force exo into dynamic mode DSTANCE, better to do this by forcing foot
switches, JRS 04-27-2004

    switch (sensorData->dynamicMode.Mode){ // use dynamic equations corresponding to current dynamic mode

    case JUMP: // Jump
        JumpTgandTHM(angles,
                     velocities,
                     accelerations,
                     torques,
                     bodyData,
                     distrData,
                     sensorData);
        break;

    case SSTANCE: // Single Support
        SingleSupportTHM(angles,
                         velocities,
                         accelerations,
                         torques,
                         bodyData,
                         sensorData->dynamicMode.groundedLeg,
                         sensorData->dynamicMode.leftHeelContact,
                         sensorData->dynamicMode.rightHeelContact,
                         distrData,
                         sensorData,
                         sysProperties);
        break;

    case DSTANCE: // Double Support
        DoubleSupportTHM(angles,
                         velocities,
                         accelerations,
                         torques,
                         bodyData,
                         sysProperties->DynDistributionFactor,
                         torsoForces,
                         distrData,
                         sensorData,
                         vguard,
                         sysProperties);
        break;

    case ONE_REDUNDANCY: // Double Support Single Redundancy
        DoubleSupportSingleRedundancyTHM(angles,
                                         velocities,
                                         accelerations,
                                         torques,
                                         bodyData,
                                         sensorData->dynamicMode.redundantLeg,
                                         sensorData->dynamicMode.leftHeelContact,
                                         sensorData->dynamicMode.rightHeelContact,
                                         sysProperties->DynDistributionFactor,
                                         torsoForces,
                                         distrData,
                                         sensorData,
                                         vguard,
                                         sysProperties);
        break;

    case TWO_REDUNDANCY: // Double Support Double Redundancy (get rid of this state?)
        DoubleSupportDoubleRedundancyTHM(angles,
                                         velocities,
                                         accelerations,
                                         torques,
                                         bodyData,
                                         sensorData->dynamicMode.leftHeelContact,
                                         sensorData->dynamicMode.rightHeelContact,
```

```
                                        sysProperties->DynDistributionFactor,
                                        torsoForces,
                                        distrData,
                                        sensorData,
                                        vguard);
            break;
        }
    }
```

Appendix A.12 – Jump.h

```
/* Function: JumpTgandTHM
 * --------------------------------
 * Calculates the joint torques or operational space vector due to the human during jump mode and updates THM.
 * The joint torques due to gravity Tg are also computed and updated.
 * The vector is as follows: [TankleL TkneeL ThipL TankleR TkneeR ThipR]
 */
void JumpTgandTHM( const double *angles, const double *velocities,
            const double *accelerations, const double * torques, const BodyDataT *bodyData, ForceDistributionT *distrData,
            SensorDataT *sensorData);

/* Function: GetTfrictionSwing
 * ---------------------------------------------
 * Calculates the joint torques vector to counteract joint friction and stiffness and updates Tf.
 * The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
 */
void GetTfrictionSwing(double *Tf, const double *angles, const double *velocities, const int legSide);
```

Appendix A.13 – Jump.c

```
#include <math.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"
#include "Jump.h"
#include "DSup.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];

extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];


/* Function: JumpTgandTHM
 * --------------------------------------------------------------------------------------------
 * Calculates the joint torques or operational space vector due to the human during jump mode and updates THM.
 * The joint torques due to gravity Tg are also computed and updated.
 * The vector is as follows: [TankleL TkneeL ThipL TankleR TkneeR ThipR]
 */
void JumpTgandTHM(const double           *angles,
                  const double           *velocities,
                  const double           *accelerations,
                  const double           *torques,
                  const BodyDataT       *bodyData,
                  ForceDistributionT    *distrData,
                  SensorDataT            *sensorData){

    double Tf[3];
    double mf,If,LGf,hGf,msh,Is,Ls,LGs,LGsp,hGs,mt,It,Lt,LGt,LGtp,hGt; // segment properties
    double q5,q6,q7, dq5,dq6,dq7,ddq5,ddq6,ddq7,T5, T6, T7, Tg5, Tg6, Tg7, Tcc5, Tcc6, Tcc7;
    double c5,s5,c56,s56,c567,s567,dq56,dq567;
    int    i, legSide;

    mf   = bodyData->foot.mass; // get mass and geometric properties
    If   = bodyData->foot.inertia;
    LGf  = bodyData->foot.Lcg;
    hGf  = bodyData->foot.hcg;
    msh  = bodyData->shank.mass;
    Is   = bodyData->shank.inertia;
    Ls   = bodyData->shank.length;
    LGs  = bodyData->shank.Lcg;
    LGsp = LGs - Ls;
    hGs  = bodyData->shank.hcg;
     mt   = bodyData->thigh.mass;
    It   = bodyData->thigh.inertia;
    Lt   = bodyData->thigh.length;
    LGt  = bodyData->thigh.Lcg;
    LGtp = LGt - Lt;
    hGt  = bodyData->thigh.hcg;

    // Run Jump.m in MATLAB to create dynamic equations

    for (legSide=0; legSide<2; legSide++) { // repeat torque calculation for each leg (0 = left leg, 1 = right leg)

        //define kinematic variables according to the leg
         if (legSide==0){ //    Left Leg
            q5   = -angles[LHIP] + sensorData->TorsoTilt;
            q6   = -angles[LKNEE];
            q7   = -angles[LANKLE];
            dq5  = -velocities[LHIP] + sensorData->torsoVelocity;
            dq6  = -velocities[LKNEE];
            dq7  = -velocities[LANKLE];
            ddq5 = -accelerations[LHIP] + sensorData->bodyAccel[UPPERBODY].angular_accel;
            ddq6 = -accelerations[LKNEE];
            ddq7 = -accelerations[LANKLE];
        }else{ // Right Leg
            q5   = -angles[RHIP] + sensorData->TorsoTilt;
            q6   = -angles[RKNEE];
            q7   = -angles[RANKLE];
               dq5  = -velocities[RHIP] + sensorData->torsoVelocity;
            dq6  = -velocities[RKNEE];
            dq7  = -velocities[RANKLE];
               ddq5 = -accelerations[RHIP] + sensorData->bodyAccel[UPPERBODY].angular_accel;
            ddq6 = -accelerations[RKNEE];
            ddq7 = -accelerations[RANKLE];
        }

        dq56 = dq5+dq6;                dq567 = dq7+dq6+dq5;
```

```
        c5   = cos(q5);                s5    = sin(q5);
        c56  = cos(q5+q6);             s56   = sin(q5+q6);
           c567 = cos(q5+q6+q7);         s567  = sin(q5+q6+q7);

        for (i=0;i<4;i++){
            distrData->unfilteredBetaFg[i]  = 0.5; // reset load distribution parameters used in double stance
            distrData->filteredBetaFg[i]    = 0.5;
            distrData->unfilteredBetaFHM[i] = 0.5;
            distrData->filteredBetaFHM[i]   = 0.5;
            distrData->unfilteredKrot[i]    = 1;
            distrData->filteredKrot[i]      = 1;
        }

        // Compute torque induced by gravity on the distal segment of the machine

        Tg5 = -mf*g*(Lt*(s5)+Ls*(s56)+LGf*(c567)-hGf*(s567))-msh*g*(Lt*(s5)-LGsp*(s56)+hGs*(c56))-mt*g*(-
LGtp*(s5)+hGt*(c5)); // JumpV5.txt
        Tg6 = -mf*g*(Ls*(s56)+LGf*(c567)-hGf*(s567))-msh*g*(-LGsp*(s56)+hGs*(c56)); // JumpV6.txt
        Tg7 = -mf*g*(LGf*(c567)-hGf*(s567)); // JumpV7.txt

        // Compute human torques due to  Coriolis and centrifugal forces

        Tcc5 = 0.5*mf*(2*(-dq56*dq56*Ls*(s56)-dq5*dq5*Lt*(s5)-((dq567))*(LGf*(c567)*((dq567))-
hGf*(s567)*((dq567))))*(Lt*(c5)+Ls*(c56)-LGf
        *(s567)-hGf*(c567))+2*(Lt*(s5)*dq5+Ls*(s56)*((dq56))+((dq567))*(LGf*(c567)-
hGf*(s567)))*(dq5*Lt*(c5)+((dq56))*Ls*(c56)-LGf*(s567)*((dq567))
        -hGf*(c567)*((dq567)))+2*(dq5*Lt*(c5)+((dq56))*Ls*(c56)-((dq567))*(LGf*(s567)+hGf*(c567)))*(-Lt*(s5)*dq5-
Ls*(s56)*((dq56))-LGf*(c567)*(((dq7
        +dq6+dq5)))+hGf*(s567)*((dq567)))+2*(dq56*dq56*Ls*(c56)+dq5*dq5*Lt*(c5)+((dq567))*(-LGf*(s567)*((dq567))-
hGf*(c567)*(((dq7
        +dq6+dq5)))))*(Lt*(s5)+Ls*(s56)+LGf*(c567)-hGf*(s567)))+0.5*mt*(4*dq5*(LGtp*(c5)+hGt*(s5))*(-
LGtp*(s5)*dq5+hGt*(c5)*dq5)+4*dq5*(-LGtp*(s5)+hGt
        *(c5))*(-LGtp*(c5)*dq5-hGt*(s5)*dq5))-0.5*mf*(2*(dq5*Lt*(c5)+((dq56))*Ls*(c56)-
((dq567))*(LGf*(s567)+hGf*(c567)))*(-Lt*(s5)*dq5-Ls*(s56)*((dq56))
        -((dq567))*(LGf*(c567)-hGf*(s567)))+2*(Lt*(s5)*dq5+Ls*(s56)*((dq56))+((dq567))*(LGf*(c567)-
hGf*(s567)))*(dq5*Lt*(c5)+((dq56))*Ls*(c56)+(((dq7
        +dq6+dq5)))*(-LGf*(s567)-hGf*(c567))))-0.5*mt*(2*dq5*dq5*(LGtp*(c5)+hGt*(s5))*(-LGtp*(s5)+hGt*(c5))+2*dq5*dq5*(-
LGtp*(s5)+hGt*(c5))*(
        -LGtp*(c5)-hGt*(s5)))+0.5*msh*(2*(-((dq56))*(-LGsp*(s56)*((dq56))+hGs*(c56)*((dq56)))-dq5*dq5*Lt*(s5))*(Lt*(c5)-
LGsp*(c56)-hGs*(s56))+2*(Lt
        *(s5)*dq5+((dq56))*(-LGsp*(s56)+hGs*(c56)))*(dq5*Lt*(c5)-LGsp*(c56)*((dq56))-hGs*(s56)*((dq56)))+2*(dq5*Lt*(c5)-
((dq56))*(LGsp*(c56)+hGs*(s56)))
        *(-Lt*(s5)*dq5+LGsp*(s56)*((dq56))-hGs*(c56)*((dq56)))+2*(((dq56))*(-LGsp*(c56)*((dq56))-
hGs*(s56)*((dq56)))+dq5*dq5*Lt*(c5))*(Lt*(s5)-LGsp
        *(s56)+hGs*(c56)))-0.5*msh*(2*(dq5*Lt*(c5)-((dq56))*(LGsp*(c56)+hGs*(s56)))*(-Lt*(s5)*dq5-((dq56))*(-
LGsp*(s56)+hGs*(c56)))+2*(Lt*(s5)*dq5+((dq56))
        *(-LGsp*(s56)+hGs*(c56)))*(dq5*Lt*(c5)+((dq56))*(-LGsp*(c56)-hGs*(s56))));

        Tcc6 = 0.5*mf*(2*(-dq56*dq56*Ls*(s56)-dq5*dq5*Lt*(s5)-((dq567))*(LGf*(c567)*((dq567))-hGf*(s567)*((dq567))))
        *(Ls*(c56)-LGf*(s567)-hGf*(c567))+2*(dq5*Lt*(s5)+Ls*(s56)*((dq56))+((dq567))*(LGf*(c567)-
hGf*(s567)))*(((dq56))*Ls*(c56)-LGf*(s567)*((dq567))
        -hGf*(c567)*((dq567)))+2*(dq5*Lt*(c5)+((dq56))*Ls*(c56)-((dq567))*(LGf*(s567)+hGf*(c567)))*(-Ls*(s56)*((dq56))-
LGf*(c567)*((dq567))
        +hGf*(s567)*((dq567)))+2*(dq56*dq56*Ls*(c56)+dq5*dq5*Lt*(c5)+((dq567))*(-LGf*(s567)*((dq567))-
hGf*(c567)*((dq567))))
        *(Ls*(s56)+LGf*(c567)-hGf*(s567)))-0.5*mf*(2*(dq5*Lt*(c5)+((dq56))*Ls*(c56)-((dq567))*(LGf*(s567)+hGf*(c567)))*(-
Ls*(s56)*((dq56))-((dq567))
        *(LGf*(c567)-hGf*(s567)))+2*(dq5*Lt*(s5)+Ls*(s56)*((dq56))+((dq567))*(LGf*(c567)-
hGf*(s567)))*(((dq56))*Ls*(c56)+((dq567))*(-LGf*(s567)
        -hGf*(c567))))+0.5*msh*(2*(-((dq56))*(-LGsp*(s56)*((dq56))+hGs*(c56)*((dq56)))-dq5*dq5*Lt*(s5))*(-LGsp*(c56)-
hGs*(s56))+2*(dq5*Lt*(s5)+((dq56))
        *(-LGsp*(s56)+hGs*(c56)))*(-LGsp*(c56)*((dq56))-hGs*(s56)*((dq56)))+2*(dq5*Lt*(c5)-
((dq56))*(LGsp*(c56)+hGs*(s56)))*(LGsp*(s56)*((dq56))
        -hGs*(c56)*((dq56)))+2*(((dq56))*(-LGsp*(c56)*((dq56))-hGs*(s56)*((dq56)))+dq5*dq5*Lt*(c5))*(-
LGsp*(s56)+hGs*(c56)))-0.5*msh*(-2*(dq5*Lt*(c5)
        -((dq56))*(LGsp*(c56)+hGs*(s56)))*((dq56))*(-LGsp*(s56)+hGs*(c56))+2*(dq5*Lt*(s5)+((dq56))*(-
LGsp*(s56)+hGs*(c56)))*((dq56))*(-LGsp*(c56)-hGs*(s56)));

        Tcc7 = 0.5*mf*(2*(-dq56*dq56*Ls*(s56)-dq5*dq5*Lt*(s5)-((dq567))*(LGf
        *(c567)*((dq567))-hGf*(s567)*((dq567))))*(-LGf*(s567)-
hGf*(c567))+2*(dq5*Lt*(s5)+((dq56))*Ls*(s56)+((dq567))*(LGf*(c567)-hGf*(s567)))
        *(-LGf*(s567)*((dq567))-hGf*(c567)*((dq567)))+2*(dq5*Lt*(c5)+((dq56))*Ls*(c56)-
((dq567))*(LGf*(s567)+hGf*(c567)))*(-LGf*(c567)*(((dq7
        +dq6+dq5)))+hGf*(s567)*((dq567)))+2*(dq56*dq56*Ls*(c56)+dq5*dq5*Lt*(c5)+((dq567))*(-LGf*(s567)*((dq567))-
hGf*(c567)*(((dq7
        +dq6+dq5)))))*(LGf*(c567)-hGf*(s567)))-0.5*mf*(-2*(dq5*Lt*(c5)+((dq56))*Ls*(c56)-
((dq567))*(LGf*(s567)+hGf*(c567)))*((dq567))*(LGf*(c567)
        -hGf*(s567))+2*(dq5*Lt*(s5)+((dq56))*Ls*(s56)+((dq567))*(LGf*(c567)-hGf*(s567)))*((dq567))*(-LGf*(s567)-
hGf*(c567)));

        // Compute human torques due to ang. acceleration only

        T5 = (If+0.5*mf*(2*(-LGf*(s567)-hGf*(c567))*(Lt*(c5)+Ls*(c56)-LGf*(s567)-hGf*(c567))+2*(LGf*(c567)-
hGf*(s567))*(Lt*(s5)+Ls*(s56)+LGf*(c567)-hGf*(s567))))*ddq7
        +(If+0.5*msh*(2*(-LGsp*(c56)-hGs*(s56))*(Lt*(c5)-LGsp*(c56)-hGs*(s56))+2*(-LGsp*(s56)+hGs*(c56))*(Lt*(s5)-
LGsp*(s56)+hGs*(c56)))+Is+0.5*mf
        *(2*(Ls*(c56)-LGf*(s567)-hGf*(c567))*(Lt*(c5)+Ls*(c56)-LGf*(s567)-hGf*(c567))+2*(Ls*(s56)+LGf*(c567)-
hGf*(s567))*(Lt*(s5)+Ls*(s56)+LGf*(c567)-hGf*(s567))))
        *ddq6+(0.5*mf*(2*(Lt*(s5)+Ls*(s56)+LGf*(c567)-hGf*(s567))*(Lt*(s5)+Ls*(s56)+LGf*(c567)-
hGf*(s567))+2*(Lt*(c5)+Ls*(c56)-LGf*(s567)-hGf*(c567))
        *(Lt*(c5)+Ls*(c56)-LGf*(s567)-hGf*(c567)))
        +If+0.5*msh*(2*(Lt*(s5)-LGsp*(s56)+hGs*(c56))*(Lt*(s5)-LGsp*(s56)+hGs*(c56))+2*(Lt*(c5)-LGsp*(c56)-
hGs*(s56))*(Lt*(c5)-LGsp*(c56)-hGs*(s56)))
        +It+Is+0.5*mt*(2*(LGtp*(c5)+hGt*(s5))*(LGtp*(c5)+hGt*(s5))+2*(-LGtp*(s5)+hGt*(c5))*(-LGtp*(s5)+hGt*(c5))))*ddq5;
            // JumpKE5.txt

        T6 = (If+0.5*mf*(2*(-LGf*(s567)-hGf*(c567))*(Ls*(c56)-LGf*(s567)-hGf*(c567))+2*(LGf*(c567)-
hGf*(s567))*(Ls*(s56)+LGf*(c567)-hGf*(s567))))*ddq7+(0.5*mf*(2
```

```
        *(Ls*(c56)-LGf*(s567)-hGf*(c567))*(Ls*(c56)-LGf*(s567)-hGf*(c567))+2*(Ls*(s56)+LGf*(c567)-
hGf*(s567))*(Ls*(s56)+LGf*(c567)-hGf*(s567)))+0.5*msh*(2
        *(-LGsp*(c56)-hGs*(s56))*(-LGsp*(c56)-hGs*(s56))+2*(-LGsp*(s56)+hGs*(c56))*(-LGsp*(s56)+hGs*(c56)))
        +Is+If)*ddq6+(If+0.5*mf*(2*(Lt*(s5)+LGf*(c567)-hGf*(s567)+Ls*(s56))*(Ls*(s56)+LGf*(c567)-hGf*(s567))+2*(Lt*(c5)-
LGf*(s567)-hGf*(c567)+Ls*(c56))*(Ls*(c56)
        -LGf*(s567)-hGf*(c567)))+Is+0.5*msh*(2*(-LGsp*(s56)+hGs*(c56)+Lt*(s5))*(-LGsp*(s56)+hGs*(c56))+2*(-LGsp*(c56)-
hGs*(s56)+Lt*(c5))*(-LGsp*(c56)
        -hGs*(s56))))*ddq5;
            // JumpKE6.txt

        T7 = (0.5*mf*(2*(-LGf*(s567)-hGf*(c567))*(-LGf*(s567)-hGf*(c567))+2*(LGf*(c567)-hGf*(s567))*(LGf*(c567)-
hGf*(s567)))+If)*ddq7+(0.5*mf*(2
            *(Ls*(c56)-LGf*(s567)-hGf*(c567))*(-LGf*(s567)-hGf*(c567))
        +2*(Ls*(s56)+LGf*(c567)-hGf*(s567))*(LGf*(c567)-hGf*(s567)))+If)*ddq6+(0.5*mf*(2*(Lt*(s5)+LGf*(c567)-
hGf*(s567)+Ls*(s56))*(LGf*(c567)-hGf*(s567))+2*(Lt
        *(c5)-LGf*(s567)-hGf*(c567)+Ls*(c56))*(-LGf*(s567)-hGf*(c567)))+If)*ddq5;
             // JumpKE7.txt

        // OUTPUT:
          // add gravity and actuator torques to compute total human-machine torques
         // Torques are multiplied by (-1) since they should be expressed as the torque of the distal segment on
         //    the proximal segment whereas the computed torques represent the oppposite sign convention.
          if (legSide==0){ //    Left Leg
            GetTfrictionSwing(Tf, &angles[LANKLE], &velocities[LANKLE], legSide); // (left) get joint friction and
stiffness torques

            sensorData->jointData[LANKLE].Tg = Tg7; // ankle torque necessary to support segment against gravity and
velocity forces
             sensorData->jointData[LKNEE].Tg  = Tg6; // knee
             sensorData->jointData[LHIP].Tg   = Tg5; // hip

            sensorData->jointData[LANKLE].Tcc = - Tcc7; // torque necessary to support segment against velocity forces
             sensorData->jointData[LKNEE].Tcc  = - Tcc6; // Negative sign is due to different sign conventino in controller
             sensorData->jointData[LHIP].Tcc   = - Tcc5 ;

            sensorData->jointData[LANKLE].Tf = Tf[ANKLE_T]; // torque necessary to compensate friction
            sensorData->jointData[LKNEE].Tf  = Tf[KNEE_T];
             sensorData->jointData[LHIP].Tf   = Tf[HIP_T];

//            sensorData->jointData[LANKLE].Thm = -T7-Tcc7; // ankle. note: actuator torques are already expressed in
correct coords
//             sensorData->jointData[LKNEE].Thm  = -T6-Tcc6; // knee
//             sensorData->jointData[LHIP].Thm   = -T5-Tcc5; // hip

            sensorData->jointData[LANKLE].Thm = Tf[ANKLE_T] - T7 + Tg7 - Tcc7 - torques[LANKLE_T]; // ankle. note: actuator
torques are already expressed in correct coords
            sensorData->jointData[LKNEE].Thm = Tf[KNEE_T] - T6 + Tg6 - Tcc6 - torques[LKNEE_T]; // knee
             sensorData->jointData[LHIP].Thm = Tf[HIP_T] - T5 + Tg5 - Tcc5 - torques[LHIP_T]; // hip

            sensorData->jointData[LANKLE].Tinertial = -T7; // Ankle torque due to inertial forces, JRS, 2004-06-24
             sensorData->jointData[LKNEE].Tinertial  = -T6; // Knee torque due to inertial forces, JRS, 2004-06-24
             sensorData->jointData[LHIP].Tinertial   = -T5; // Hip torque due to inertial forces, JRS, 2004-06-24

        }else{ // Right Leg
            GetTfrictionSwing(Tf, &angles[RANKLE], &velocities[RANKLE], legSide); // (right)

            sensorData->jointData[RANKLE].Tg = Tg7; // ankle
             sensorData->jointData[RKNEE].Tg  = Tg6; // knee
             sensorData->jointData[RHIP].Tg   = Tg5; // hip

            sensorData->jointData[RANKLE].Tcc = - Tcc7; // ankle
             sensorData->jointData[RKNEE].Tcc  = - Tcc6; // knee
             sensorData->jointData[RHIP].Tcc   = - Tcc5; // hip

            sensorData->jointData[RANKLE].Tf = Tf[ANKLE_T]; // torque necessary to compensate friction
            sensorData->jointData[RKNEE].Tf  = Tf[KNEE_T];
             sensorData->jointData[RHIP].Tf   = Tf[HIP_T];

//             sensorData->jointData[RANKLE].Thm = -T7-Tcc7; // ankle
//             sensorData->jointData[RKNEE].Thm  = -T6-Tcc6; // knee
//              sensorData->jointData[RHIP].Thm   = -T5-Tcc5; // hip

            sensorData->jointData[RANKLE].Thm = Tf[ANKLE_T] -T7 - Tcc7 + Tg7 - torques[RANKLE_T]; // ankle
            sensorData->jointData[RKNEE].Thm = Tf[KNEE_T] -T6 - Tcc6 + Tg6 - torques[RKNEE_T]; // knee
             sensorData->jointData[RHIP].Thm = Tf[HIP_T] -T5 - Tcc5 + Tg5 - torques[RHIP_T]; // hip

            sensorData->jointData[LANKLE].Tinertial = -T7; // Ankle torque due to inertial forces, JRS, 2004-06-24
             sensorData->jointData[LKNEE].Tinertial  = -T6; // Knee torque due to inertial forces, JRS, 2004-06-24
             sensorData->jointData[LHIP].Tinertial   = -T5; // Hip torque due to inertial forces, JRS, 2004-06-24


        }
    } // for loop

    sensorData->jointData[LANKLE].Tlin = sensorData->jointData[LANKLE].Tg
                                       + sensorData->jointData[LANKLE].Tf; // compute torque for feedback linearization

    sensorData->jointData[LKNEE].Tlin = sensorData->jointData[LKNEE].Tg
                                      + sensorData->jointData[LKNEE].Tf; // compute torque for feedback linearization

    sensorData->jointData[LHIP].Tlin = sensorData->jointData[LHIP].Tg
                                     + sensorData->jointData[LHIP].Tf; // compute torque for feedback linearization

    sensorData->jointData[RANKLE].Tlin = sensorData->jointData[RANKLE].Tg
                                       + sensorData->jointData[RANKLE].Tf; // compute torque for feedback linearization

    sensorData->jointData[RKNEE].Tlin = sensorData->jointData[RKNEE].Tg
                                      + sensorData->jointData[RKNEE].Tf; // compute torque for feedback linearization
```

```
    sensorData->jointData[RHIP].Tlin = sensorData->jointData[RHIP].Tg
                                     + sensorData->jointData[RHIP].Tf; // compute torque for feedback linearization

    // set toe torques to zero    (there's no actuator at the toe)
    sensorData->jointData[LTOE].Tg            = 0;
    sensorData->jointData[RTOE].Tg            = 0;
    sensorData->jointData[LTOE].Thm           = 0;
    sensorData->jointData[RTOE].Thm           = 0;
    sensorData->jointData[LTOE].Tlin        = 0;
    sensorData->jointData[RTOE].Tlin        = 0;
    sensorData->jointData[LTOE].Tcc           = 0;
    sensorData->jointData[RTOE].Tcc           = 0;
    sensorData->jointData[LTOE].Tf            = 0;
    sensorData->jointData[RTOE].Tf            = 0;
    sensorData->jointData[LTOE].Tinertial   = 0;
    sensorData->jointData[RTOE].Tinertial   = 0;
}


/* Function: GetTfrictionSwing
 * --------------------------------------------------------------------------------------------
 * Calculates the joint torques vector to counteract joint friction and stiffness for one leg. Updates Tf.
 * The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
 * Equations are obtained from Excel documents 'rankle stiffness.xls', 'rknee stiffness.xls', 'rhip stiffness.xls'.
 * legSide =0 for left leg and 1 for right leg
 */
void GetTfrictionSwing(double        *Tf,
                       const double    *angles,
                       const double    *velocities,
                       const int    legSide){

    double torque_slope[3], torque_offset[3], force_slope[3], force_offset[3],shankAngle;

    // the slope of the offsets of the joint stiffness torques w.r.t. to angular position, are function of velocity

    // Compute ankle stiffness

/*    torque_slope[0] = ANKLE_STIFFNESS_SLOPE_SLOPE * velocities[0] + ANKLE_STIFFNESS_SLOPE_OFFSET;

    if(velocities[0] >= 0){
        torque_offset[0] = ANKLE_STIFFNESS_OFFSET_POS_VEL;    // average offset for positive velocities
    }else{
        torque_offset[0] = ANKLE_STIFFNESS_OFFSET_NEG_VEL;    // average offset for negative velocities
    }

   Tf[0] = (torque_slope[0] * angles[0] + torque_offset[0]); // ankle friction (rankle)

*/

// USE ONLY STATIC FRICTION VALUES

//    Tf[0] = 0.0053 * ((angles[1] + angles[2])*180/Pi + 13.8) + 0.0297*angles[0]*180/Pi-2.5546
//            - 0.62468*9.81*(0.050978*cos(13.8*Pi/180+angles[0]+angles[1]+angles[2])
//            + 0.03244*sin(13.8*Pi/180+angles[0]+angles[1]+angles[2])); // old

//    Tf[0] = (0.0043 * ((angles[1] + angles[2])*180/Pi - 13.8) + 0.029*angles[0]*180/Pi-2.2366);
      //    - 0.62468*9.81*(0.050978*cos(-13.8*Pi/180+angles[0]+angles[1]+angles[2])
      //    + 0.03244*sin(-13.8*Pi/180+angles[0]+angles[1]+angles[2]));


    shankAngle = angles[1] + angles[2] - 13.8*Pi/180;

    if(angles[0] < -15*Pi/180){ //  -15 > ankle angle > -30
        if(shankAngle < -50*Pi/180){
            Tf[ANKLE_T] = -1.8; // -1.9;
        }else if(shankAngle < -40*Pi/180){
            Tf[ANKLE_T] = -1.8;
        }else if(shankAngle < -20*Pi/180){
            Tf[ANKLE_T] = -1.8; // -1.9;
        }else if(shankAngle < -10*Pi/180){
            Tf[ANKLE_T] = -1.8;
        }else if(shankAngle < 0){
            Tf[ANKLE_T] = -1.8;
        }else if(shankAngle < 5*Pi/180){ // from here on is for hip angle = 0
            Tf[ANKLE_T] = -1.65;
        }else if(shankAngle < 10*Pi/180){
            Tf[ANKLE_T] = -1.65;
        }else if(shankAngle < 15*Pi/180){
            Tf[ANKLE_T] = -1.75;
        }else if(shankAngle < 45*Pi/180){
            Tf[ANKLE_T] = -1.75;
        }else if(shankAngle < 60*Pi/180){
            Tf[ANKLE_T] = -1.75;
        }else if(shankAngle < 80*Pi/180){
            Tf[ANKLE_T] = -1.75;
        }else{
            Tf[ANKLE_T] = -2.5;
        }


    //    Tf[ANKLE_T] = -1.8;

    }else if(angles[0] < 0){ //  0 > ankle angle > -15
        if(shankAngle < -50*Pi/180){
            Tf[ANKLE_T] = -1.8;
        }else if(shankAngle < -40*Pi/180){
            Tf[ANKLE_T] = -1.8;
        }else if(shankAngle < -20*Pi/180){
            Tf[ANKLE_T] = -1.75;
```

```
        }else if(shankAngle < -10*Pi/180){
            Tf[ANKLE_T] = -1.75;
        }else if(shankAngle < 0){
            Tf[ANKLE_T] = -1.75; // -1.7
        }else if(shankAngle < 5*Pi/180){ // from here on is for hip angle = 0
            Tf[ANKLE_T] = -1.75;
        }else if(shankAngle < 10*Pi/180){
            Tf[ANKLE_T] = -1.75;
        }else if(shankAngle < 15*Pi/180){
            Tf[ANKLE_T] = -1.6;
        }else if(shankAngle < 45*Pi/180){
            Tf[ANKLE_T] = -1.5;
        }else if(shankAngle < 60*Pi/180){
            Tf[ANKLE_T] = -1.75;
        }else if(shankAngle < 80*Pi/180){
            Tf[ANKLE_T] = -1.95;
        }else{
            Tf[ANKLE_T] = -2.5;
        }


    //    Tf[ANKLE_T] = -1.75;

    }else if(angles[0] < 15*Pi/180){ //  0 < ankle angle < 15
        if(shankAngle < -50*Pi/180){
            Tf[ANKLE_T] = -1.75;
        }else if(shankAngle < -40*Pi/180){
            Tf[ANKLE_T] = -1.5;
        }else if(shankAngle < -20*Pi/180){
            Tf[ANKLE_T] = -1.5;
        }else if(shankAngle < -10*Pi/180){
            Tf[ANKLE_T] = -1.5; // -1.45
        }else if(shankAngle < 0){
            Tf[ANKLE_T] = -1.5;
        }else{
            Tf[ANKLE_T] = -1.25;
        }

//        Tf[ANKLE_T] = -1.5;

    }else if(angles[0] < 23*Pi/180){ //  15 < ankle angle < 23
        if(shankAngle < -50*Pi/180){
            Tf[ANKLE_T] = -1.3;
        }else if(shankAngle < -40*Pi/180){
            Tf[ANKLE_T] = -1.15;
        }else if(shankAngle < -20*Pi/180){
            Tf[ANKLE_T] = -1.25;
        }else if(shankAngle < -10*Pi/180){
            Tf[ANKLE_T] = -1.2;
        }else if(shankAngle < 0){
            Tf[ANKLE_T] = -1.7;
        }else if(shankAngle < 60*Pi/180){ // from here on is for hip angle = 0
            Tf[ANKLE_T] = -0.85;
        }else{
            Tf[ANKLE_T] = -0.5;
        }

//        Tf[ANKLE_T] = -1.2;


    }else{ // 23 < ankle angle < 30
        if(shankAngle < -50*Pi/180){
            Tf[ANKLE_T] = -1;
        }else if(shankAngle < -40*Pi/180){
            Tf[ANKLE_T] = -1;
        }else if(shankAngle < -20*Pi/180){
            Tf[ANKLE_T] = -1;
        }else if(shankAngle < -10*Pi/180){
            Tf[ANKLE_T] = -1; // -0.75;
        }else if(shankAngle < 0){
            Tf[ANKLE_T] = -1;
        }else if(shankAngle < 45*Pi/180){ // from here on is for hip angle = 0
            Tf[ANKLE_T] = -1.15;
        }else if(shankAngle < 60*Pi/180){
            Tf[ANKLE_T] = -0.7;
        }else{
            Tf[ANKLE_T] = -1;
        }


//        Tf[ANKLE_T]  =-1;
    }

    // Compute knee stiffness
/*
    if(angles[1] > 1.5){ // use different values for higher angles
        force_slope[1] = KNEE_STIFFNESS_SLOPE_A2 * velocities[1]*velocities[1] + KNEE_STIFFNESS_SLOPE_A1*velocities[1]
                        + KNEE_STIFFNESS_SLOPE_A0;

        force_offset[1] = KNEE_STIFFNESS_OFFSET_A2 * velocities[1]*velocities[1] + KNEE_STIFFNESS_OFFSET_A1*velocities[1]
                        + KNEE_STIFFNESS_OFFSET_A0;
    }else{
        force_slope[1] =  KNEE_STIFFNESS_SLOPE_SLOPE * velocities[1] + KNEE_STIFFNESS_SLOPE_OFFSET;
        force_offset[1] = KNEE_STIFFNESS_OFFSET_SLOPE * velocities[1] + KNEE_STIFFNESS_OFFSET_OFFSET;
    }

    Tf[1] = -momentArms[1]*(force_slope[1] * angles[1] + force_offset[1]); // knee stiffness (validated for right knee
only)
*/
```

```
    if (angles[2] > 0){ // change knee friction torque wih hip angle
        Tf[KNEE_T] = 0.3;
    }else if(angles[2] > -30*Pi/180){
        Tf[KNEE_T] = 0.4;
    }else if(angles[2] > -45*Pi/180){
        Tf[KNEE_T] = 0.5;
    }else{
        Tf[KNEE_T] = 0.6;
    }

    if(angles[1] > 1.8){ // if knee is almost fully flexed decrease the friction
        Tf[KNEE_T] = Tf[KNEE_T] - 0.2;
    }


// USE ONLY STATIC FRICTION VALUES


    // Compute hip stiffness

    torque_slope[2] = HIP_STIFFNESS_SLOPE_SLOPE * velocities[2] + HIP_STIFFNESS_SLOPE_OFFSET;

/*    if(velocities[2] > 0){ // eq. for positive velocities
        torque_offset[2] = HIP_STIFFNESS_OFFSET_SLOPE_POS_VEL * velocities[2] + HIP_STIFFNESS_OFFSET_OFFSET_POS_VEL;
    }else{ // for negative or zero velocities
        torque_offset[2] = HIP_STIFFNESS_OFFSET_SLOPE_NEG_VEL * velocities[2] + HIP_STIFFNESS_OFFSET_OFFSET_NEG_VEL;

    }
*/
    torque_offset[2] = HIP_STIFFNESS_OFFSET_OFFSET_NEG_VEL;

  //  Tf[2] = (torque_slope[2] * angles[2] + torque_offset[2]); // hip friction

//    Tf[2] = 1.9828* angles[2]* angles[2]* angles[2] +1.1377 * angles[2]* angles[2] +0.3567* angles[2] +1.7427;

    Tf[HIP_T] = 0.9879* angles[2]* angles[2]* angles[2] -0.796 * angles[2]* angles[2] +0.362 * angles[2] + 2.2884;
//    if(legSide==0){ //left leg
//        Tf[ANKLE_T] = sysProperties->jointControl[LANKLE].manualTorque;
//        Tf[KNEE_T]  = sysProperties->jointControl[LKNEE].manualTorque;
//        Tf[HIP_T]   = sysProperties->jointControl[LHIP].manualTorque;
//    }else{ //right leg
//        Tf[ANKLE_T] = sysProperties->jointControl[RANKLE].manualTorque;
//        Tf[KNEE_T]  = sysProperties->jointControl[RKNEE].manualTorque;
//        Tf[HIP_T]   = sysProperties->jointControl[RHIP].manualTorque;
}
```

Appendix A.14 – SSup.h

```
/* Function: SingleSupportTHM
* --------------------------------
* Calculates the joint torques due to the human during single support and updates THM.
* does not use the backpack force sensor.
* The torque vectors are as follows: - [TankleL TkneeL ThipL TankleR TkneeR ThipR]
*/
void SingleSupportTHM(const double           *angles,
                      const double           *otherLegAngles,
                      const double           *accelerations,
                      const double           *torques,
                      const BodyDataT        *bodyData,
                      const int               groundedLeg,
                      const int               leftHeelContact,
                      const int               rightHeelContact,
                      ForceDistributionT     *distrData,
                      SensorDataT            *sensorData,
                      const SysPropertiesT   *sysProperties);


/* Function: GetJTsensorInTorsoFrame
* ---------------------------------------------------------------
* Calculates the transpose jacobian matrix of the backpack force sensor for a 3dof leg and updates JT.
*/
void GetJTsensorInTorsoFrame(double               JT[][3],
                             const double         kneeAngle,
                             const double         hipAngle,
                             const BodyDataT     *bodyData);


/* Function: GetTfrictionSStance
* ------------------------------------------------
* Calculates the joint torques vector to counteract joint friction and stiffness for one leg. Updates Tf.
* The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
* Equations are obtained from Excel documents 'rankle stiffness.xls', 'rknee stiffness.xls', 'rhip stiffness.xls'.
* legSide =0 for left leg and 1 for right leg
*/
void GetTfrictionSStance(double              *Tf,
                         const double        *angles,
                         const double        *velocities,
                         const double        *momentArms,
                         const SysPropertiesT *sysProperties,
                         const char           legSide);
```

Appendix A.15 – SSup.c

```
#include <math.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"
#include "SSup.h"
#include "DSup.h"
#include "Jump.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];

extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];


/* Function: SingleSupportTHM
 * --------------------------------------------------------------------------------------------
 * Calculates the joint torques due to the human during single support and updates THM.
 * does not use the backpack force sensor.
 * The torque vectors are as follows: - [TankleL TkneeL ThipL TankleR TkneeR ThipR]
 */
void SingleSupportTHM(const double           *angles,
                      const double           *velocities,
                      const double           *accelerations,
                      const double           *torques,
                      const BodyDataT        *bodyData,
                      const int               groundedLeg,
                      const int               leftHeelContact,
                      const int               rightHeelContact,
                      ForceDistributionT     *distrData,
                      SensorDataT            *sensorData,
                      const SysPropertiesT    *sysProperties){

    double mf,If,Lf,LGf,hGf,ms,Is,Ls,LGs,hGs, mt,It,Lt,LGt,hGt,mub,Iub,LGub, hGub,LGsp, LGtp;// segment properties
    double T2,T3,T4,T5,T6,T7,Tke[6]; // local variable for joint torques in the form [TGroundedAnkle  .... TSwingAnkle]
                                     // these torques are due to inertial components of the dynamic equations
    double
q1,q2,q3,q4,q5,q6,q7,dq1,dq2,dq3,dq4,dq5,dq6,dq7,dq12,dq123,dq1234,dq12345,dq123456,dq1234567,ddq1,ddq2,ddq3,ddq4,ddq5,ddq6
,ddq7;
    double c1,s1,c12,c123,c1234,c12345,c123456,c1234567,s12,s123,s1234,s12345,s123456,s1234567;
    double p1,p2,p3,p4,p5,p6,p7,p8,p9,p10,p11,p12,p13,p14,p15;
    double Tg2, Tg3, Tg4, Tg5, Tg6, Tg7, Tcc2, Tcc3, Tcc4, Tcc5, Tcc6, Tcc7; // gravity and velocity torques;
    double Tf[6]; // torque to resist joint stiffness and friction
    int i;
    double momentArms[3] = {0,0,0};

    mf   = bodyData->foot.mass;
    If   = bodyData->foot.inertia;
    Lf   = bodyData->foot.length;
    LGf  = bodyData->foot.Lcg;
    hGf  = bodyData->foot.hcg;
    ms   = bodyData->shank.mass;
    Is   = bodyData->shank.inertia;
    Ls   = bodyData->shank.length;
    LGs  = bodyData->shank.Lcg;
    LGsp = LGs - Ls;
    hGs  = bodyData->shank.hcg;
     mt   = bodyData->thigh.mass;
    It   = bodyData->thigh.inertia;
    Lt   = bodyData->thigh.length;
    LGt  = bodyData->thigh.Lcg;
    LGtp = LGt - Lt;
    hGt  = bodyData->thigh.hcg;
    mub  = bodyData->upperBody.mass;
    LGub = bodyData->upperBody.Lcg;
    hGub = bodyData->upperBody.hcg;
    Iub  = bodyData->upperBody.inertia;

    if (groundedLeg == LEFT){ // Adjust angles to match with local sign convention [q1 ...q7] = [grounded toe ... swing
toe]
        q1   =  angles[LTOE] - ANKLE_TOE_HEEL_ANGLE;
        q2   =  angles[LANKLE] + ANKLE_TOE_HEEL_ANGLE;
        q3   =  angles[LKNEE];
        q4   =  angles[LHIP];
        q5   = -angles[RHIP];
        q6   = -angles[RKNEE];
        q7   = -angles[RANKLE];
        dq1  =  velocities[LTOE];
        dq2  =  velocities[LANKLE];
```

```
        dq3  =  velocities[LKNEE];
        dq4  =  velocities[LHIP];
        dq5  = -velocities[RHIP];
        dq6  = -velocities[RKNEE];
        dq7  = -velocities[RANKLE];
        ddq1 =  accelerations[LTOE];
        ddq2 =  accelerations[LANKLE];
        ddq3 =  accelerations[LKNEE];
        ddq4 =  accelerations[LHIP];
        ddq5 = -accelerations[RHIP];
        ddq6 = -accelerations[RKNEE];
        ddq7 = -accelerations[RANKLE];

        for (i=0;i<4;i++){
            distrData->unfilteredBetaFg[i]  = 1; // reset beta parameters used in double stance
            distrData->filteredBetaFg[i]    = 1;
            distrData->unfilteredBetaFHM[i] = 1;
            distrData->filteredBetaFHM[i]   = 1;
        }

    } else{ // Right leg is grounded
        q1   =  angles[RTOE] - ANKLE_TOE_HEEL_ANGLE;
        q2   =  angles[RANKLE] + ANKLE_TOE_HEEL_ANGLE;
        q3   =  angles[RKNEE];
        q4   =  angles[RHIP];
        q5   = -angles[LHIP];
        q6   = -angles[LKNEE];
        q7   = -angles[LANKLE];
        dq1  =  velocities[RTOE];
        dq2  =  velocities[RANKLE];
        dq3  =  velocities[RKNEE];
        dq4  =  velocities[RHIP];
        dq5  = -velocities[LHIP];
        dq6  = -velocities[LKNEE];
        dq7  = -velocities[LANKLE];
        ddq1 =  accelerations[RTOE];
        ddq2 =  accelerations[RANKLE];
        ddq3 =  accelerations[RKNEE];
        ddq4 =  accelerations[RHIP];
        ddq5 = -accelerations[LHIP];
        ddq6 = -accelerations[LKNEE];
        ddq7 = -accelerations[LANKLE];

        for (i=4; i; i--){
            distrData->unfilteredBetaFg[i-1]  = 0; // reset beta parameters used in double stance
            distrData->filteredBetaFg[i-1]    = 0;
            distrData->unfilteredBetaFHM[i-1] = 0;
            distrData->filteredBetaFHM[i-1]   = 0;
        }

    }


    // make acc. signals for grounded leg = 0
    ddq1=0; ddq2=0; ddq3=0; ddq4=0; // added 10/13/2003 by JRS, NOTE: Thm=0 for grounded leg b/c Kp = 0 for grounded leg,
see FHM.c!

    // make vel. signals for grounded leg = 0;
    dq1=0;  dq2=0;  dq3=0;  dq4=0; // added 10/13/2003, by JRS

    for (i=4; i; i--){
        distrData->unfilteredKrot[i-1] = 1;
        distrData->filteredKrot[i-1] = 1;
    }

    if (((groundedLeg == LEFT) && leftHeelContact) || ((groundedLeg == RIGHT) && rightHeelContact)){ //Adjust foot segment
length
        Lf = bodyData->heel.length;
           //LGf = bodyData->heel.Lcg;   // for heel contact the foot length is from the ankle to the heel
        //hGf = bodyData->heel.hcg;
        q1 = q1 - HEEL_ANKLE_TOE_ANGLE;
        q2 = q2 + HEEL_ANKLE_TOE_ANGLE;
    }

    c1       = cos(q1);                              s1       = sin(q1);
    c12      = cos(q1+q2);                           s12      = sin(q1+q2);
    c123     = cos(q1+q2+q3);                        s123     = sin(q1+q2+q3);
    c1234    = cos(q1+q2+q3+q4);                     s1234    = sin(q1+q2+q3+q4);
    c12345   = cos(q1+q2+q3+q4+q5);                  s12345   = sin(q1+q2+q3+q4+q5);
    c123456  = cos(q1+q2+q3+q4+q5+q6);               s123456  = sin(q1+q2+q3+q4+q5+q6);
    c1234567 = cos(q1+q2+q3+q4+q5+q6+q7);            s1234567 = sin(q1+q2+q3+q4+q5+q6+q7);

    dq12      = dq1 + dq2;
    dq123     = dq12 + dq3;
    dq1234    = dq123 + dq4;
    dq12345   = dq1234 + dq5;
    dq123456  = dq12345 + dq6;
    dq1234567 = dq123456 + dq7;

    // Use Tg later for feedback linearization

    Tg2 = -ms*g*(-LGs*(s12)+hGs*(c12))-mt*g*(-Ls*(s12)-LGt*(s123)+hGt*(c123))-mub*g*(-Ls*(s12)-Lt*(s123)-
LGub*(s1234)+hGub*(c1234))-mt*g*(-Ls*(s12)-Lt*(s123)-(LGt
        -Lt)*(s12345)+hGt*(c12345))-ms*g*(-Ls*(s12)-Lt*(s123)+Lt*(s12345)-LGsp*(s123456)+hGs*(c123456))-mf*g*(-Ls*(s12)-
Lt*(s123)+Lt*(s12345)+Ls*(s123456)+LGf
          *(c1234567)-hGf*(s1234567)); // SSupV2.txt

    Tg3 = -mt*g*(-LGt*(s123)+hGt*(c123))-mub*g*(-Lt*(s123)-LGub*(s1234)+hGub*(c1234))-mt*g*(-Lt*(s123)-
LGtp*(s12345)+hGt*(c12345))-ms*g*(-Lt*(s123)+Lt*(s12345)
```

```
        -LGsp*(s123456)+hGs*(c123456))-mf*g*(-Lt*(s123)+Lt*(s12345)+Ls*(s123456)+LGf*(c1234567)-hGf*(s1234567)); //
SSupV3.txt

    Tg4 = -mub*g*(-LGub*(s1234)+hGub*(c1234))-mt*g*(-LGtp*(s12345)+hGt*(c12345))-ms*g*(Lt*(s12345)-
LGsp*(s123456)+hGs*(c123456))-mf*g*(Lt*(s12345)+Ls*(s123456)
        +LGf*(c1234567)-hGf*(s1234567)); // SSupV4.txt

    Tg5 = -mt*g*(-LGtp*(s12345)+hGt*(c12345))-ms*g*(Lt*(s12345)-LGsp*(s123456)+hGs*(c123456))-
mf*g*(Lt*(s12345)+Ls*(s123456)+LGf*(c1234567)-hGf*(s1234567)); // SSupV5.txt

    Tg6 = -ms*g*(-LGsp*(s123456)+hGs*(c123456))-mf*g*(Ls*(s123456)+LGf*(c1234567)-hGf*(s1234567)); // SSupV6.txt

    Tg7 = -mf*g*(LGf*(c1234567)-hGf*(s1234567)); // SSupV7.txt

    //compute stance leg human-induced joint torques (THM)
    // Run SevenLinkSwing2dof.m in MATLAB to create these equation

    // SSupKE2Subs.txt
    p1  = -Ls*s12-LGt*s123+hGt*c123;
    p2  = -LGt*s123+hGt*c123;
    p3  = -Ls*c12-LGt*c123-hGt*s123;
    p4  = -(LGs-Ls)*s123456+hGs*c123456;
    p5  = -Ls*s12-Lt*s123+Lt*s12345-(LGs-Ls)*s123456+hGs*c123456;
    p6  = -Ls*c12-Lt*c123+Lt*c12345-(LGs-Ls)*c123456-hGs*s123456;
    p7  = -Ls*s12-Lt*s123-LGub*s1234+hGub*c1234;
    p8  = -LGub*s1234+hGub*c1234;
    p9  = -Ls*c12-Lt*c123-LGub*c1234-hGub*s1234;
    p10 = -Ls*s12-Lt*s123+Lt*s12345+Ls*s123456+LGf*c1234567-hGf*s1234567;
    p11 =  LGf*c1234567-hGf*s1234567;
    p12 = -Ls*c12-Lt*c123+Lt*c12345+Ls*c123456-LGf*s1234567-hGf*c1234567;
    p13 = -Ls*s12-Lt*s123-(LGt-Lt)*s12345+hGt*c12345;
    p14 = -(LGt-Lt)*s12345+hGt*c12345;
    p15 = -Ls*c12-Lt*c123-(LGt-Lt)*c12345-hGt*s12345;

    // SSupKE2.txt
    Tcc2 = .5*ms*(2*(-dq12*(-LGs*s12*dq12+hGs*c12*dq12)+dq1*dq1*Lf*c1)*(-LGs*c12-hGs*s12)+2*(-dq1*Lf*c1+dq12
        *(-LGs*s12+hGs*c12))*(-LGs*c12*dq12-hGs*s12*dq12)+2*(dq1*Lf*s1-dq12*(LGs*c12+hGs*s12))*(LGs*s12*dq12-
hGs*c12*dq12)+2*(dq12*(-LGs*c12*dq12-hGs*s12*dq12)
        +dq1*dq1*Lf*s1)*(-LGs*s12+hGs*c12))-.5*mf*(2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-dq1234567*(LGf*s1234567+hGf*c1234567))
        *(Ls*s12*dq12+Lt*s123*dq123-Lt*s12345*dq12345-Ls*s123456*dq123456-dq1234567*p11)+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+Lt*s12345*dq12345+Ls*s123456*dq123456
        +dq1234567*p11)*(-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456+dq1234567*(-LGf*s1234567-
hGf*c1234567)))-.5*mt*(2*(dq1*Lf*s1-dq12*Ls*c12
        -dq123*Lt*c123-dq12345*((LGt-Lt)*c12345+hGt*s12345))*(Ls*s12*dq12+Lt*s123*dq123-dq12345*p14)+2*(-dq1*Lf*c1-
Ls*s12*dq12-Lt*s123*dq123+dq12345*p14)*(-dq12
        *Ls*c12-dq123*Lt*c123+dq12345*(-(LGt-Lt)*c12345-hGt*s12345)))-.5*mub*(2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123-
dq1234*(LGub*c1234+hGub*s1234))*(Ls*s12*dq12
        +Lt*s123*dq123-dq1234*p8)+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+dq1234*p8)*(-dq12*Ls*c12-dq123*Lt*c123+dq1234*(-
LGub*c1234-hGub*s1234)))+.5*mt*(2*(dq12*dq12
        *Ls*s12+dq1*dq1*Lf*c1-dq12345*(-(LGt-Lt)*s12345*dq12345+hGt*c12345*dq12345)+dq123*dq123*Lt*s123)*p15+2*(-dq1*Lf*c1-
Ls*s12*dq12-Lt*s123*dq123+dq12345
        *p14)*(-dq12*Ls*c12-dq123*Lt*c123-(LGt-Lt)*c12345*dq12345-hGt*s12345*dq12345)+2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123-dq12345*((LGt-Lt)*c12345+hGt*s12345))
        *(Ls*s12*dq12+Lt*s123*dq123+(LGt-Lt)*s12345*dq12345-hGt*c12345*dq12345)+2*(-
dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq12345*(-(LGt-Lt)*c12345*dq12345-hGt*s12345
        *dq12345)-dq123*dq123*Lt*c123)*p13)-.5*mt*(2*(dq1*Lf*s1-dq12*Ls*c12-dq123*(LGt*c123+hGt*s123))*(Ls*s12*dq12-
dq123*p2)+2*(-dq1*Lf*c1-Ls*s12*dq12+dq123*p2)
        *(-dq12*Ls*c12+dq123*(-LGt*c123-hGt*s123)))+.5*mf*(2*(-dq123456*dq123456*Ls*s123456+dq1*dq1*Lf*c1-
dq1234567*(LGf*c1234567*dq1234567-hGf*s1234567*dq1234567)
        +dq12*dq12*Ls*s12-dq12345*dq12345*Lt*s12345+dq123*dq123*Lt*s123)*p12+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567
        *p11)*(-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-LGf*s1234567*dq1234567-
hGf*c1234567*dq1234567)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt
        *c123+dq12345*Lt*c12345+dq123456*Ls*c123456-dq1234567*(LGf*s1234567+hGf*c1234567))*(Ls*s12*dq12+Lt*s123*dq123-
Lt*s12345*dq12345-Ls*s123456*dq123456-LGf
        *c1234567*dq1234567+hGf*s1234567*dq1234567)+2*(dq123456*dq123456*Ls*c123456+dq1*dq1*Lf*s1+dq1234567*(-
LGf*s1234567*dq1234567-hGf*c1234567*dq1234567)-dq12*dq12
        *Ls*c12+dq12345*dq12345*Lt*c12345-dq123*dq123*Lt*c123)*p10)+.5*mub*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-dq1234*(-
LGub*s1234*dq1234+hGub*c1234*dq1234)
        +dq123*dq123*Lt*s123)*p9+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+dq1234*p8)*(-dq12*Ls*c12-dq123*Lt*c123-
LGub*c1234*dq1234-hGub*s1234*dq1234)+2*(dq1*Lf
        *s1-dq12*Ls*c12-dq123*Lt*c123-dq1234*(LGub*c1234+hGub*s1234))*(Ls*s12*dq12+Lt*s123*dq123+LGub*s1234*dq1234-
hGub*c1234*dq1234)+2*(-dq12*dq12*Ls*c12+dq1*dq1
        *Lf*s1+dq1234*(-LGub*c1234*dq1234-hGub*s1234*dq1234)-dq123*dq123*Lt*c123)*p7)+.5*ms*(2*(-dq123456*(-(LGs-
Ls)*s123456*dq123456+hGs*c123456*dq123456)+dq1*dq1
        *Lf*c1+dq123*dq123*Lt*s123-dq12345*dq12345*Lt*s12345+dq12*dq12*Ls*s12)*p6+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+Lt*s12345*dq12345+dq123456*p4)*(-dq12
        *Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345-(LGs-Ls)*c123456*dq123456-hGs*s123456*dq123456)+2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123+dq12345*Lt*c12345-dq123456
        *((LGs-Ls)*c123456+hGs*s123456))*(Ls*s12*dq12+Lt*s123*dq123-Lt*s12345*dq12345+(LGs-Ls)*s123456*dq123456-
hGs*c123456*dq123456)+2*(dq123456*(-(LGs-Ls)*c123456
        *dq123456-hGs*s123456*dq123456)+dq1*dq1*Lf*s1-dq123*dq123*Lt*c123+dq12345*dq12345*Lt*c12345-dq12*dq12*Ls*c12)*p5)-
.5*ms*(2*(dq1*Lf*s1-dq12*Ls*c12-dq123
        *Lt*c123+dq12345*Lt*c12345-dq123456*((LGs-Ls)*c123456+hGs*s123456))*(Ls*s12*dq12+Lt*s123*dq123-Lt*s12345*dq12345-
dq123456*p4)+2*(-dq1*Lf*c1-Ls*s12*dq12
        -Lt*s123*dq123+Lt*s12345*dq12345+dq123456*p4)*(-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345+dq123456*(-(LGs-
Ls)*c123456-hGs*s123456)))+.5*mt*(2*(dq12*dq12
        *Ls*s12+dq1*dq1*Lf*c1-dq123*(-LGt*s123*dq123+hGt*c123*dq123))*p3+2*(-dq1*Lf*c1-Ls*s12*dq12+dq123*p2)*(-dq12*Ls*c12-
LGt*c123*dq123-hGt*s123*dq123)+2*(dq1
        *Lf*s1-dq12*Ls*c12-dq123*(LGt*c123+hGt*s123))*(Ls*s12*dq12+LGt*s123*dq123-hGt*c123*dq123)+2*(-
dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq123*(-LGt*c123*dq123
        -hGt*s123*dq123))*p1)-.5*ms*(-2*(dq1*Lf*s1-dq12*(LGs*c12+hGs*s12))*dq12*(-LGs*s12+hGs*c12)+2*(-dq1*Lf*c1+dq12*(-
LGs*s12+hGs*c12))*dq12*(-LGs*c12-hGs*s12));


    T2 = (.5*mf*(2*(-LGf*s1234567-hGf*c1234567)*p12+2*p11*p10)+If)*ddq7+(.5*ms*(2*(-(LGs-Ls)*c123456-
hGs*s123456)*p6+2*p4*p5)+If+.5*mf*(2*(-LGf*s1234567-hGf*c1234567
```

```
        +Ls*c123456)*p12+2*(LGf*c1234567-hGf*s1234567+Ls*s123456)*p10)+Is)*ddq6+(.5*mf*(2*(Lt*c12345-LGf*s1234567-
hGf*c1234567+Ls*c123456)*p12+2*(Lt*s12345+LGf
        *c1234567-hGf*s1234567+Ls*s123456)*p10)+If+.5*ms*(2*(Lt*c12345-(LGs-Ls)*c123456-hGs*s123456)*p6+2*(Lt*s12345-(LGs-
Ls)*s123456+hGs*c123456)*p5)+Is+.5*mt
        *(2*(-(LGt-Lt)*c12345-hGt*s12345)*p15+2*p14*p13)+It)*ddq5+(Iub+.5*mub*(2*(-LGub*c1234-
hGub*s1234)*p9+2*p8*p7)+.5*ms*(2*(Lt*c12345-(LGs-Ls)*c123456-hGs*s123456)
        *p6+2*(Lt*s12345-(LGs-Ls)*s123456+hGs*c123456)*p5)+If+.5*mf*(2*(Lt*c12345-LGf*s1234567-
hGf*c1234567+Ls*c123456)*p12+2*(Lt*s12345+LGf*c1234567-hGf*s1234567
        +Ls*s123456)*p10)+It+Is+.5*mt*(2*(-(LGt-Lt)*c12345-hGt*s12345)*p15+2*p14*p13))*ddq4+(2*It+.5*ms*(2*(-
Lt*c123+Lt*c12345-(LGs-Ls)*c123456-hGs*s123456)*p6
        +2*(-Lt*s123+Lt*s12345-(LGs-Ls)*s123456+hGs*c123456)*p5)+.5*mt*(2*(-LGt*c123-
hGt*s123)*p3+2*p2*p1)+Is+If+.5*mt*(2*(-(LGt-Lt)*c12345-hGt*s12345-Lt*c123)
        *p15+2*(-(LGt-Lt)*s12345+hGt*c12345-Lt*s123)*p13)+.5*mf*(2*(-Lt*c123+Lt*c12345-LGf*s1234567-
hGf*c1234567+Ls*c123456)*p12+2*(-Lt*s123+Lt*s12345+LGf*c1234567
        -hGf*s1234567+Ls*s123456)*p10)+.5*mub*(2*(-LGub*c1234-hGub*s1234-Lt*c123)*p9+2*(-LGub*s1234+hGub*c1234-
Lt*s123)*p7)+Iub)*ddq3+(2*It+2*Is+.5*ms*(2*p6*p6
        +2*p5*p5)+.5*mt*(2*p3*p3+2*p1*p1)+.5*ms*(2*(-LGs*c12-hGs*s12)*(-LGs*c12-hGs*s12)+2*(-LGs*s12+hGs*c12)*(-
LGs*s12+hGs*c12))+.5*mt*(2*p15*p15+2*p13*p13)+.5*mub*(2
        *p9*p9+2*p7*p7)+If+.5*mf*(2*p12*p12+2*p10*p10)+Iub)*ddq2+(.5*mt*(2*(-Lf*c1-LGt*s123+hGt*c123-Ls*s12)*p1+2*(Lf*s1-
LGt*c123-hGt*s123-Ls*c12)
        *p3)+2*Is+2*It+.5*ms*(2*(Lt*s12345-Lt*s123-Ls*s12-(LGs-Ls)*s123456+hGs*c123456-Lf*c1)*p5+2*(Lt*c12345-Lt*c123-
Ls*c12-(LGs-Ls)*c123456-hGs*s123456+Lf*s1)
        *p6)+.5*mub*(2*(-Ls*s12-Lf*c1-LGub*s1234+hGub*c1234-Lt*s123)*p7+2*(-Ls*c12+Lf*s1-LGub*c1234-hGub*s1234-
Lt*c123)*p9)+If+Iub+.5*mf*(2*(Lt*s12345+Ls*s123456
        -Lf*c1-Lt*s123+LGf*c1234567-hGf*s1234567-Ls*s12)*p10+2*(Lt*c12345+Ls*c123456+Lf*s1-Lt*c123-LGf*s1234567-
hGf*c1234567-Ls*c12)*p12)+.5*mt*(2*(-Ls*s12-Lf*c1
        -(LGt-Lt)*s12345+hGt*c12345-Lt*s123)*p13+2*(-Ls*c12+Lf*s1-(LGt-Lt)*c12345-hGt*s12345-Lt*c123)*p15)+.5*ms*(2*(-
LGs*s12+hGs*c12-Lf*c1)*(-LGs*s12+hGs*c12)
        +2*(-LGs*c12-hGs*s12+Lf*s1)*(-LGs*c12-hGs*s12)))*ddq1;

    // SSupKE3Subs.txt
    p1  = -Lt*s123+Lt*s12345-(LGs-Ls)*s123456+hGs*c123456;
    p2  = -(LGs-Ls)*s123456+hGs*c123456;
    p3  = -Lt*c123+Lt*c12345-(LGs-Ls)*c123456-hGs*s123456;
    p4  = -Lt*s123+Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456;
    p5  =  LGf*c1234567-hGf*s1234567;
    p6  = -Lt*c123+Lt*c12345-LGf*s1234567-hGf*c1234567+Ls*c123456;
    p7  = -(LGt-Lt)*s12345+hGt*c12345-Lt*s123;
    p8  = -(LGt-Lt)*s12345+hGt*c12345;
    p9  = -(LGt-Lt)*c12345-hGt*s12345-Lt*c123;
    p10 = -LGub*s1234+hGub*c1234-Lt*s123;
    p11 = -LGub*s1234+hGub*c1234;
    p12 = -LGub*c1234-hGub*s1234-Lt*c123;
    p13 = -LGt*c123-hGt*s123;
    p14 = -LGt*s123+hGt*c123;

    // SSupKE3.txt
    Tcc3 = .5*mt*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1
        -dq123*(-LGt*s123*dq123+hGt*c123*dq123))*p13+2*(-dq1*Lf*c1-Ls*s12*dq12+dq123*p14)*(-LGt*c123*dq123-
hGt*s123*dq123)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*(LGt*c123
        +hGt*s123))*(LGt*s123*dq123-hGt*c123*dq123)+2*(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq123*(-LGt*c123*dq123-
hGt*s123*dq123))*p14)-.5*mt*(2*(dq1*Lf*s1-dq12
        *Ls*c12-dq123*Lt*c123-dq12345*((LGt-Lt)*c12345+hGt*s12345))*(Lt*s123*dq123-dq12345*p8)+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+dq12345*p8)*(-dq123*Lt*c123
        +dq12345*(-(LGt-Lt)*c12345-hGt*s12345)))-.5*mt*(-2*(dq1*Lf*s1-dq12*Ls*c12-dq123*(LGt*c123+hGt*s123))*dq123*p14+2*(-
dq1*Lf*c1-Ls*s12*dq12+dq123*p14)*dq123
        *p13)-.5*ms*(2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345-dq123456*((LGs-
Ls)*c123456+hGs*s123456))*(Lt*s123*dq123-Lt*s12345*dq12345-dq123456
        *p2)+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+Lt*s12345*dq12345+dq123456*p2)*(-
dq123*Lt*c123+dq12345*Lt*c12345+dq123456*(-(LGs-Ls)*c123456-hGs*s123456)))
        -.5*mub*(2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123-dq1234*(LGub*c1234+hGub*s1234))*(Lt*s123*dq123-dq1234*p11)+2*(-
dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+dq1234
        *p11)*(-dq123*Lt*c123+dq1234*(-LGub*c1234-hGub*s1234)))+.5*mub*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-dq1234*(-
LGub*s1234*dq1234+hGub*c1234*dq1234)+dq123*dq123
        *Lt*s123)*p12+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+dq1234*p11)*(-dq123*Lt*c123-LGub*c1234*dq1234-
hGub*s1234*dq1234)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt
        *c123-dq1234*(LGub*c1234+hGub*s1234))*(Lt*s123*dq123+LGub*s1234*dq1234-hGub*c1234*dq1234)+2*(-
dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq1234*(-LGub*c1234*dq1234
        -hGub*s1234*dq1234)-dq123*dq123*Lt*c123)*p10)+.5*mt*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-dq12345*(-(LGt-
Lt)*s12345*dq12345+hGt*c12345*dq12345)+dq123*dq123
        *Lt*s123)*p9+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+dq12345*p8)*(-dq123*Lt*c123-(LGt-Lt)*c12345*dq12345-
hGt*s12345*dq12345)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123
        *Lt*c123-dq12345*((LGt-Lt)*c12345+hGt*s12345))*(Lt*s123*dq123+(LGt-Lt)*s12345*dq12345-hGt*c12345*dq12345)+2*(-
dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq12345
        *(-(LGt-Lt)*c12345*dq12345-hGt*s12345*dq12345)-dq123*dq123*Lt*c123)*p7)-.5*mf*(2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456
        -dq1234567*(LGf*s1234567+hGf*c1234567))*(Lt*s123*dq123-Lt*s12345*dq12345-Ls*s123456*dq123456-dq1234567*p5)+2*(-
dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+Lt*s12345
        *dq12345+Ls*s123456*dq123456+dq1234567*p5)*(-dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456+dq1234567*(-
LGf*s1234567-hGf*c1234567)))+.5*mf*(2*(-dq123456*dq123456
        *Ls*s123456+dq1*dq1*Lf*c1-dq1234567*(LGf*c1234567*dq1234567-hGf*s1234567*dq1234567)+dq12*dq12*Ls*s12-
dq12345*dq12345*Lt*s12345+dq123*dq123*Lt*s123)
        *p6+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567*p5)*(-
dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-LGf
        *s1234567*dq1234567-hGf*c1234567*dq1234567)+2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-dq1234567*(LGf*s1234567+hGf*c1234567))
        *(Lt*s123*dq123-Lt*s12345*dq12345-Ls*s123456*dq123456-
LGf*c1234567*dq1234567+hGf*s1234567*dq1234567)+2*(dq123456*dq123456*Ls*c123456+dq1*dq1*Lf*s1+dq1234567
        *(-LGf*s1234567*dq1234567-hGf*c1234567*dq1234567)-dq12*dq12*Ls*c12+dq12345*dq12345*Lt*c12345-
dq123*dq123*Lt*c123)*p4)+.5*ms*(2*(-dq123456*(-(LGs-Ls)*s123456
        *dq123456+hGs*c123456*dq123456)+dq1*dq1*Lf*c1+dq123*dq123*Lt*s123-
dq12345*dq12345*Lt*s12345+dq12*dq12*Ls*s12)*p3+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123
        *dq123+Lt*s12345*dq12345+dq123456*p2)*(-dq123*Lt*c123+dq12345*Lt*c12345-(LGs-Ls)*c123456*dq123456-
hGs*s123456*dq123456)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt
        *c123+dq12345*Lt*c12345-dq123456*((LGs-Ls)*c123456+hGs*s123456))*(Lt*s123*dq123-Lt*s12345*dq12345+(LGs-
Ls)*s123456*dq123456-hGs*c123456*dq123456)+2*(dq123456
        *(-(LGs-Ls)*c123456*dq123456-hGs*s123456*dq123456)+dq1*dq1*Lf*s1-dq123*dq123*Lt*c123+dq12345*dq12345*Lt*c12345-
dq12*dq12*Ls*c12)*p1);
```

```
    T3 = (.5*mf*(2*(-LGf*s1234567-hGf*c1234567)*p6+2*p5*p4)+If)*ddq7+(Is+.5*ms*(2*(-(LGs-Ls)*c123456-
hGs*s123456)*p3+2*p2*p1)+If+.5*mf*(2*(-LGf*s1234567-hGf*c1234567
        +Ls*c123456)*p6+2*(LGf*c1234567-hGf*s1234567+Ls*s123456)*p4))*ddq6+(It+If+.5*ms*(2*(Lt*c12345-(LGs-Ls)*c123456-
hGs*s123456)*p3+2*(Lt*s12345-(LGs-Ls)*s123456
        +hGs*c123456)*p1)+.5*mt*(2*(-(LGt-Lt)*c12345-hGt*s12345)*p9+2*p8*p7)+.5*mf*(2*(Lt*c12345-LGf*s1234567-
hGf*c1234567+Ls*c123456)*p6+2*(Lt*s12345+LGf*c1234567
        -hGf*s1234567+Ls*s123456)*p4)+Is)*ddq5+(It+Iub+.5*mf*(2*(Lt*c12345-LGf*s1234567-
hGf*c1234567+Ls*c123456)*p6+2*(Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456)
        *p4)+.5*ms*(2*(Lt*c12345-(LGs-Ls)*c123456-hGs*s123456)*p3+2*(Lt*s12345-(LGs-
Ls)*s123456+hGs*c123456)*p1)+Is+.5*mub*(2*(-LGub*c1234-hGub*s1234)*p12+2*p11
        *p10)+.5*mt*(2*(-(LGt-Lt)*c12345-
hGt*s12345)*p9+2*p8*p7)+If)*ddq4+(.5*mub*(2*p12*p12+2*p10*p10)+.5*mt*(2*p13*p13+2*p14*p14)+If+Is+.5*mt*(2*p9*p9
        +2*p7*p7)+2*It+Iub+.5*ms*(2*p3*p3+2*p1*p1)+.5*mf*(2*p6*p6+2*p4*p4))*ddq3+(2*It+.5*ms*(2*p3*(-Ls*c12-
Lt*c123+Lt*c12345-(LGs-Ls)*c123456
        -hGs*s123456)+2*p1*(-Ls*s12-Lt*s123+Lt*s12345-(LGs-Ls)*s123456+hGs*c123456))+.5*mt*(2*p13*(-Ls*c12-LGt*c123-
hGt*s123)+2*p14*(-Ls*s12-LGt*s123+hGt*c123))
        +Is+If+.5*mt*(2*p9*(-Ls*c12-Lt*c123-(LGt-Lt)*c12345-hGt*s12345)+2*p7*(-Ls*s12-Lt*s123-(LGt-
Lt)*s12345+hGt*c12345))+.5*mf*(2*p6*(-Ls*c12-Lt*c123+Lt*c12345
        +Ls*c123456-LGf*s1234567-hGf*c1234567)+2*p4*(-Ls*s12-Lt*s123+Lt*s12345+Ls*s123456+LGf*c1234567-
hGf*s1234567))+.5*mub*(2*p12*(-Ls*c12-Lt*c123-LGub*c1234
        -hGub*s1234)+2*p10*(-Ls*s12-Lt*s123-LGub*s1234+hGub*c1234))+Iub)*ddq2+(.5*mt*(2*(-Lf*c1-LGt*s123+hGt*c123-
Ls*s12)*p14+2*(Lf*s1-LGt*c123-hGt*s123-Ls*c12)
        *p13)+2*It+.5*ms*(2*(Lt*s12345-Lt*s123-Ls*s12-(LGs-Ls)*s123456+hGs*c123456-Lf*c1)*p1+2*(Lt*c12345-Lt*c123-Ls*c12-
(LGs-Ls)*c123456-hGs*s123456+Lf*s1)*p3)
        +.5*mub*(2*(-Ls*s12-Lf*c1-LGub*s1234+hGub*c1234-Lt*s123)*p10+2*(-Ls*c12+Lf*s1-LGub*c1234-hGub*s1234-
Lt*c123)*p12)+Is+Iub+.5*mf*(2*(Lt*s12345+Ls*s123456
        -Lf*c1-Lt*s123+LGf*c1234567-hGf*s1234567-Ls*s12)*p4+2*(Lt*c12345+Ls*c123456+Lf*s1-Lt*c123-LGf*s1234567-
hGf*c1234567-Ls*c12)*p6)+.5*mt*(2*(-Ls*s12-Lf*c1
        -(LGt-Lt)*s12345+hGt*c12345-Lt*s123)*p7+2*(-Ls*c12+Lf*s1-(LGt-Lt)*c12345-hGt*s12345-Lt*c123)*p9)+If)*ddq1;

    // SSupKE4Subs.txt
    p1  = Lt*s12345-(LGs-Ls)*s123456+hGs*c123456;
    p2  = -(LGs-Ls)*s123456+hGs*c123456;
    p3  = Lt*c12345-(LGs-Ls)*c123456-hGs*s123456;
    p4  = -(LGt-Lt)*s12345+hGt*c12345;
    p5  = -(LGt-Lt)*c12345-hGt*s12345;
    p6  = Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456;
    p7  = LGf*c1234567-hGf*s1234567;
    p8  = Lt*c12345-LGf*s1234567-hGf*c1234567+Ls*c123456;
    p9  = -LGub*c1234-hGub*s1234;
    p10 = -LGub*s1234+hGub*c1234;

    // SSupKE4.txt
    Tcc4 = .5*mub*(2
        *(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-dq1234*(-LGub*s1234*dq1234+hGub*c1234*dq1234)+dq123*dq123*Lt*s123)*p9+2*(-
dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+dq1234
        *p10)*(-LGub*c1234*dq1234-hGub*s1234*dq1234)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123-
dq1234*(LGub*c1234+hGub*s1234))*(LGub*s1234*dq1234-hGub*c1234*dq1234)
        +2*(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq1234*(-LGub*c1234*dq1234-hGub*s1234*dq1234)-dq123*dq123*Lt*c123)*p10)-
.5*mub*(-2*(dq1*Lf*s1-dq12*Ls*c12-dq123
        *Lt*c123-dq1234*(LGub*c1234+hGub*s1234))*dq1234*p10+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+dq1234*p10)*dq1234*p9)-
.5*mf*(2*(dq1*Lf*s1-dq12*Ls*c12-dq123
        *Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-dq1234567*(LGf*s1234567+hGf*c1234567))*(-Lt*s12345*dq12345-
Ls*s123456*dq123456-dq1234567*p7)+2*(-dq1*Lf*c1
        -Ls*s12*dq12-
Lt*s123*dq123+Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567*p7)*(dq12345*Lt*c12345+dq123456*Ls*c123456+dq1234567*(-
LGf*s1234567-hGf*c1234567)))
        +.5*mf*(2*(-dq123456*dq123456*Ls*s123456+dq1*dq1*Lf*c1-dq1234567*(LGf*c1234567*dq1234567-
hGf*s1234567*dq1234567)+dq12*dq12*Ls*s12-dq12345*dq12345*Lt*s12345
        +dq123*dq123*Lt*s123)*p8+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567*p7)*(dq12345*Lt*c12345+dq123456*Ls*c123456
        -LGf*s1234567*dq1234567-hGf*c1234567*dq1234567)+2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-dq1234567*(LGf*s1234567+hGf
        *c1234567))*(-Lt*s12345*dq12345-Ls*s123456*dq123456-
LGf*c1234567*dq1234567+hGf*s1234567*dq1234567)+2*(dq123456*dq123456*Ls*c123456+dq1*dq1*Lf*s1+dq1234567
        *(-LGf*s1234567*dq1234567-hGf*c1234567*dq1234567)-dq12*dq12*Ls*c12+dq12345*dq12345*Lt*c12345-
dq123*dq123*Lt*c123)*p6)-.5*mt*(-2*(dq1*Lf*s1-dq12*Ls*c12
        -dq123*Lt*c123-dq12345*((LGt-Lt)*c12345+hGt*s12345))*dq12345*p4+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+dq12345*p4)*dq12345*p5)-.5*ms*(2*(dq1*Lf*s1-dq12
        *Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345-dq123456*((LGs-Ls)*c123456+hGs*s123456))*(-Lt*s12345*dq12345-
dq123456*p2)+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123
        +Lt*s12345*dq12345+dq123456*p2)*(dq12345*Lt*c12345+dq123456*(-(LGs-Ls)*c123456-
hGs*s123456)))+.5*mt*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-dq12345*(-(LGt
        -Lt)*s12345*dq12345+hGt*c12345*dq12345)+dq123*dq123*Lt*s123)*p5+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+dq12345*p4)*(-(LGt-Lt)*c12345*dq12345-hGt*s12345
        *dq12345)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123-dq12345*((LGt-Lt)*c12345+hGt*s12345))*((LGt-Lt)*s12345*dq12345-
hGt*c12345*dq12345)+2*(-dq12*dq12*Ls*c12
        +dq1*dq1*Lf*s1+dq12345*(-(LGt-Lt)*c12345*dq12345-hGt*s12345*dq12345)-dq123*dq123*Lt*c123)*p4)+.5*ms*(2*(-
dq123456*(-(LGs-Ls)*s123456*dq123456+hGs*c123456
        *dq123456)+dq1*dq1*Lf*c1+dq123*dq123*Lt*s123-dq12345*dq12345*Lt*s12345+dq12*dq12*Ls*s12)*p3+2*(-dq1*Lf*c1-
Ls*s12*dq12-Lt*s123*dq123+Lt*s12345*dq12345
        +dq123456*p2)*(dq12345*Lt*c12345-(LGs-Ls)*c123456*dq123456-hGs*s123456*dq123456)+2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123+dq12345*Lt*c12345-dq123456*((LGs
        -Ls)*c123456+hGs*s123456))*(-Lt*s12345*dq12345+(LGs-Ls)*s123456*dq123456-hGs*c123456*dq123456)+2*(dq123456*(-(LGs-
Ls)*c123456*dq123456-hGs*s123456*dq123456)
        +dq1*dq1*Lf*s1-dq123*dq123*Lt*c123+dq12345*dq12345*Lt*c12345-dq12*dq12*Ls*c12)*p1);

    T4 = (If+.5*mf*(2*(-LGf*s1234567-hGf*c1234567)*p8+2*p7*p6))*ddq7+(If+Is+.5*ms*(2*(-(LGs-Ls)*c123456-
hGs*s123456)*p3+2*p2*p1)+.5*mf*(2*(-LGf*s1234567-hGf*c1234567
        +Ls*c123456)*p8+2*(LGf*c1234567-
hGf*s1234567+Ls*s123456)*p6))*ddq6+(.5*mf*(2*p8*p8+2*p6*p6)+.5*mt*(2*p5*p5+2*p4*p4)+If+It+Is+.5*ms*(2*p3*p3

+2*p1*p1))*ddq5+(.5*mub*(2*p9*p9+2*p10*p10)+It+If+.5*ms*(2*p3*p3+2*p1*p1)+Iub+Is+.5*mf*(2*p8*p8+2*p6*p6)+.5*mt*(2*p5*p5+2
        *p4*p4))*ddq4+(It+Iub+.5*mf*(2*p8*(-Lt*c123+Lt*c12345-LGf*s1234567-hGf*c1234567+Ls*c123456)+2*p6*(-
Lt*s123+Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456))
        +.5*ms*(2*p3*(-Lt*c123+Lt*c12345-(LGs-Ls)*c123456-hGs*s123456)+2*p1*(-Lt*s123+Lt*s12345-(LGs-
Ls)*s123456+hGs*c123456))+Is+.5*mub*(2*p9*(-LGub*c1234-hGub
```

```
        *s1234-Lt*c123)+2*p10*(-LGub*s1234+hGub*c1234-Lt*s123))+.5*mt*(2*p5*(-(LGt-Lt)*c12345-hGt*s12345-Lt*c123)+2*p4*(-
(LGt-Lt)*s12345+hGt*c12345-Lt*s123))+If)
        *ddq3+(.5*mt*(2*(-Ls*c12-Lt*c123-(LGt-Lt)*c12345-hGt*s12345)*p5+2*(-Ls*s12-Lt*s123-(LGt-
Lt)*s12345+hGt*c12345)*p4)+.5*mub*(2*(-Ls*c12-Lt*c123-LGub*c1234
        -hGub*s1234)*p9+2*(-Ls*s12-Lt*s123-LGub*s1234+hGub*c1234)*p10)+.5*ms*(2*(-Ls*c12-Lt*c123+Lt*c12345-(LGs-
Ls)*c123456-hGs*s123456)*p3+2*(-Ls*s12-Lt*s123+Lt
        *s12345-(LGs-Ls)*s123456+hGs*c123456)*p1)+It+If+Is+Iub+.5*mf*(2*(-Ls*c12-Lt*c123+Lt*c12345+Ls*c123456-LGf*s1234567-
hGf*c1234567)*p8+2*(-Ls*s12-Lt*s123+Lt
        *s12345+Ls*s123456+LGf*c1234567-hGf*s1234567)*p6))*ddq2+(It+.5*ms*(2*(Lt*s12345-Lt*s123-Ls*s12-(LGs-
Ls)*s123456+hGs*c123456-Lf*c1)*p1+2*(Lt*c12345-Lt*c123
        -Ls*c12-(LGs-Ls)*c123456-hGs*s123456+Lf*s1)*p3)+.5*mub*(2*(-Ls*s12-Lf*c1-LGub*s1234+hGub*c1234-Lt*s123)*p10+2*(-
Ls*c12+Lf*s1-LGub*c1234-hGub*s1234-Lt*c123)
        *p9)+Is+Iub+.5*mf*(2*(Lt*s12345+Ls*s123456-Lf*c1-Lt*s123+LGf*c1234567-hGf*s1234567-
Ls*s12)*p6+2*(Lt*c12345+Ls*c123456+Lf*s1-Lt*c123-LGf*s1234567-hGf*c1234567
        -Ls*c12)*p8)+.5*mt*(2*(-Ls*s12-Lf*c1-(LGt-Lt)*s12345+hGt*c12345-Lt*s123)*p4+2*(-Ls*c12+Lf*s1-(LGt-Lt)*c12345-
hGt*s12345-Lt*c123)*p5)+If)*ddq1;

    // SSupKE5Subs.txt
    p1 =   Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456;
    p2 =   LGf*c1234567-hGf*s1234567;
    p3 =   Lt*c12345-LGf*s1234567-hGf*c1234567+Ls*c123456;
    p4 =   Lt*s12345-(LGs-Ls)*s123456+hGs*c123456;
    p5 = -(LGs-Ls)*s123456+hGs*c123456;
    p6 =   Lt*c12345-(LGs-Ls)*c123456-hGs*s123456;
    p7 = -(LGt-Lt)*c12345-hGt*s12345;
    p8 = -(LGt-Lt)*s12345+hGt*c12345;

    // SSupKE5.txt
    Tcc5 = .5*mt*(2*(dq12*dq12*Ls*s12+dq1*dq1
        *Lf*c1-dq12345*(-(LGt-Lt)*s12345*dq12345+hGt*c12345*dq12345)+dq123*dq123*Lt*s123)*p7+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+dq12345*p8)*(-(LGt-Lt)*c12345
        *dq12345-hGt*s12345*dq12345)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123-dq12345*((LGt-Lt)*c12345+hGt*s12345))*((LGt-
Lt)*s12345*dq12345-hGt*c12345*dq12345)+2
        *(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq12345*(-(LGt-Lt)*c12345*dq12345-hGt*s12345*dq12345)-dq123*dq123*Lt*c123)*p8)-
.5*mt*(-2*(dq1*Lf*s1-dq12*Ls*c12
        -dq123*Lt*c123-dq12345*((LGt-Lt)*c12345+hGt*s12345))*dq12345*p8+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+dq12345*p8)*dq12345*p7)-.5*ms*(2*(dq1*Lf*s1-dq12
        *Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345-dq123456*((LGs-Ls)*c123456+hGs*s123456))*(-Lt*s12345*dq12345-
dq123456*p5)+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123
        +Lt*s12345*dq12345*dq123456*p5)*(dq12345*Lt*c12345+dq123456*(-(LGs-Ls)*c123456-hGs*s123456)))+.5*ms*(2*(-
dq123456*(-(LGs-Ls)*s123456*dq123456+hGs*c123456
        *dq123456)+dq1*dq1*Lf*c1+dq123*dq123*Lt*s123-dq12345*dq12345*Lt*s12345+dq12*dq12*Ls*s12)*p6+2*(-dq1*Lf*c1-
Ls*s12*dq12-Lt*s123*dq123+Lt*s12345*dq12345
        +dq123456*p5)*(dq12345*Lt*c12345-(LGs-Ls)*c123456*dq123456-hGs*s123456*dq123456)+2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123+dq12345*Lt*c12345-dq123456*((LGs
        -Ls)*c123456+hGs*s123456))*(-Lt*s12345*dq12345+(LGs-Ls)*s123456*dq123456-hGs*c123456*dq123456)+2*(dq123456*(-(LGs-
Ls)*c123456*dq123456-hGs*s123456*dq123456)
        +dq1*dq1*Lf*s1-dq123*dq123*Lt*c123+dq12345*dq12345*Lt*c12345-dq12*dq12*Ls*c12)*p4)-.5*mf*(2*(dq1*Lf*s1-dq12*Ls*c12-
dq123*Lt*c123+dq12345*Lt*c12345
        +dq123456*Ls*c123456-dq1234567*(LGf*s1234567+hGf*c1234567))*(-Lt*s12345*dq12345-Ls*s123456*dq123456-
dq1234567*p2)+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123
        +Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567*p2)*(dq12345*Lt*c12345+dq123456*Ls*c123456+dq1234567*(-
LGf*s1234567-hGf*c1234567)))+.5*mf*(2*(-dq123456*dq123456
        *Ls*s123456+dq1*dq1*Lf*c1-dq1234567*(LGf*c1234567*dq1234567-hGf*s1234567*dq1234567)+dq12*dq12*Ls*s12-
dq12345*dq12345*Lt*s12345+dq123*dq123*Lt*s123)
        *p3+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567*p2)*(dq12345*Lt*c12345+dq123456*Ls*c123456-
LGf*s1234567*dq1234567
        -hGf*c1234567*dq1234567)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-
dq1234567*(LGf*s1234567+hGf*c1234567))*(-Lt*s12345
        *dq12345-Ls*s123456*dq123456-
LGf*c1234567*dq1234567+hGf*s1234567*dq1234567)+2*(dq123456*dq123456*Ls*c123456+dq1*dq1*Lf*s1+dq1234567*(-
LGf*s1234567*dq1234567
        -hGf*c1234567*dq1234567)-dq12*dq12*Ls*c12+dq12345*dq12345*Lt*c12345-dq123*dq123*Lt*c123)*p1);

    T5 =(If+.5*mf*(2*(-LGf*s1234567-hGf*c1234567)*p3+2*p2*p1))*ddq7+(If+Is+.5*ms*(2*(-(LGs-Ls)*c123456-
hGs*s123456)*p6+2*p5*p4)+.5*mf*(2*(-LGf*s1234567-hGf*c1234567
        +Ls*c123456)*p3+2*(LGf*c1234567-
hGf*s1234567+Ls*s123456)*p1))*ddq6+(.5*mf*(2*p3*p3+2*p1*p1)+.5*mt*(2*p7*p7+2*p8*p8)+If+It+Is+.5*ms*(2*p6*p6

+2*p4*p4))*ddq5+(.5*mf*(2*p3*p3+2*p1*p1)+.5*mt*(2*p7*p7+2*p8*p8)+If+It+Is+.5*ms*(2*p6*p6+2*p4*p4))*ddq4+(.5*mt*(2*p7*(-(LGt
        -Lt)*c12345-hGt*s12345-Lt*c123)+2*p8*(-(LGt-Lt)*s12345+hGt*c12345-Lt*s123))+Is+It+.5*mf*(2*p3*(-Lt*c123+Lt*c12345-
LGf*s1234567-hGf*c1234567+Ls*c123456)
        +2*p1*(-Lt*s123+Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456))+.5*ms*(2*p6*(-Lt*c123+Lt*c12345-(LGs-Ls)*c123456-
hGs*s123456)+2*p4*(-Lt*s123+Lt*s12345
        -(LGs-Ls)*s123456+hGs*c123456))+If)*ddq3+(Is+.5*ms*(2*(-Ls*c12-Lt*c123+Lt*c12345-(LGs-Ls)*c123456-
hGs*s123456)*p6+2*(-Ls*s12-Lt*s123+Lt*s12345-(LGs-Ls)
        *s123456+hGs*c123456)*p4)+.5*mf*(2*(-Ls*c12-Lt*c123+Lt*c12345+Ls*c123456-LGf*s1234567-hGf*c1234567)*p3+2*(-Ls*s12-
Lt*s123+Lt*s12345+Ls*s123456+LGf*c1234567
        -hGf*s1234567)*p1)+It+If+.5*mt*(2*(-Ls*c12-Lt*c123-(LGt-Lt)*c12345-hGt*s12345)*p7+2*(-Ls*s12-Lt*s123-(LGt-
Lt)*s12345+hGt*c12345)*p8))*ddq2+(It+Is+.5*ms
        *(2*(Lt*s12345-Lt*s123-Ls*s12-(LGs-Ls)*s123456+hGs*c123456-Lf*c1)*p4+2*(Lt*c12345-Lt*c123-Ls*c12-(LGs-Ls)*c123456-
hGs*s123456+Lf*s1)*p6)+If+.5*mf*(2*(Lt
        *s12345+Ls*s123456-Lf*c1-Lt*s123+LGf*c1234567-hGf*s1234567-Ls*s12)*p1+2*(Lt*c12345+Ls*c123456+Lf*s1-Lt*c123-
LGf*s1234567-hGf*c1234567-Ls*c12)*p3)+.5*mt
        *(2*(-Ls*s12-Lf*c1-(LGt-Lt)*s12345+hGt*c12345-Lt*s123)*p8+2*(-Ls*c12+Lf*s1-(LGt-Lt)*c12345-hGt*s12345-
Lt*c123)*p7))*ddq1;

    // SSupKE6Sbs.txt
    p1 =  LGf*c1234567-hGf*s1234567;
    p2 =  LGf*c1234567-hGf*s1234567+Ls*s123456;
    p3 = -LGf*s1234567-hGf*c1234567+Ls*c123456;
    p4 = -(LGs-Ls)*c123456-hGs*s123456;
    p5 = -(LGs-Ls)*s123456+hGs*c123456;

    // SSupKE6.txt
    Tcc6 = .5*ms*(2*(-dq123456*(-(LGs-Ls)*s123456*dq123456+hGs*c123456*dq123456)
        +dq1*dq1*Lf*c1+dq123*dq123*Lt*s123-dq12345*dq12345*Lt*s12345+dq12*dq12*Ls*s12)*p4+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+Lt*s12345*dq12345+dq123456
```

```
        *p5)*(-(LGs-Ls)*c123456*dq123456-hGs*s123456*dq123456)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345-
dq123456*((LGs-Ls)*c123456+hGs*s123456))
        *((LGs-Ls)*s123456*dq123456-hGs*c123456*dq123456)+2*(dq123456*(-(LGs-Ls)*c123456*dq123456-
hGs*s123456*dq123456)+dq1*dq1*Lf*s1-dq123*dq123*Lt*c123+dq12345*dq12345
        *Lt*c12345-dq12*dq12*Ls*c12)*p5)-.5*ms*(-2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345-dq123456*((LGs-
Ls)*c123456+hGs*s123456))*dq123456*p5
        +2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+Lt*s12345*dq12345+dq123456*p5)*dq123456*p4)+.5*mf*(2*(-
dq123456*dq123456*Ls*s123456+dq1*dq1*Lf*c1-dq1234567*(LGf
        *c1234567*dq1234567-hGf*s1234567*dq1234567)+dq12*dq12*Ls*s12-dq12345*dq12345*Lt*s12345+dq123*dq123*Lt*s123)*p3+2*(-
dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123
        +Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567*p1)*(dq123456*Ls*c123456-LGf*s1234567*dq1234567-
hGf*c1234567*dq1234567)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123
        *Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-dq1234567*(LGf*s1234567+hGf*c1234567))*(-Ls*s123456*dq123456-
LGf*c1234567*dq1234567+hGf*s1234567*dq1234567)
        +2*(dq123456*dq123456*Ls*c123456+dq1*dq1*Lf*s1+dq1234567*(-LGf*s1234567*dq1234567-hGf*c1234567*dq1234567)-
dq12*dq12*Ls*c12+dq12345*dq12345*Lt*c12345-dq123*dq123
        *Lt*c123)*p2)-.5*mf*(2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-
dq1234567*(LGf*s1234567+hGf*c1234567))*(-Ls*s123456*dq123456
        -dq1234567*p1)+2*(-dq1*Lf*c1-Ls*s12*dq12-
Lt*s123*dq123+Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567*p1)*(dq123456*Ls*c123456+dq1234567*(-LGf*s1234567-
hGf*c1234567))) ;


    T6 = (.5*mf*(2*(-LGf*s1234567-
hGf*c1234567)*p3+2*p1*p2)+If)*ddq7+(Is+.5*ms*(2*p4*p4+2*p5*p5)+.5*mf*(2*p3*p3+2*p2*p2)+If)*ddq6+(.5*ms*(2*(Lt*c12345
        -(LGs-Ls)*c123456-hGs*s123456)*p4+2*(Lt*s12345-(LGs-Ls)*s123456+hGs*c123456)*p5)+.5*mf*(2*(Lt*c12345-LGf*s1234567-
hGf*c1234567+Ls*c123456)*p3+2*(Lt*s12345
        +LGf*c1234567-hGf*s1234567+Ls*s123456)*p2)+Is+If)*ddq5+(.5*ms*(2*(Lt*c12345-(LGs-Ls)*c123456-
hGs*s123456)*p4+2*(Lt*s12345-(LGs-Ls)*s123456+hGs*c123456)
        *p5)+.5*mf*(2*(Lt*c12345-LGf*s1234567-hGf*c1234567+Ls*c123456)*p3+2*(Lt*s12345+LGf*c1234567-
hGf*s1234567+Ls*s123456)*p2)+Is+If)*ddq4+(.5*ms*(2*(-Lt*c123
        +Lt*c12345-(LGs-Ls)*c123456-hGs*s123456)*p4+2*(-Lt*s123+Lt*s12345-(LGs-Ls)*s123456+hGs*c123456)*p5)+Is+.5*mf*(2*(-
Lt*c123+Lt*c12345-LGf*s1234567-hGf*c1234567
        +Ls*c123456)*p3+2*(-Lt*s123+Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456)*p2)+If)*ddq3+(Is+.5*mf*(2*(-Ls*c12-
Lt*c123+Lt*c12345+Ls*c123456-LGf*s1234567
        -hGf*c1234567)*p3+2*(-Ls*s12-Lt*s123+Lt*s12345+Ls*s123456+LGf*c1234567-hGf*s1234567)*p2)+If+.5*ms*(2*(-Ls*c12-(LGs-
Ls)*c123456-hGs*s123456-Lt*c123+Lt*c12345)
        *p4+2*(-Ls*s12-(LGs-Ls)*s123456+hGs*c123456-Lt*s123+Lt*s12345)*p5))*ddq2+(Is+.5*ms*(2*(Lt*s12345-Lt*s123-Ls*s12-
(LGs-Ls)*s123456+hGs*c123456-Lf*c1)*p5+2
        *(Lt*c12345-Lt*c123-Ls*c12-(LGs-Ls)*c123456-hGs*s123456+Lf*s1)*p4)+If+.5*mf*(2*(Lt*s12345+Ls*s123456-Lf*c1-
Lt*s123+LGf*c1234567-hGf*s1234567-Ls*s12)*p2
        +2*(Lt*c12345+Ls*c123456+Lf*s1-Lt*c123-LGf*s1234567-hGf*c1234567-Ls*c12)*p3))*ddq1;

    // SSupKE7Subs.txt
    p1 = -LGf*s1234567-hGf*c1234567;
    p2 =  LGf*c1234567-hGf*s1234567;

    // SSupKE7.txt
    Tcc7 = .5*mf*(2*(-dq123456*dq123456*Ls*s123456+dq1*dq1*Lf*c1
        -dq1234567*(LGf*c1234567*dq1234567-hGf*s1234567*dq1234567)+dq12*dq12*Ls*s12-
dq12345*dq12345*Lt*s12345+dq123*dq123*Lt*s123)*p1+2*(-dq1*Lf*c1-Ls*s12*dq12
        -Lt*s123*dq123+Lt*s12345+dq123456*dq1234567*p2)*(-LGf*s1234567*dq1234567-
hGf*c1234567*dq1234567)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123
        +dq12345*Lt*c12345+dq123456*Ls*c123456-dq1234567*(LGf*s1234567+hGf*c1234567))*(-
LGf*c1234567*dq1234567+hGf*s1234567*dq1234567)+2*(dq123456*dq123456*Ls*c123456
        +dq1*dq1*Lf*s1+dq1234567*(-LGf*s1234567*dq1234567-hGf*c1234567*dq1234567)-
dq12*dq12*Ls*c12+dq12345*dq12345*Lt*c12345-dq123*dq123*Lt*c123)*p2)-.5*mf
        *(-2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123+dq12345*Lt*c12345+dq123456*Ls*c123456-
dq1234567*(LGf*s1234567+hGf*c1234567))*dq1234567*p2+2*(-dq1*Lf*c1-Ls*s12
        *dq12-Lt*s123*dq123+Lt*s12345*dq12345+Ls*s123456*dq123456+dq1234567*p2)*dq1234567*p1);

    T7 = (.5*mf*(2*p1*p1+2*p2*p2)+If)*ddq7+(.5*mf*(2*p1*(-LGf*s1234567-hGf*c1234567+Ls*c123456)+2*p2*(LGf*c1234567-
hGf*s1234567+Ls*s123456))+If)*ddq6+(.5
        *mf*(2*(Lt*c12345-LGf*s1234567-hGf*c1234567+Ls*c123456)*p1+2*(Lt*s12345+LGf*c1234567-
hGf*s1234567+Ls*s123456)*p2)+If)*ddq5+(.5*mf*(2*(Lt*c12345-LGf*s1234567
        -hGf*c1234567+Ls*c123456)*p1+2*(Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456)*p2)+If)*ddq4+(.5*mf*(2*(-
Lt*c123+Lt*c12345-LGf*s1234567-hGf*c1234567+Ls
        *c123456)*p1+2*(-Lt*s123+Lt*s12345+LGf*c1234567-hGf*s1234567+Ls*s123456)*p2)+If)*ddq3+(.5*mf*(2*(-Ls*c12-
Lt*c123+Lt*c12345+Ls*c123456-LGf*s1234567-hGf*c1234567)
        *p1+2*(-Ls*s12-Lt*s123+Lt*s12345+Ls*s123456+LGf*c1234567-
hGf*s1234567)*p2)+If)*ddq2+(.5*mf*(2*(Lt*s12345+Ls*s123456-Lf*c1-Lt*s123+LGf*c1234567-hGf*s1234567
        -Ls*s12)*p2+2*(Lt*c12345+Ls*c123456+Lf*s1-Lt*c123-LGf*s1234567-hGf*c1234567-Ls*c12)*p1)+If)*ddq1;

    if (groundedLeg == LEFT){ // LEFT leg is grounded, Adjust torques to match sign convention [TankleL TkneeL ThipL
TankleR TkneeR ThipR]

        momentArms[0] = sensorData->jointData[LANKLE].momentArm;
        momentArms[1] = sensorData->jointData[LKNEE].momentArm;
        momentArms[2] = sensorData->jointData[LHIP].momentArm;

        GetTfrictionSwing(&Tf[RANKLE_T], &angles[RANKLE], &velocities[RANKLE], 1); // right leg is swing leg, legside (last
arg) not used in GetTfrictionSwing
        GetTfrictionSStance(&Tf[LANKLE_T], &angles[LANKLE], &angles[RANKLE], momentArms, sysProperties,'L'); //left leg is
stance leg

        // Tg = torque needed to hold device against gravity
        // Use Tg later for feedback linearization
        sensorData->jointData[LANKLE].Tg = -Tg2;
        sensorData->jointData[LKNEE].Tg  = -Tg3;
        sensorData->jointData[LHIP].Tg   = -Tg4; // left leg
        sensorData->jointData[RANKLE].Tg =  Tg7;
        sensorData->jointData[RKNEE].Tg  =  Tg6;
        sensorData->jointData[RHIP].Tg   =  Tg5; // right leg

        sensorData->jointData[LANKLE].Tcc =  Tcc2; // torques to counteract velocity forces
        sensorData->jointData[LKNEE].Tcc  =  Tcc3;
        sensorData->jointData[LHIP].Tcc   =  Tcc4; // left leg
        sensorData->jointData[RANKLE].Tcc = -Tcc7;
        sensorData->jointData[RKNEE].Tcc  = -Tcc6;
```

```
        sensorData->jointData[RHIP].Tcc   = -Tcc5; // right leg

        sensorData->jointData[LANKLE].Tinertial =  T2; // torques to counteract velocity forces
        sensorData->jointData[LKNEE].Tinertial  =  T3;
        sensorData->jointData[LHIP].Tinertial   =  T4; // left leg
        sensorData->jointData[RANKLE].Tinertial = -T7;
        sensorData->jointData[RKNEE].Tinertial  = -T6;
        sensorData->jointData[RHIP].Tinertial   = -T5; // right leg

        sensorData->jointData[LANKLE].Thm =  T2+Tcc2; // - torques[LANKLE_T]; //Tf[LANKLE_T] - Tg2 - torques[LANKLE_T] + T2
+ Tcc2;    // GROUNDED leg
        sensorData->jointData[LKNEE].Thm  =  T3+Tcc3; // - torques[LKNEE_T];  //Tf[LKNEE_T]  - Tg3 - torques[LKNEE_T]  + T3
+ Tcc3; // compute total THM torque for each joint
        sensorData->jointData[LHIP].Thm   =  T4+Tcc4; // - torques[LHIP_T];   //Tf[LHIP_T]   - Tg4 - torques[LHIP_T]   + T4
+ Tcc4;
        sensorData->jointData[RANKLE].Thm = -T7-Tcc7; // ankle
        sensorData->jointData[RKNEE].Thm  = -T6-Tcc6; // knee
        sensorData->jointData[RHIP].Thm   = -T5-Tcc5; // hip

    } else { // RIGHT leg is grounded

        momentArms[0] = sensorData->jointData[RANKLE].momentArm;
        momentArms[1] = sensorData->jointData[RKNEE].momentArm;
        momentArms[2] = sensorData->jointData[RHIP].momentArm;

        GetTfrictionSwing(&Tf[LANKLE_T], &angles[LANKLE], &velocities[LANKLE], 0); // left leg is swing leg, legside (last
arg) not used in GetTfrictionSwing
        GetTfrictionSStance(&Tf[RANKLE_T], &angles[RANKLE], &angles[LANKLE], momentArms, sysProperties,'R'); // right leg
is stance leg

        // Use Tg later for feedback linearization
        sensorData->jointData[LANKLE].Tg =  Tg7;
        sensorData->jointData[LKNEE].Tg  =  Tg6;
        sensorData->jointData[LHIP].Tg   =  Tg5; // left leg
        sensorData->jointData[RANKLE].Tg = -Tg2;
        sensorData->jointData[RKNEE].Tg  = -Tg3;
        sensorData->jointData[RHIP].Tg   = -Tg4; // right leg

        sensorData->jointData[LANKLE].Tcc = -Tcc7; // torques to counteract velocity forces
        sensorData->jointData[LKNEE].Tcc  = -Tcc6;
        sensorData->jointData[LHIP].Tcc   = -Tcc5; // left leg
        sensorData->jointData[RANKLE].Tcc =  Tcc2;
        sensorData->jointData[RKNEE].Tcc  =  Tcc3;
        sensorData->jointData[RHIP].Tcc   =  Tcc4; // right leg

        sensorData->jointData[LANKLE].Tinertial = -T7; // torques to counteract velocity forces
        sensorData->jointData[LKNEE].Tinertial  = -T6;
        sensorData->jointData[LHIP].Tinertial   = -T5; // left leg
        sensorData->jointData[RANKLE].Tinertial =  T2;
        sensorData->jointData[RKNEE].Tinertial  =  T3;
        sensorData->jointData[RHIP].Tinertial   =  T4; // right leg

        sensorData->jointData[LANKLE].Thm = -T7-Tcc7; // ankle
        sensorData->jointData[LKNEE].Thm  = -T6-Tcc6; // knee
        sensorData->jointData[LHIP].Thm   = -T5-Tcc5; // hip
        sensorData->jointData[RANKLE].Thm =  T2+Tcc2; // - torques[RANKLE_T]; //Tf[RANKLE_T] - Tg2 - torques[RANKLE_T] + T2
+ Tcc2;    // GROUNDED leg
        sensorData->jointData[RKNEE].Thm  =  T3+Tcc3; // - torques[RKNEE_T];  //Tf[RKNEE_T]  - Tg3 - torques[RKNEE_T]  + T3
+ Tcc3; // compute total THM torque for each joint
        sensorData->jointData[RHIP].Thm   =  T4+Tcc4; // - torques[RHIP_T];   //Tf[RHIP_T]   - Tg4 - torques[RHIP_T]   + T4
+ Tcc4;
    }

    // friction and stiffness
    sensorData->jointData[LANKLE].Tf = Tf[LANKLE_T];
    sensorData->jointData[LKNEE].Tf  = Tf[LKNEE_T];
    sensorData->jointData[LHIP].Tf   = Tf[LHIP_T];
    sensorData->jointData[RANKLE].Tf = Tf[RANKLE_T];
    sensorData->jointData[RKNEE].Tf  = Tf[RKNEE_T];
    sensorData->jointData[RHIP].Tf   = Tf[RHIP_T];

    // compute torque for feedback linearization
    sensorData->jointData[LANKLE].Tlin = sensorData->jointData[LANKLE].Tg + sensorData->jointData[LANKLE].Tf;
    sensorData->jointData[LKNEE].Tlin  = sensorData->jointData[LKNEE].Tg  + sensorData->jointData[LKNEE].Tf;
    sensorData->jointData[LHIP].Tlin   = sensorData->jointData[LHIP].Tg   + sensorData->jointData[LHIP].Tf;
    sensorData->jointData[RANKLE].Tlin = sensorData->jointData[RANKLE].Tg + sensorData->jointData[RANKLE].Tf;
    sensorData->jointData[RKNEE].Tlin  = sensorData->jointData[RKNEE].Tg  + sensorData->jointData[RKNEE].Tf;
    sensorData->jointData[RHIP].Tlin   = sensorData->jointData[RHIP].Tg   + sensorData->jointData[RHIP].Tf;

    // set toe torques to zero
    sensorData->jointData[LTOE].Tg            = 0;
    sensorData->jointData[RTOE].Tg            = 0;
    sensorData->jointData[LTOE].Thm           = 0;
    sensorData->jointData[RTOE].Thm           = 0;
    sensorData->jointData[LTOE].Tlin        = 0;
    sensorData->jointData[RTOE].Tlin        = 0;
    sensorData->jointData[LTOE].Tcc           = 0;
    sensorData->jointData[RTOE].Tcc           = 0;
    sensorData->jointData[LTOE].Tf            = 0;
    sensorData->jointData[RTOE].Tf            = 0;
    sensorData->jointData[LTOE].Tinertial   = 0;
    sensorData->jointData[RTOE].Tinertial   = 0;
}


/* Function: GetJTsensorInTorsoFrame
 * ---------------------------------------------------------------------------------------------
 * Calculates the transpose jacobian matrix of the backpack force sensor for a 3dof leg and updates JT.
 * [T2 T3 T4]' = JT_4 * [Fx Fy Tz]_sensor
```

```
 * and JT_4 = JT_0 R04, where JT_0 is the Jacobian in frame 0 and R04 is the rotation matrix fom frame0
 * to frame 4.
 * Jacobian is obtained from sensorJacobian.m
 */
void GetJTsensorInTorsoFrame(double             JT[][3],
                             const double     kneeAngle,
                             const double     hipAngle,
                             const BodyDataT *bodyData){

    double q3, q4, Ls, Lt, Lsn, hsn, c4, s4;

    Ls  = bodyData->shank.length;
    Lt  = bodyData->thigh.length;
    Lsn = bodyData->torsoSensor_L;
    hsn = bodyData->torsoSensor_h;
    q3  = kneeAngle;
    q4  = hipAngle;
    c4  = cos(q4);
    s4  = sin(q4);

    //[row][col]
    JT[0][0] = -Ls*cos(q3+q4)-Lt*c4-Lsn;
    JT[0][1] =  Ls*sin(q3+q4)+Lt*s4+hsn;
    JT[0][2] =  1;
    JT[1][0] = -Lt*c4-Lsn;
    JT[1][1] =  Lt*s4+hsn;
    JT[1][2] =  1;
    JT[2][0] = -Lsn;
    JT[2][1] =  hsn;
    JT[2][2] =  1;
}




/* Function: GetTfrictionSStance
 * ---------------------------------------------------------------------------------------------
 * Calculates the joint torques vector to counteract joint friction and stiffness for one leg. Updates Tf.
 * The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
 * Equations are obtained from Excel documents 'rankle stiffness.xls', 'rknee stiffness.xls', 'rhip stiffness.xls'.
 * legSide =0 for left leg and 1 for right leg
 */
void GetTfrictionSStance(double                   *Tf,
                         const double            *angles,
                         const double            *otherLegAngles,
                         const double            *momentArms,
                         const SysPropertiesT    *sysProperties,
                         const char               legSide){

    double torque_slope[3], torque_offset[3], force_slope[3], force_offset[3], shankAngle, hipAngleD, extraKneeT;

//    if(angles[0] < -15*Pi/180){         // ankle
//        Tf[ANKLE_T] = 20;
//    }else if(angles[0] < 0){
//        Tf[ANKLE_T] = 10;
//    }else{
//        Tf[ANKLE_T] = 0;
//    }
//
//
//    if(legSide =='L'){    // left leg
//        hipAngleD = otherLegAngles[2] - angles[2]; // difference in hip joint angles
//        if(hipAngleD > 0){
//            extraKneeT = 43*hipAngleD; // extra knee torque so that stance leg doesn't straighten too much when swing leg
is back
//        }else{
//            extraKneeT = 0;
//        }
//
//        //Tf[ANKLE_T] = -6; //-2;         // left stance 06 03 03
//        Tf[KNEE_T] =  -2 + extraKneeT; //-12 + extraKneeT;  //left stance 06 10 03
//        Tf[HIP_T] = -13; //-20;
//
//    }else{            // right leg
//
//        hipAngleD = otherLegAngles[2] - angles[2]; // difference in hip joint angles
//        if(hipAngleD > 0){
//            extraKneeT = 43*hipAngleD; // extra knee torque so that stance leg doesn't straighten too much when swing leg
is back
//        }else{
//            extraKneeT = 0;
//        }
//
//        //Tf[ANKLE_T] = -6;
//        Tf[KNEE_T] = -2 + extraKneeT;
//        Tf[HIP_T] = -13;
//    }

    Tf[ANKLE_T] = 0;
    Tf[KNEE_T]  = 0;
    Tf[HIP_T]   = 0;

}
```

Appendix A.16 – 1Red.h

```
/* Function: DoubleSupportSingleRedundancyTHM
* ------------------------------------------------
* Calculates the joint torques due to gravity(Tg) and due to the human (THM) during the double support
* mode with a kinematic redundancy in one of the the legs. i.e. one leg is 3dof, the other is 4dof
* The vector is as follows:-[TankleL TkneeL ThipL TankleR TkneeR ThipR]
*/
void DoubleSupportSingleRedundancyTHM(double               *angles,
                                      double               *velocities,
                                      double               *accelerations,
                                      double               *torques,
                                      const BodyDataT       *bodyData,
                                      int                   redundantLeg,
                                      int                   leftHeelContact,
                                      int                   rightHeelContact,
                                      const double           Kf,
                                      const double           *torsoForces,
                                      ForceDistributionT    *distrData,
                                      SensorDataT           *sensorData,
                                      const VirtualGuardT    vguard,
                                      const SysPropertiesT   *sysProperties);


/* Function: Get4dofTorques
* ------------------------------------------------
* Calculates the joint torques vector due to the human for a 4dof leg, and updates THM.
* The vector is as follows:[Tankle Tknee Thip]
*/
void Get4dofTorques(double               *Tg,
                    double               *Tcc,
                    double               *Tf,
                    double               *THM,
                    const double        *angles,
                    const double        *velocities,
                    const double        *accelerations,
                    const double        *torques,
                    const BodyDataT       *bodyData,
                    const int            heelContact,
                    const double        *trig,
                    const char            side,
                    const SensorDataT    *sensorData);


/* Function: GetJT44 - ABZ 2004-10-05
* ------------------------------------------------
* Calculates the transpose jacobian matrix for a 4dof leg and updates JT.
* The jacobian transforms an opertional force of the form [Fx_hip Fy_hip Tz_hip Tz_foot] into a torque vector [Tfoot Tankl
Tknee Thip]
* T = JT44 F
*/
void GetJT44(double               JT[][4],
             const BodyDataT    *bodyData,
             const double        *trig);


/* Function: GetJinvT44
* ------------------------------------------------------------------------------------------------
* Calculates the inverse transpose jacobian matrix JinT for a 4dof leg. F = JinvT T where T is the
* joint torque vector and F is the operational force [Fx@hip Fy@hip Tz@hip Tfoot] .
*/
void GetJinvT44(double            JinvT[][4],
                const BodyDataT *bodyData,
                const int        heelContact,
                const double    *trig);


/* Function: GetJp6T
* ------------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp6T for a double support system with one redundancy.
* Tp6 = Jp6T Fp6 is the joint torque vector and F is the operational force system vector
* F = [FLx@hip FLy@hip FRx@hip FRy@hip Tz@hip Tz@foot] .
*/
void GetJp6T(double               Jp6T[][6],
             const BodyDataT    *bodyData,
             const int           heelContact,
             const double        *trigRD,
             const double        *trigNR);


/* Function: GetJp6Tnew   (may 27 03)
* ------------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp6T for a double support system with one redundancy.
* Tp6 = Jp6T Fp6 is the joint torque vector and F is the operational force system vector
* F = [FLx@hip FLy@hip FRx@hip FRy@hip Tz@hip Tz@foot] .
* see J6xT in TestForceDistr.m
*/
void GetJp6Tnew(double            Jp6T[][6],
                const BodyDataT *bodyData,
                const int        heelContact,
                const double    *trigRD,
                const double    *trigNR);


/* Function: GetJp5xT
* ------------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp5T for a system with a zero ankle torque at the non-redundant leg.
```

```
* Tp5 = Jp5T Fp5 is the joint torque vector and F is the operational force system vector
* Fp5 = [FNRx@hip FRDx@hip Fy@hip Tz@hip Tz@foot]
*/
void GetJp5xT(double            Jp5xT[][5],
              const BodyDataT    *bodyData,
              const int           heelContact,
              const double        *trigRD,
              const double        *trigNR);


/* Function: GetJp5xxT
* ------------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp5T for a system with a zero ankle torque at the redundant leg.
* Tp5 = Jp5xxT Fp5 is the joint torque vector and F is the operational force system vector
* Fp5 = [FNRx@hip FRDx@hip Fy@hip Tz@hip Tz@foot]
*/
void GetJp5xxT(double            Jp5xxT[][5],
               const BodyDataT    *bodyData,
               const int         heelContact,
               const double        *trigRD,
               const double        *trigNR);


/* Function: GetTRDfootOperational
* ------------------------------------------------------------------------------------------------
* Calculates the foot operational torque for a 4dof leg. TRDfoot is the result
* THMRD is the leg's joint torque vector.
*/
double GetTRDfootOperational(double               *THMRD,
                             const BodyDataT    *bodyData,
                             const int           heelContact,
                             const double        *trig);


/* Function: GetJp4xT
* ------------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp4T for a system with two zero ankle torques.
* Tp4 = Jp4xT Fp4 is the joint torque vector and F is the operational force system vector
* Fp4 = [Fx@hip Fy@hip Tz@hip Tz@foot]
*/
void GetJp4xT(double Jp4xT[][4],
              const double *RDangles,
              const double *NRangles,
              const BodyDataT *bodyData,
              const int heelContact,
              const double *trigRD,
              const double *trigNR);


/* Function: ComputeTrig_1Red
* ------------------------------------------------------------------------------------------------
* Computes trigonometric sin and cos functions for the 1Redundancy Double Support state and stores them in an
* array.
* this function reduce the number of cos and sin to be computed in the 1Red double stance state by half.
*/
void ComputeTrig_1Red(double *trigRD,
                      double *trigNR,
                      const double *RDangles,
                      const double *NRangles,
                      const int heelContact);


/* Function: GetTfrictionRedtLeg
* ------------------------------------------------
* Calculates the joint torques vector to counteract joint friction and stiffness for one leg. Updates Tf.
* The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
*/
void GetTfrictionRedtLeg(double *Tf,
                         const double *angles,
                         const char side,
                         const SensorDataT *sensorData);


/* Function: GetTfrictionNRedtLeg
* ------------------------------------------------
* Calculates the joint torques vector to counteract joint friction and stiffness for one leg. Updates Tf.
* The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
*/
void GetTfrictionNRedtLeg(double *Tf,
                          const double *angles,
                          const char side,
                          const int heelContact,
                          const double dNR,
                          const double dRD,
                          const BodyDataT *bodyData);
```

Appendix A.17 – 1Red.c

```
#include <math.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"
#include "DSup.h"
#include "1Red.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];

extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];

/* Function: DoubleSupportSingleRedundancyTHM
* ------------------------------------------------------------------------------------------
* Calculates the joint torques due to gravity(Tg) and due to the human (THM) during the double support
* mode with a kinematic redundancy in one of the the legs. i.e. one leg is 3dof, the other is 4dof
* The vector is as follows:-[TankleL TkneeL ThipL TankleR TkneeR ThipR]
*/
void DoubleSupportSingleRedundancyTHM(double               *angles,
                                      double               *velocities,
                                      double               *accelerations,
                                      double               *torques,
                                      const BodyDataT       *bodyData,
                                      int                   redundantLeg,
                                      int                   leftHeelContact,
                                      int                   rightHeelContact,
                                      const double           Kf,
                                      const double           *torsoForces,
                                      ForceDistributionT   *distrData,
                                      SensorDataT           *sensorData,
                                      const VirtualGuardT   vguard,
                                      const SysPropertiesT   *sysProperties){

    double *RDangles, *NRangles, *RDvel, *NRvel, *RDacc, *NRacc, *RDhipAngles, *NRhipAngles; // joint
angles,velocities,acc., vectors for each leg
    double TgRD[4], TgNR[3], THMRD[4], THMNR[3]; // computed leg torques in the form [Tankle Tknee Thip]
    double TccNR[3] = {0,0,0}; // centrifugal and coriolis force vectors
    double TccRD[4] = {0,0,0,0};
    double *RDtorques, *NRtorques; // measured input torques for each leg [Tankle Tknee Thip]
    double Jp6T[6][6], Jp5xT[5][5], Jp5xxT[5][5], Jp4xT[4][4]; // jacobians
    double JNRinvT[3][3], JRDinvT[4][4];
    double FgRD[4], FgNR[3], FHMRD[4], FHMNR[3];// operational force computed on each leg [Fx@hip Fy@hip Tz@hip Tfoot]
    double Fgin[4];
    double FHMin[4]; //operational force on machine [Fx@hip Fy@hip Tz@hip Tfoot]
    double FgRDin[4], FgNRin[3],FHMRDin[4],FHMNRin[3]; // operational forces distibuted to each leg [Fx@hip Fy@hip Tz@hip
Tfoot]
    double Fgp6[6],FHMp6[6],Fgp5[5],FHMp5[5]; // augmented operational forces
    double Tgtemp[6], THMtemp[6];// temporary joint torque vectors [TankleNR TkneeNR ThipNR TankleRD TkneeRD ThipRD]
    char   NRside, RDside; // side of the non redundant annd redundant legs ('L' or 'R')
    int    i, sgn_dRD, sgn_dNR, heelContact;
    double dRD = 0;     // transverse plane distance from the torso CG to the redundant foot pressure point
    double dNR = 0;     // transverse plane distance from the torso CG to the non-redundant foot pressure point
    double xRD = 0; // sagittal plane horizontal distance from the torso CG to the redundant foot pressure point
    double xNR = 0; // sagittal plane horizontal distance from the torso CG to the non-redundant foot pressure point
    double alpha   = 0;
    double betaFg  = 0;
    double betaFHM = 0;
    double Krot; // hip rotation factor
    double c234, s234; //torso angle, cos and sin
    double THM5xx[5], Tg5xx[5], THM4[4] ,Tg4[4];
    double TlinNR[3], TlinRD[4]; // feedback linearization torques
    double Lsn, hsn; // distances from hip to F/T sensor on the torso
    double TfNR[3]       = {0,0,0};
    double TfRD[4]       = {0,0,0,0};   // joint torque stiffness and friction vectors
    double trigNR[10]    = {0,0,0,0,0,0,0,0,0,0}; // point to trig array for redundant or non-redundant leg only
       double trigRD[14]    = {0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // contains sin and cos functions used throughout this
function
    double momentArms[3] = {0,0,0}; // actuator moment arms (m)
    double J3[3][3], J4[4][4], Tgtemp2[4]; // ABZ 2004-10-05

    Lsn = bodyData->torsoSensor_L; // distances b/w F/T sensor and hip
    hsn = bodyData->torsoSensor_h;

    if (redundantLeg == LEFT){
        heelContact = leftHeelContact;
        RDangles    = &angles[LTOE]; // Redundant leg angles: Rangles = [Ltoe Lankle Lknee Lhip]
        NRangles    = &angles[RANKLE]; // Non-Redundant leg angles NRangles = [Rangle Rknee Rhip]
        NRangles[0] = angles[RTOE] + angles[RANKLE]; //include toe angle in ankle
```

```
        RDhipAngles = &angles[LHIP_ROT]; // redundant leg hip [rotation abduction]
        NRhipAngles = &angles[RHIP_ROT]; // non-redundant leg hip [rotation abduction]
        RDvel       = &velocities[LTOE];
        NRvel       = &velocities[RANKLE];
        RDacc       = &accelerations[LTOE];
        NRacc       = &accelerations[RANKLE];
        RDtorques   = torques; // [Lankle Lknee Lhip]
        NRtorques   = &torques[RANKLE_T]; // [Rankle Rknee Rhip]
        NRside      = 'R'; // set the non-redundant side to be the right side
        RDside      = 'L'; // set the redundant side to be the left side
    }else{     // Right leg is redundant
        heelContact = rightHeelContact;
        RDangles    = &angles[RTOE]; // Redundant leg angles: RDangles = [Rtoe Rankle Rknee Rhip]
        NRangles    = &angles[LANKLE]; // Non-Redundant leg angles NRangles = [Langle Lknee Lhip]
        NRangles[0] = angles[LTOE] + angles[LANKLE]; //include toe angle in ankle
        RDhipAngles = &angles[RHIP_ROT]; // redundant leg hip [rotation abduction]
        NRhipAngles = &angles[LHIP_ROT]; // non-redundant leg hip [rotation abduction]
        RDvel       = &velocities[RTOE];
        NRvel       = &velocities[LANKLE];
        RDacc       = &accelerations[RTOE];
        NRacc       = &accelerations[LANKLE];
        RDtorques   = &torques[RANKLE_T]; // [Rankle Rknee Rhip]
        NRtorques   = torques; // [Lankle Lknee Lhip]
        NRside      = 'L'; // set the non-redundant side to be the left side
        RDside      = 'R'; // set the redundant side to be the right side
    }

    ComputeTrig_1Red(trigRD, trigNR, RDangles, NRangles, heelContact);

    // compute foot distances
    GetFootDist2D(NRhipAngles, bodyData, 0, NRside, trigNR, &xNR, &dNR, 1, 1, sensorData); // Get transverse plane
distance  from the ankle
    GetFootDist2D(RDhipAngles, bodyData, 1, RDside, trigRD, &xRD, &dRD, 0, heelContact, sensorData); // to the CG
    sgn_dRD = (int) (dRD/ fabs(dRD)); // compute the sign of dRD
    sgn_dNR = (int) (dNR/ fabs(dNR)); // compute the sign of dNR

    // Compute human-machine torques for the redundant leg

    Get4dofTorques(TgRD, TccRD, TfRD, THMRD, RDangles, RDvel, RDacc, RDtorques, bodyData, heelContact, trigRD, RDside,
sensorData); // Human-machine torques of redundant leg

    // Compute feedback linearization joint torques

    GetTfrictionNRedtLeg(TfNR, NRangles, NRside, heelContact, dNR, dRD, bodyData); // get joint friction and stiffness
torques

    Get3dofTg(TgNR, bodyData, trigNR, NRside, sensorData); //Compute joint torques to hold gravity

    GetJinvT33(JNRinvT, bodyData, trigNR); // Jacobian inverse transpose for non-redundant leg
    GetJinvT44(JRDinvT, bodyData, heelContact, trigRD); // Jacobian inverse transpose for redundant leg

    for(i=3; i; i--){
        TlinNR[i-1] = TgNR[i-1] + TfNR[i-1]; // compute linearizing torque
        TlinRD[i-1] = TgRD[i-1] + TfRD[i-1];
    }

    TlinRD[3] = TgRD[3] + TfRD[3]; // RD hip

    // NOTE: Fg, despite its name, incorporates all linearizing elements (Tcc,Tg and Tf). In the remainder of this function
    // all variables with the 'g' suffix should be assumed to incorporate not only gravity, but velocity and friction
terms.

    MatVectMult(FgNR, &JNRinvT[0][0], TlinNR,3,3); // Compute the operational force due to gravity (and friction) for each
leg
    MatVectMult(FgRD, &JRDinvT[0][0], TlinRD,4,4);

    for (i=3; i; i--){
        Fgin[i-1] = FgRD[i-1] + FgNR[i-1]; // Compute the total operational force due to gravity
    }

    Fgin[3] = FgRD[3]; // redundant foot torque

//    // Compute force distribution factors
//
//    // ----------------------NEW -------------------------------06 04 03
//    if(NRside == 'L'){
//        alpha = fabs(dRD)/ (fabs(dRD)+fabs(dNR)) + Kf;         // compute the weight distribution factor
//        betaFg = alpha - sgn_dNR * fabs(dRD - dNR) * Kf * Fgin[0]; // Distribution factor (method 1)
//                                                                if( sensorData->Rfootswitch[TOE]/* ||            // if
//foot almost off the ground put all force on other leg
//                                                                    (sensorData->Rfootswitch[BALL] && !(sensorData-
//>Rfootswitch[MIDFOOT]) )*/ ){
//                                                                    //    betaFg = betaFg + 4*fabs(sensorData-
//>jointData[LTOE].position);
//                                                                    betaFg = 1;
//                                                                //if(betaFg < 0.7) betaFg = 0.7;
//                                                                }
//    }else{ // NRside == 'R'
//        alpha = fabs(dNR)/ (fabs(dRD)+fabs(dNR)) - Kf;
//        betaFg = alpha - sgn_dRD * fabs(dRD - dNR) * Kf * Fgin[0]; // Distribution factor (method 1)
//                                                                if( sensorData->Lfootswitch[TOE] /*||            // if
//foot almost off the ground put all force on other leg
//                                                                    (sensorData->Lfootswitch[BALL] && !(sensorData-
//>Lfootswitch[MIDFOOT]) ) */){
//                                                                    //    betaFg = betaFg + 4*fabs(sensorData-
//>jointData[RTOE].position);
//                                                                    betaFg = 0; //07 02 03
//                                                                //if(betaFg > 0.3) betaFg = 0.3;
//                                                                }
```

```
//    }
//
//    //-------------------------------------------------
//    // ----------------------NEW --------------------------------07 11 03
//    /*    if(NRside == 'L'){
//    alpha = fabs(dRD)/ (fabs(dRD)+fabs(dNR)) + Kf;          // compute the weight distribution factor
//    betaFg = alpha - sgn_dNR * fabs(dRD - dNR) * Kf * Fgin[0]; // Distribution factor (method 1)
//    if( sensorData->Rfootswitch[TOE] ||             // if foot almost off the ground put all force on other leg
//    (sensorData->Rfootswitch[BALL] && !(sensorData->Rfootswitch[MIDFOOT]) ) ){
//                betaFg = 1;
//                }
//                }else{ // NRside == 'R'
//                alpha = fabs(dNR)/ (fabs(dRD)+fabs(dNR)) - Kf;
//                betaFg = alpha - sgn_dRD * fabs(dRD - dNR) * Kf * Fgin[0]; // Distribution factor (method 1)
//                if( sensorData->Lfootswitch[TOE] ||            // if foot almost off the ground put all force on other
leg
//                (sensorData->Lfootswitch[BALL] && !(sensorData->Lfootswitch[MIDFOOT]) ) ){
//                betaFg = 0; //07 02 03
//                }
//                }
//    */
//    //-------------------------------------------------
//
//    if(NRside == 'L'){
//        alpha = fabs(dRD)/ (fabs(dRD)+fabs(dNR)) + Kf;          // compute the weight distribution factor
//        betaFg = alpha - sgn_dNR * fabs(dRD - dNR) * Kf * Fgin[0]; // Distribution factor (method 1)
//        if( sensorData->Rfootswitch[TOE] && !(sensorData->Rfootswitch[MIDFOOT])  ){
//            betaFg = 1;
//        }else if(sensorData->Rfootswitch[BALL] && !(sensorData->Rfootswitch[MIDFOOT]) ){
//            if(betaFg < 0.7) betaFg = 0.7;
//        }
//    }else{ // NRside == 'R'
//        alpha = fabs(dNR)/ (fabs(dRD)+fabs(dNR)) - Kf;
//        betaFg = alpha - sgn_dRD * fabs(dRD - dNR) * Kf * Fgin[0]; // Distribution factor (method 1)
//        if( sensorData->Lfootswitch[TOE] && !(sensorData->Lfootswitch[MIDFOOT]) ){
//            betaFg = 0; //07 02 03
//        }else if(sensorData->Lfootswitch[BALL] && !(sensorData->Lfootswitch[MIDFOOT]) ){
//            if(betaFg > 0.3) betaFg = 0.3;
//        }
//    }
//
//
//    //    betaFg = alpha - sgn_dRD * fabs(dRD - dNR) * Kf * Fgin[0]; // Distribution factor (method 1)
//    //    betaFg = alpha - sgn_dRD * fabs(dRD - dNR) * Fgin[0]/(fabs(Fgin[0])+Kf*fabs(Fgin[1])); // (method 2)
//    //    betaFg = alpha * (1 - sgn_dRD * Kf * Fgin[0]); //  (method 3)
//    //    betaFg = 0.5 + (alpha-0.5)/fabs(alpha-0.5) * pow(fabs(0.5-alpha),/*Kf*/1);// (method 4)
//
//    if (betaFg < 0){            // saturate beta
//        betaFg = 0;
//    }else if(betaFg > 1){
//        betaFg = 1;
//    }
//
//    // filter beta to ensure smooth transition between states
////    LowpassFilter(distrData->unfilteredBetaFg, distrData->filteredBetaFg, &betaFg, filterCoeffs2nd05); // 1 Hz (05)
//
//    if (betaFg < 0){            // saturate beta
//        betaFg = 0;
//    }else if(betaFg > 1){
//        betaFg = 1;
//    }


    // ******************for DEBUGGING 2004-09-08*******
    betaFg = 0.5;
    //*************************************************

    /*  // METHOD 1
    for (i=2; i; i--){
    FgRDin[i-1] = betaFg * Fgin[i-1]; // Distribute the horiz. and vert. operational forces
    FgNRin[i-1] = Fgin[i-1] - FgRDin[i-1];
    }

      Fgp6[0] = FgNRin[0];  // FNRx         // create augmented operational force vectors
      Fgp6[1] = FgNRin[1];  // FNRy
      Fgp6[2] = FgRDin[0];  // FRDx
      Fgp6[3] = FgRDin[1];  // FRDy
      Fgp6[4] = Fgin[2];    // Thip
      Fgp6[5] = Fgin[3];    // TRDfoot

        GetJp6T(Jp6T, bodyData, heelContact, trigRD, trigNR); // Compute the new system Jacobian transpose

          MatVectMult(Tgtemp, &Jp6T[0][0], Fgp6,6,6);        // Compute temp joint torque vectors
    */

    // METHOD 2 05 27 03  much better, no oscillations or vibrations


    // ----------------------NEW --------------------------------06 04 03 METHOD A
    for (i=1; i<3; i++){
        if(NRside == 'L'){
            FgNRin[i] = betaFg * Fgin[i]; // Distribute the torque and vert. operational forces
            FgRDin[i] = Fgin[i] - FgNRin[i];
        }else{
            FgRDin[i] = betaFg * Fgin[i]; // Distribute the torque and vert. operational forces
            FgNRin[i] = Fgin[i] - FgRDin[i];
        }
    }
```

```
    //-----------------------NEW ---------------------------------07 06 03
    /*
    if(NRside == 'L'){
    FgNRin[1] = alpha * Fgin[1]; // Distribute the vert. operational forces
    FgNRin[2] = betaFg * Fgin[2]; // Distribute the hip torque

      for (i=1; i<3; i++){
      FgRDin[i] = Fgin[i] - FgNRin[i];
      }
      }else{
      FgRDin[1] = alpha * Fgin[1]; // Distribute the vert. operational forces
      FgRDin[2] = betaFg * Fgin[2]; // Distribute the hip torque

                          for (i=1; i<3; i++){
                          FgNRin[i] = Fgin[i] - FgRDin[i];
                          }
                          }
    */
    // -----------------------NEW ---------------------------------06 06 03  METHOD B
    /*    if(!sensorData->Rfootswitch[TOE]){ // no toe contact
    for (i=1; i<3; i++){
    if(NRside == 'L'){
                FgNRin[i] = betaFg * Fgin[i]; // Distribute the torque and vert. operational forces
                FgRDin[i] = Fgin[i] - FgNRin[i];

                  }else{
                  FgRDin[i] = betaFg * Fgin[i]; // Distribute the torque and vert. operational forces
                  FgNRin[i] = Fgin[i] - FgRDin[i];
                  }
                  }
                  }else{ // toe contact
                  FgNRin[1] = Fgin[1];        // Assign full vert. operational force to NR leg
                  FgRDin[1] = 0;

                    if(NRside == 'L'){
                    FgNRin[2] = betaFg * Fgin[2];        // Distribute the hip torque
                    FgRDin[2] = Fgin[2] - FgNRin[2];
                    }else{
                    FgRDin[2] = betaFg * Fgin[2];
                    FgNRin[2] = Fgin[2] - FgNRin[2];    // Distribute the hip torque
                    }
                    }
    */

    // commented out on 2004/10/26
    Fgp6[0] = Fgin[0];    // Fx      // create augmented operational force vectors
    Fgp6[1] = FgNRin[1]; // FNRy
    Fgp6[2] = FgRDin[1]; // FRDy
    Fgp6[3] = FgNRin[2]; // TNRhip
    Fgp6[4] = FgRDin[2]; // TRDhip
    Fgp6[5] = Fgin[3];   // TRDfoot

    GetJp6Tnew(Jp6T, bodyData, heelContact, trigRD, trigNR); // Compute the new system Jacobian transpose

    MatVectMult(Tgtemp, &Jp6T[0][0], Fgp6,6,6); // Compute temp joint torque vectors

//NEW SCHOOL METHOD - ABZ 2004-10-05
//    GetJT33(J3, bodyData, trigNR);
//    GetJT44(J4, bodyData, trigRD);
//
//    MatVectMult(Tgtemp,  &J3[0][0], FgNRin, 3, 3);
//    MatVectMult(Tgtemp2, &J4[0][0], FgRDin, 4, 4);
//
//    Tgtemp[3] = Tgtemp2[1];
//    Tgtemp[4] = Tgtemp2[2];
//    Tgtemp[5] = Tgtemp2[3];
//NEW SCHOOL METHOD - ABZ 2004-10-05

    // Compute the total operational force vector
    /*
    #ifdef USE_BACKPACK_FORCE_SENSOR

      // Compute human-machine backpack force in inertial reference frame
      s234 = trigNR[S234]; c234 = trigNR[C234];

        // Sensor output torsoForces[] is given in torso sensor coordinate frame at the sensor location
        // ->express the equivalent operational force in the inertial Frame and at the hip in  FHMin[]

          FHMin[0] = torsoForces[0]*c234 - torsoForces[1]*s234; // Fx_0
          FHMin[1] = torsoForces[0]*s234 + torsoForces[1]*c234; // Fy_0
          FHMin[2] = torsoForces[2] - Lsn*torsoForces[0] + hsn*torsoForces[1];
          // torque_0 = torque_4 - Lsn * Fx_4 + hsn * Fy_4
          // Compute redundant foot torque component of operational force vector FHMin = [Fx Fy Thip TRDfoot]
          FHMin[3] = GetTRDfootOperational(THMRD, bodyData, heelContact, trigRD); // redundant foot op. torque

            #else
    */
    Get3dofTcc(TccNR, NRvel, bodyData, trigNR); // compute torque due to coriolis and centrifugal forces: Tcc = [ankle knee
hip]

    //    Get3dofTHM(TgNR, TccNR, TfNR, THMNR, TinertialNR, NRvel, NRacc, NRtorques, bodyData, trigNR); // Human-machine
torques of non-redundant leg

    MatVectMult(FHMNR, &JNRinvT[0][0], THMNR,3,3); // Compute the operational force due to human for each leg
    MatVectMult(FHMRD, &JRDinvT[0][0], THMRD,4,4);

    for (i=3; i; i--){
        FHMin[i-1] = FHMRD[i-1] + FHMNR[i-1]; // Compute the total operational forces at the hip due to the human
    }
```

```
    FHMin[3] = FHMRD[3]; // redundant foot op. torque

    //#endif

    // compute virtual guard force if necessary
    GetVGuardForces(sensorData, bodyData, vguard, xNR, xRD, trigNR[C234], trigNR[S234]);
    FHMin[0] = FHMin[0] + sensorData->virtualGuardFx; // add horizontal VGuard force
    FHMin[2] = FHMin[2] + sensorData->virtualGuardT;  // add hip moment caused by VGuard force at torso CG

    GetHipRotationFactor(&Krot, angles); //compute horizontal force hip rotation factor
//    LowpassFilter(distrData->unfilteredKrot, distrData->filteredKrot, &Krot, filterCoeffs2nd2); // filter Krot for smooth
transition


    //************FOR DEBUGGING ONLY
    Krot = 1; // set this to one for debugging
    //*************************************

    FHMin[0] = Krot*FHMin[0];

    if(NRside == 'L'){
        betaFHM = alpha - sgn_dNR * fabs(dRD - dNR) * Kf * FHMin[0]; // Distribution factor (method 1)
    }else{ // RDside == 'R'
        betaFHM = alpha - sgn_dRD * fabs(dRD - dNR) * Kf * FHMin[0]; // Distribution factor (method 1)
    }

    //    betaFHM = alpha - sgn_dRD * fabs(dRD - dNR) * Kf * FHMin[0]; // Distribution factor (method 1)
    //    betaFHM =  alpha - sgn_dRD * FHMin[0]/(fabs(FHMin[0])+Kf*fabs(FHMin[1])); // (method 2)
    //    betaFHM = alpha * (1 - sgn_dRD * Kf * FHMin[0]); // (method 3)

    if (betaFHM < 0){ // saturate beta
        betaFHM = 0;
    }else if(betaFHM > 1){
        betaFHM = 1;
    }
    // filter beta to ensure smooth transition between states
//    LowpassFilter(distrData->unfilteredBetaFHM, distrData->filteredBetaFHM, &betaFHM, filterCoeffs2nd1);

    if (betaFHM < 0){ // saturate beta
        betaFHM = 0;
    }else if(betaFHM > 1){
        betaFHM = 1;
    }

    // METHOD 1
    /*
       for (i=2; i; i--){
    FHMRDin[i-1] = betaFHM * FHMin[i-1]; // Distribute the operational forces (method 1)
    FHMNRin[i-1] = FHMin[i-1] - FHMRDin[i-1];
       }

      // create augmented operational force vectors
      FHMp6[0] = FHMNRin[0]; // FNRx
      FHMp6[1] = FHMNRin[1]; // FNRy
      FHMp6[2] = FHMRDin[0]; // FRDx
      FHMp6[3] = FHMRDin[1]; // FRDy
      FHMp6[4] = FHMin[2];   // Thip
      FHMp6[5] = FHMin[3];   // TRDfoot

        MatVectMult(THMtemp, &Jp6T[0][0], FHMp6,6,6); // Compute temp joint torque vectors
        // THMtemp = [TNRankle TNRknee TNRhip TRDankle TRDknee TRDhip]
    */

    // METHOD 2 05 27 03  ---OLD
    /*
    for (i=1; i<3; i++){
    FHMRDin[i] = betaFHM * FHMin[i]; // Distribute the torque and vert. operational forces
    FHMNRin[i] = FHMin[i] - FHMRDin[i];
    }
    */

    // ********************for DEBUGGING 2004-09-08*******
    betaFHM = 0.5;
    //*************************************************


    // ----------------------NEW -------------------------------06 04 03 METHOD A
    for (i=1; i<3; i++){
        if(NRside == 'L'){
            FHMNRin[i] = betaFHM * FHMin[i]; // Distribute the torque and vert. operational forces
            FHMRDin[i] = FHMin[i] - FHMNRin[i];

        }else{
            FHMRDin[i] = betaFHM * FHMin[i]; // Distribute the torque and vert. operational forces
            FHMNRin[i] = FHMin[i] - FHMRDin[i];
        }
    }

    FHMp6[0] = FHMin[0];   // Fx  // create augmented operational force vectors
       FHMp6[1] = FHMNRin[1]; // FNRy
    FHMp6[2] = FHMRDin[1]; // FRDy
    FHMp6[3] = FHMNRin[2]; // TNRhip
    FHMp6[4] = FHMRDin[2]; // TRDhip
    FHMp6[5] = FHMin[3];   // TRDfoot

       GetJp6Tnew(Jp6T, bodyData, heelContact, trigRD, trigNR); // Compute the new system Jacobian transpose

    MatVectMult(THMtemp, &Jp6T[0][0], FHMp6,6,6); // Compute temp joint torque vectors
```

```
//
//    if ( THMtemp[0]<0 ) {// If ankle torque of the non-redundant leg is negative use alternate solution
//
//        // Make operational force 6x1 vectors
//        FHMp5[0] = FHMNRin[0];  // FNRx
//        FHMp5[1] = FHMRDin[0];  // FRDx
//        FHMp5[2] = FHMin[1];    // Fy
//        FHMp5[3] = FHMin[2];    // Thip
//        FHMp5[4] = FHMin[3];     // TRDfoot
//
//        GetJp5xT(Jp5xT, bodyData, heelContact, trigRD, trigNR); // Compute the new system Jacobian transpose
//        MatVectMult(&THMtemp[1], &Jp5xT[0][0], FHMp5, 5, 5);
//
//        THMtemp[0] = 0; // Set the corresponding ankle torque to zero
//
//        Fgp5[0] = FgNRin[0];
//        Fgp5[1] = FgRDin[0];
//        Fgp5[2] = Fgin[1];
//        Fgp5[3] = Fgin[2];
//        Fgp5[4] = Fgin[3];
//
//        MatVectMult(&Tgtemp[1], &Jp5xT[0][0], Fgp5, 5, 5);    // Compute the required joint torques
//        Tgtemp[0] = 0;     // Set the corresponding ankle torque to zero
//
//    } else if ( THMtemp[3]<0) { // if redundant ankle torque is negative use alternate solution
//
//        // Make operational force 6x1 vectors
//        FHMp5[0] = FHMNRin[0];  // FNRx
//        FHMp5[1] = FHMRDin[0];  // FRDx
//        FHMp5[2] = FHMin[1];    // Fy
//        FHMp5[3] = FHMin[2];    // Thip
//        FHMp5[4] = FHMin[3];     // TRDfoot
//
//        GetJp5xxT(Jp5xxT, bodyData, heelContact, trigRD, trigNR); // Compute the new system Jacobian transpose
//        MatVectMult(THM5xx, &Jp5xxT[0][0], FHMp5, 5, 5);
//
//        THMtemp[0] = THM5xx[0];
//        THMtemp[1] = THM5xx[1];
//        THMtemp[2] = THM5xx[2];
//        THMtemp[3] = 0;              // Set this ankle torque to zero
//        THMtemp[4] = THM5xx[3];
//        THMtemp[5] = THM5xx[4];
//
//        //compute gravity torques for feedback linearization
//        Fgp5[0] = FgNRin[0];
//        Fgp5[1] = FgRDin[0];
//        Fgp5[2] = Fgin[1];
//        Fgp5[3] = Fgin[2];
//        Fgp5[4] = Fgin[3];
//
//        MatVectMult(Tg5xx, &Jp5xxT[0][0], Fgp5, 5, 5);    // Compute the required joint torques
//
//        Tgtemp[0] = Tg5xx[0];
//        Tgtemp[1] = Tg5xx[1];
//        Tgtemp[2] = Tg5xx[2];
//        Tgtemp[3] = 0;             // Set this ankle torque to zero
//        Tgtemp[4] = Tg5xx[3];
//        Tgtemp[5] = Tg5xx[4];
//    }
//
//    if ( THMtemp[0]<0 && THMtemp[3]<0) { // if both ankle torques are negative use alternate solution
//        // Use FHMin 4x1 operational force vector FHMin = [Fx Fy Thip TRDfoot]
//        // Compute the new system Jacobian transpose
//        GetJp4xT(Jp4xT, RDangles, NRangles, bodyData, heelContact, trigRD, trigNR);
//        MatVectMult(THM4, &Jp4xT[0][0], FHMin, 4, 4);
//
//        THMtemp[0] = 0;                // Set this ankle torque to zero
//        THMtemp[1] = THM4[0];
//        THMtemp[2] = THM4[1];
//        THMtemp[3] = 0;                // Set this ankle torque to zero
//        THMtemp[4] = THM4[2];
//        THMtemp[5] = THM4[3];
//
//        //compute gravity torques for feedback linearization
//        MatVectMult(Tg4, &Jp4xT[0][0], Fgin, 4, 4);    // Compute the required joint torques
//
//        Tgtemp[0] = 0;             // Set this ankle torque to zero
//        Tgtemp[1] = Tg4[0];
//        Tgtemp[2] = Tg4[1];
//        Tgtemp[3] = 0;              // Set this ankle torque to zero
//        Tgtemp[4] = Tg4[2];
//        Tgtemp[5] = Tg4[3];
//    }

if (redundantLeg == LEFT) { // assign corresponding values to joint torque vectors

    sensorData->jointData[LTOE].Tcc   = TccRD[0]; // velocity-induced force
    sensorData->jointData[LANKLE].Tcc = TccRD[1];
    sensorData->jointData[LKNEE].Tcc  = TccRD[2];
    sensorData->jointData[LHIP].Tcc   = TccRD[3];

    sensorData->jointData[RTOE].Tcc   = 0;
    sensorData->jointData[RANKLE].Tcc = TccNR[0];
    sensorData->jointData[RKNEE].Tcc  = TccNR[1];
    sensorData->jointData[RHIP].Tcc   = TccNR[2];

    sensorData->jointData[LTOE].Tg   = TgRD[0]; // gravity induced force
```

```
    sensorData->jointData[LANKLE].Tg = TgRD[1];
    sensorData->jointData[LKNEE].Tg  = TgRD[2];
    sensorData->jointData[LHIP].Tg   = TgRD[3];

    sensorData->jointData[RTOE].Tg   = 0;
    sensorData->jointData[RANKLE].Tg = TgNR[0];
    sensorData->jointData[RKNEE].Tg  = TgNR[1];
    sensorData->jointData[RHIP].Tg   = TgNR[2];

    sensorData->jointData[LTOE].Tf   = 0; // friction-induced force
    sensorData->jointData[LANKLE].Tf = TfRD[1];
    sensorData->jointData[LKNEE].Tf  = TfRD[2];
    sensorData->jointData[LHIP].Tf   = TfRD[3];

    sensorData->jointData[RTOE].Tf   = 0;
    sensorData->jointData[RANKLE].Tf = TfNR[0];
    sensorData->jointData[RKNEE].Tf  = TfNR[1];
    sensorData->jointData[RHIP].Tf   = TfNR[2];

    sensorData->jointData[LTOE].Tlin   = 0;
    sensorData->jointData[LANKLE].Tlin = Tgtemp[3]; // left leg feedback linearization torque
    sensorData->jointData[LKNEE].Tlin  = Tgtemp[4];
    sensorData->jointData[LHIP].Tlin   = Tgtemp[5];

    sensorData->jointData[RTOE].Tlin   = 0;
    sensorData->jointData[RANKLE].Tlin = Tgtemp[0]; // right leg
    sensorData->jointData[RKNEE].Tlin  = Tgtemp[1];
    sensorData->jointData[RHIP].Tlin   = Tgtemp[2];

    sensorData->jointData[LTOE].Thm   = 0;
    sensorData->jointData[LANKLE].Thm = 0; //0 - RDtorques[0]; //THMtemp[3]; // left leg human-machine torque
    sensorData->jointData[LKNEE].Thm  = 0; //0 - RDtorques[1]; //THMtemp[4];
    sensorData->jointData[LHIP].Thm   = 0; //0 - RDtorques[2]; //THMtemp[5];

    sensorData->jointData[RTOE].Thm   = 0;
    sensorData->jointData[RANKLE].Thm = 0; //0 - NRtorques[0]; //THMtemp[0]; // right leg
    sensorData->jointData[RKNEE].Thm  = 0; //0 - NRtorques[1]; //THMtemp[1];
    sensorData->jointData[RHIP].Thm   = 0; //0 - NRtorques[2]; //THMtemp[2];
}else{  //if the right leg is redundant
    sensorData->jointData[LTOE].Tcc   = 0; // velocity-induced force
    sensorData->jointData[LANKLE].Tcc = TccNR[0];
    sensorData->jointData[LKNEE].Tcc  = TccNR[1];
    sensorData->jointData[LHIP].Tcc   = TccNR[2];

    sensorData->jointData[RTOE].Tcc   = TccRD[0];;
    sensorData->jointData[RANKLE].Tcc = TccRD[1];
    sensorData->jointData[RKNEE].Tcc  = TccRD[2];
    sensorData->jointData[RHIP].Tcc   = TccRD[3];

    sensorData->jointData[LTOE].Tg   = 0; // gravity
    sensorData->jointData[LANKLE].Tg = TgNR[0];
    sensorData->jointData[LKNEE].Tg  = TgNR[1];
    sensorData->jointData[LHIP].Tg   = TgNR[2];

    sensorData->jointData[RTOE].Tg   = TgRD[0];
    sensorData->jointData[RANKLE].Tg = TgRD[1];
    sensorData->jointData[RKNEE].Tg  = TgRD[2];
    sensorData->jointData[RHIP].Tg   = TgRD[3];

    sensorData->jointData[LTOE].Tf   = 0; // friction
    sensorData->jointData[LANKLE].Tf = TfNR[0];
    sensorData->jointData[LKNEE].Tf  = TfNR[1];
    sensorData->jointData[LHIP].Tf   = TfNR[2];

    sensorData->jointData[RTOE].Tf   = 0;
    sensorData->jointData[RANKLE].Tf = TfRD[1];
    sensorData->jointData[RKNEE].Tf  = TfRD[2];
    sensorData->jointData[RHIP].Tf   = TfRD[3];

    sensorData->jointData[LTOE].Tlin   = 0;
    sensorData->jointData[LANKLE].Tlin = Tgtemp[LANKLE_T]; // left leg feedback linearization torque
    sensorData->jointData[LKNEE].Tlin  = Tgtemp[LKNEE_T];
    sensorData->jointData[LHIP].Tlin   = Tgtemp[LHIP_T];

    sensorData->jointData[RTOE].Tlin   = 0;
    sensorData->jointData[RANKLE].Tlin = Tgtemp[RANKLE_T]; //  right leg
    sensorData->jointData[RKNEE].Tlin  = Tgtemp[RKNEE_T];
    sensorData->jointData[RHIP].Tlin   = Tgtemp[RHIP_T];

    sensorData->jointData[LTOE].Thm   = 0;
    sensorData->jointData[LANKLE].Thm = 0; //0 - NRtorques[0]; //THMtemp[LANKLE_T]; // left leg human-machine torque
    sensorData->jointData[LKNEE].Thm  = 0; //0 - NRtorques[1]; //THMtemp[LKNEE_T];
    sensorData->jointData[LHIP].Thm   = 0; //0 - NRtorques[2]; //THMtemp[LHIP_T];

    sensorData->jointData[RTOE].Thm   = 0;
    sensorData->jointData[RANKLE].Thm = 0; //0 - RDtorques[0]; //THMtemp[RANKLE_T]; // right leg
    sensorData->jointData[RKNEE].Thm  = 0; //0 - RDtorques[1]; //THMtemp[RKNEE_T];
    sensorData->jointData[RHIP].Thm   = 0; //0 - RDtorques[2]; //THMtemp[RHIP_T];
}


// update distrData array for GUI
if (redundantLeg == LEFT) {
    distrData->LankleDistance = dRD; // transverse plane distance from CG to ankles
    distrData->RankleDistance = dNR;
}else{
    distrData->LankleDistance = dNR;
    distrData->RankleDistance = dRD;
}
```

```
    distrData->weightDistrFactor = alpha;
}


/* Function: Get4dofTorques
* ------------------------------------------------------------------------------------------
* Calculates the joint torques vector due to the human for a 4dof leg, and updates THM.
* The vector is as follows:[Tankle Tknee Thip]
*/
void Get4dofTorques(double               *Tg,
                    double               *Tcc,
                    double               *Tf,
                    double               *THM,
                    const double        *angles,
                    const double        *velocities,
                    const double        *accelerations,
                    const double        *torques,
                    const BodyDataT        *bodyData,
                    const int            heelContact,
                    const double        *trig,
                    const char             side,
                    const SensorDataT    *sensorData){

    double mf, If, Lf, LGf, hGf, ms, Is, Ls, LGs, hGs, mt, It, Lt, LGt, hGt, mub, Iub,LGub, hGub;  // segment properties
    double dq1, dq2,dq3,dq4, dq123, dq12, dq1234, ddq1, ddq2,ddq3,ddq4;
    double c1,s1,c12,c123,c1234,s12,s123,s1234,p1,p2, p3,p4,p5,p6,p7,p8,p9, p10;

    if (heelContact == 0){
        Lf = bodyData->foot.length;
        LGf = bodyData->foot.Lcg;
        hGf = bodyData->foot.hcg;
    } else {
        Lf = bodyData->heel.length;
        LGf = bodyData->heel.Lcg;
        hGf = bodyData->heel.hcg;
    }

    mf   = bodyData->foot.mass; // get mass and geometric properties
    If   = bodyData->foot.inertia;
    ms   = bodyData->shank.mass;
    Is   = bodyData->shank.inertia;
    Ls   = bodyData->shank.length;
    LGs  = bodyData->shank.Lcg;
    hGs  = bodyData->shank.hcg;
    mt   = bodyData->thigh.mass;
    It   = bodyData->thigh.inertia;
    Lt   = bodyData->thigh.length;
    LGt  = bodyData->thigh.Lcg;
    hGt  = bodyData->thigh.hcg;
    mub  = bodyData->upperBody.mass;
    Iub  = bodyData->upperBody.inertia;
    LGub = bodyData->upperBody.Lcg;
    hGub = bodyData->upperBody.hcg;

    ddq1   = accelerations[0];
    ddq2   = accelerations[1];
    ddq3   = accelerations[2];
    ddq4   = accelerations[3];
    dq1    = velocities[0];
    dq2    = velocities[1];
    dq3    = velocities[2];
    dq4    = velocities[3];
    dq12   = dq1 + dq2; dq123 = dq12 + dq3;
    dq1234 = dq123 + dq4;

    c1    = trig[C1RD];            s1    = trig[S1RD]; // get pre-computed trigonometric functions
    c12   = trig[C12RD];        s12   = trig[S12RD];
    c123  = trig[C123RD];        s123  = trig[S123RD];
    c1234 = trig[C1234RD];        s1234 = trig[S1234RD];

    // joint friction and stiffness
    GetTfrictionRedtLeg(&Tf[1], &angles[1], side, sensorData); // get joint friction and stiffness torques
    Tf[0] = 0; // assume no friction at toe

    // compute reoccurring terms
    p1  =  Lf*s1-Ls*c12-Lt*c123-LGub*c1234-hGub*s1234; // 4dof_KE1_NoVelSubs.txt
    p2  = -Lf*c1-Ls*s12-Lt*s123-LGub*s1234+hGub*c1234;
       p3  = -LGub*s1234+hGub*c1234-Ls*s12-Lt*s123; // 4dof_KE2_NoVelSubs.txt
    p4  = -LGub*c1234-hGub*s1234-Ls*c12-Lt*c123;
       p5  = -LGub*c1234-hGub*s1234-Lt*c123; // 4dof_KE3_NoVelSubs.txt
    p6  = -LGub*s1234+hGub*c1234-Lt*s123;
    p7  = -LGub*c1234-hGub*s1234; // 4dof_KE4_NoVelSubs.txt
    p8  = -LGub*s1234+hGub*c1234;
    p9  = -Lf*c1-Ls*s12-LGt*s123+hGt*c123;
    p10 =  Lf*s1-Ls*c12-LGt*c123-hGt*s123;

    // torques from SingleRedModel2dof.m
    // Centrifugal and coriolis terms

    Tcc[0] = 0.5*mf*(4*dq1*((-Lf+LGf)*s1+hGf*c1)*((-Lf+LGf)*c1*dq1-hGf*s1*dq1)+4*dq1*((-Lf+LGf)*c1-hGf*s1)*(-(-
Lf+LGf)*s1*dq1-hGf*c1*dq1))+0.25*mub*(2*(dq12*dq12
        *Ls*s12+dq1*dq1*Lf*c1-(dq1234)*(-LGub*s1234*(dq1234)+hGub*c1234*(dq1234))+dq123*dq123*Lt*s123)*(Lf*s1-Ls*c12-
Lt*c123-LGub*c1234
        -hGub*s1234)+2*(-Lf*c1*dq1-Ls*s12*(dq12)-Lt*s123*(dq123)+(dq1234)*(p8))*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*Lt*c123-
LGub
        *c1234*(dq1234)-hGub*s1234*(dq1234))+2*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*Lt*c123-
(dq1234)*(LGub*c1234+hGub*s1234))*(Lf*c1*dq1+Ls*s12*(dq12)
```

```
        +Lt*s123*(dq123)+LGub*s1234*(dq1234)-hGub*c1234*(dq1234))+2*(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+(dq1234)*(-
LGub*c1234*(dq1234)-hGub*s1234
        *(dq1234))-dq123*dq123*Lt*c123)*(p2))-0.25*mub*(2*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*Lt*c123
        -(dq1234)*(LGub*c1234+hGub*s1234))*(Lf*c1*dq1+Ls*s12*(dq12)+Lt*s123*(dq123)-(dq1234)*(p8))+2*(-Lf*c1*dq1-
Ls*s12*(dq12)
        -Lt*s123*(dq123)+(dq1234)*(p8))*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*Lt*c123+(dq1234)*(p7)))-0.5*mt
        *(2*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*(LGt*c123+hGt*s123))*(Lf*c1*dq1+Ls*s12*(dq12)-(dq123)*(-
LGt*s123+hGt*c123))+2*(-Lf*c1*dq1-Ls*s12
        *(dq12)+(dq123)*(-LGt*s123+hGt*c123))*(dq1*Lf*s1-(dq12)*Ls*c12+(dq123)*(-LGt*c123-hGt*s123)))-0.5*mf*(2*dq1*dq1*((-
Lf+LGf)*s1+hGf*c1)
        *((-Lf+LGf)*c1-hGf*s1)+2*dq1*dq1*((-Lf+LGf)*c1-hGf*s1)*(-(-Lf+LGf)*s1-hGf*c1))+0.5*ms*(2*(-(dq12)*(-
LGs*s12*(dq12)+hGs*c12*(dq12))+dq1*dq1
        *Lf*c1)*(Lf*s1-LGs*c12-hGs*s12)+2*(-Lf*c1*dq1+(dq12)*(-LGs*s12+hGs*c12))*(dq1*Lf*s1-LGs*c12*(dq12)-
hGs*s12*(dq12))+2*(dq1*Lf*s1-(dq12)
        *(LGs*c12+hGs*s12))*(Lf*c1*dq1+LGs*s12*(dq12)-hGs*c12*(dq12))+2*((dq12)*(-LGs*c12*(dq12)-
hGs*s12*(dq12))+dq1*dq1*Lf*s1)*(-Lf*c1-LGs
        *s12+hGs*c12))+0.5*mt*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-(dq123)*(-LGt*s123*(dq123)+hGt*c123*(dq123)))*(Lf*s1-
Ls*c12-LGt*c123-hGt
        *s123)+2*(-Lf*c1*dq1-Ls*s12*(dq12)+(dq123)*(-LGt*s123+hGt*c123))*(dq1*Lf*s1-(dq12)*Ls*c12-LGt*c123*(dq123)-
hGt*s123*(dq123))+2*(dq1*Lf
        *s1-(dq12)*Ls*c12-(dq123)*(LGt*c123+hGt*s123))*(Lf*c1*dq1+Ls*s12*(dq12)+LGt*s123*(dq123)-hGt*c123*(dq123))+2*(-
dq12*dq12*Ls*c12+dq1*dq1
        *Lf*s1+(dq123)*(-LGt*c123*(dq123)-hGt*s123*(dq123)))*(p9))-0.5*ms*(2*(dq1*Lf*s1-(dq12)*(LGs*c12+hGs*s12))
        *(Lf*c1*dq1-(dq12)*(-LGs*s12+hGs*c12))+2*(-Lf*c1*dq1+(dq12)*(-LGs*s12+hGs*c12))*(dq1*Lf*s1+(dq12)*(-LGs*c12-
hGs*s12)));

    Tcc[1] = 0.5*ms*(2*(-dq12*(-LGs*s12
        *dq12+hGs*c12*dq12)+dq1*dq1*Lf*c1)*(-LGs*c12-hGs*s12)+2*(-dq1*Lf*c1+dq12*(-LGs*s12+hGs*c12))*(-LGs*c12*dq12-hGs*s12
        *dq12)+2*(dq1*Lf*s1-dq12*(LGs*c12+hGs*s12))*(LGs*s12*dq12-hGs*c12*dq12)+2*(dq12*(-LGs*c12*dq12-hGs*s12*(dq2
        +dq1))+dq1*dq1*Lf*s1)*(-LGs*s12+hGs*c12))+0.25*mub*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-dq1234*(-
LGub*s1234*(dq4+dq3+dq2
        +dq1)+hGub*c1234*dq1234)+dq123*dq123*Lt*s123)*(-Ls*c12-Lt*c123+p7)+2*(-dq1*Lf*c1-Ls*s12*dq12
        -Lt*s123*dq123+dq1234*(p8))*(-dq12*Ls*c12-dq123*Lt*c123-LGub*c1234*dq1234-hGub
        *s1234*dq1234)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123-dq1234*(LGub*c1234+hGub*s1234))*(Ls*s12*dq12
        +Lt*s123*dq123+LGub*s1234*dq1234-hGub*c1234*dq1234)+2*(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq1234
        *(-LGub*c1234*dq1234-hGub*s1234*dq1234)-dq123*dq123*Lt*c123)*(-Ls*s12-Lt*s123+p8))-0.5*ms
        *(-2*(dq1*Lf*s1-dq12*(LGs*c12+hGs*s12))*dq12*(-LGs*s12+hGs*c12)+2*(-dq1*Lf*c1+dq12*(-LGs*s12+hGs*c12))*dq12*(-
LGs*c12
        -hGs*s12))-0.25*mub*(2*(dq1*Lf*s1-dq12*Ls*c12-dq123*Lt*c123-dq1234*(LGub*c1234+hGub*s1234))*(Ls*s12*dq12+Lt*s123
        *dq123-dq1234*(p8))+2*(-dq1*Lf*c1-Ls*s12*dq12-Lt*s123*dq123+dq1234*(-LGub*s1234
        +hGub*c1234))*(-dq12*Ls*c12-dq123*Lt*c123+dq1234*(p7)))-0.5*mt*(2*(dq1*Lf*s1-dg12*Ls*c12
        -dq123*(LGt*c123+hGt*s123))*(Ls*s12*dq12-dq123*(-LGt*s123+hGt*c123))+2*(-dq1*Lf*c1-Ls*s12*dq12+dq123*(-LGt
        *s123+hGt*c123))*(-dq12*Ls*c12+dq123*(-LGt*c123-hGt*s123)))+0.5*mt*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-dq123
        *(-LGt*s123*dq123+hGt*c123*dq123))*(-Ls*c12-LGt*c123-hGt*s123)+2*(-dq1*Lf*c1-Ls*s12*dq12+dq123*(-LGt*s123+hGt
        *c123))*(-dq12*Ls*c12-LGt*c123*dq123-hGt*s123*dq123)+2*(dq1*Lf*s1-dq12*Ls*c12-dq123*(LGt*c123+hGt*s123))
        *(Ls*s12*dq12+LGt*s123*dq123-hGt*c123*dq123)+2*(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+dq123*(-LGt*c123*(dq3
        +dq2+dq1)-hGt*s123*dq123))*(-Ls*s12-LGt*s123+hGt*c123));

    Tcc[2] = +0.5*mt*(2
        *(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-(dq123)*(-LGt*s123*(dq123)+hGt*c123*(dq123)))*(-LGt*c123-hGt*s123)+2*(-dq1*Lf*c1-
(dq12)*Ls*s12+(dq123)
        *(-LGt*s123+hGt*c123))*(-LGt*c123*(dq123)-hGt*s123*(dq123))+2*(dq1*Lf*s1-(dq12)*Ls*c12-
(dq123)*(LGt*c123+hGt*s123))*(LGt*s123*(dq123)
        -hGt*c123*(dq123))+2*(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+(dq123)*(-LGt*c123*(dq123)-hGt*s123*(dq123)))*(-
LGt*s123+hGt*c123))+0.25*mub*(2
        *(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-(dq1234)*(-LGub*s1234*(dq1234)+hGub*c1234*(dq1234))+dq123*dq123*Lt*s123)*(-
Lt*c123-LGub*c1234-hGub
        *s1234)+2*(-dq1*Lf*c1-(dq12)*Ls*s12-Lt*s123*(dq123)+(dq1234)*(p8))*(-(dq123)*Lt*c123-LGub*c1234*(dq1234)-hGub*s1234
        *(dq1234))+2*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*Lt*c123-
(dq1234)*(LGub*c1234+hGub*s1234))*(Lt*s123*(dq123)+LGub*s1234*(dq1234)-hGub*c1234
        *(dq1234))+2*(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+(dq1234)*(-LGub*c1234*(dq1234)-hGub*s1234*(dq1234))-
dq123*dq123*Lt*c123)*(-Lt*s123-LGub
        *s1234+hGub*c1234))-0.25*mub*(2*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*Lt*c123-
(dq1234)*(LGub*c1234+hGub*s1234))*(Lt*s123*(dq123)-(dq1234)*(-LGub
        *s1234+hGub*c1234))+2*(-dq1*Lf*c1-(dq12)*Ls*s12-Lt*s123*(dq123)+(dq1234)*(p8))*(-(dq123)*Lt*c123+(dq1234)*(-
LGub*c1234
        -hGub*s1234)))-0.5*mt*(-2*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*(LGt*c123+hGt*s123))*(dq123)*(-LGt*s123+hGt*c123)+2*(-
dq1*Lf*c1-(dq12)*Ls*s12
        +(dq123)*(-LGt*s123+hGt*c123))*(dq123)*(-LGt*c123-hGt*s123));

    Tcc[3] = 0.25*mub*(2*(dq12*dq12*Ls*s12+dq1*dq1*Lf*c1-(dq1234)*(-LGub*s1234*(dq1234)+hGub*c1234*(dq1234))+dq123*dq123
        *Lt*s123)*(p7)+2*(-dq1*Lf*c1-(dq12)*Ls*s12-(dq123)*Lt*s123+(dq1234)*(p8))*(-LGub*c1234*(dq1234)
        -hGub*s1234*(dq1234))+2*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*Lt*s123-
(dq1234)*(LGub*c1234+hGub*s1234))*(LGub*s1234*(dq1234)-hGub*c1234*(dq1234))
        +2*(-dq12*dq12*Ls*c12+dq1*dq1*Lf*s1+(dq1234)*(-LGub*c1234*(dq1234)-hGub*s1234*(dq1234))-dq123*dq123*Lt*c123)*(p8))
        -0.25*mub*(-2*(dq1*Lf*s1-(dq12)*Ls*c12-(dq123)*Lt*c123-(dq1234)*(LGub*c1234+hGub*s1234))*(dq1234)*(p8)+2*(-
dq1*Lf*c1
        -(dq12)*Ls*s12-(dq123)*Lt*s123+(dq1234)*(p8))*(dq1234)*(p7));


    // gravity torques

    Tg[0] = -(-mf*g*((-Lf+LGf)*c1-hGf*s1)-ms*g*(-Lf*c1-LGs*s12+hGs*c12)-mt*g*(p9)-0.5*mub*g*(-Lf*c1-Ls*s12-Lt*s123+p8)); //
negative of toe/heel (foot ground joint) V1_4dof.txt
    Tg[1] = -(-ms*g*(-LGs*s12+hGs*c12)-mt*g*(-Ls*s12-LGt*s123+hGt*c123)-0.5*mub*g*(-Ls*s12-Lt*s123+p8)); // negative of
ankle V2_4dof.txt
    Tg[2] = -(-mt*g*(-LGt*s123+hGt*c123)-0.5*mub*g*(-Lt*s123+p8));  // negative of knee V3_4dof.txt
    Tg[3] = -(-0.5*mub*g*(p8)); // negative of hip V4_4dof.txt

    THM[0] =  (0.25*mub*(2*(p7)*(p1)+2*(p8)*(-Lf*c1-Ls*s12-Lt*s123+p8))+0.5*Iub)*ddq4+(0.25*mub*(2*(p5)*(p1)+2*(
        p6)*(p2))+0.5*Iub+It+0.5*mt*(2*(-LGt*c123-hGt*s123)*(Lf*s1-Ls
        *c12-LGt*c123-hGt*s123)+2*(-LGt*s123+hGt*c123)*(p9)))*ddq3+(0.5*mt*(2*(-Ls*c12-LGt*c123-hGt*s123)*(Lf
        *s1-Ls*c12-LGt*c123-hGt*s123)+2*(-Ls*s12-LGt*s123+hGt*c123)*(p9))+It+0.5*ms*(2*(-LGs*c12-hGs*s12)
        *(Lf*s1-LGs*c12-hGs*s12)+2*(-LGs*s12+hGs*c12)*(-Lf*c1-LGs*s12+hGs*c12))+0.5*Iub+Is+0.25*mub*(2*(p4)
        *(p1)+2*(p3)*(-Lf*c1-Ls*s12-Lt*s123-LGub*s1234+hGub
        *c1234)))*ddq2+(Is+0.5*mt*(2*p9*p9+2*p10*p10)+0.5*mf*(2*((-Lf+LGf)*s1+hGf*c1)*((-Lf+LGf)*s1+hGf*c1)+2*((-
Lf+LGf)*c1-hGf*s1)*((-Lf+LGf)*c1-hGf*s1))+It+If+0.25*mub*(2*p2*p2
        +2*(Lf*s1-Ls*c12-Lt*c123+p7)*(Lf*s1-Ls*c12-Lt*c123+p7))+0.5*ms*(2*(-Lf*c1-LGs*s12+hGs*c12)*(-Lf*c1-
LGs*s12+hGs*c12)+2*(Lf*s1-LGs*c12-hGs*s12)*(Lf*s1-LGs*c12-hGs*s12))+0.5*Iub)*ddq1
```

```
        + Tg[0] + Tcc[0]; // toe KE1_4dof.txt

    THM[1] = (0.25*mub*(2*(p7)*(-Ls*c12-Lt*c123+p7)+2*(p8)*(-Ls*s12-Lt*s123-LGub*s1234
        +hGub*c1234))+0.5*Iub)*ddq4+(0.25*mub*(2*(p5)*(-Ls*c12-Lt*c123+p7)+2*(p8
        -Lt*s123)*(-Ls*s12-Lt*s123+p8))+0.5*mt*(2*(-LGt*c123-hGt*s123)*(-Ls*c12-LGt*c123-hGt*s123)+2*(-LGt*s123+hGt*c123)
        *(-Ls*s12-LGt*s123+hGt*c123))+It+0.5*Iub)*ddq3+(It+0.5*ms*(2*(-LGs*c12-hGs*s12)*(-LGs*c12-hGs*s12)+2*(-
LGs*s12+hGs*c12)*(-LGs*s12+hGs*c12))
        +0.25*mub*(2*(-Ls*c12-Lt*c123+p7)*(-Ls*c12-Lt*c123+p7)+2*(-Ls*s12-Lt*s123+p8)*(-Ls*s12-
Lt*s123+p8))+Is+0.5*Iub+0.5*mt*(2*(-Ls*c12-LGt*c123-hGt*s123)*(-Ls*c12-LGt*c123-hGt*s123)
        +2*(-Ls*s12-LGt*s123+hGt*c123)*(-Ls*s12-LGt*s123+hGt*c123)))*ddq2+(Is+It+0.5*mt*(2*(-Lf*c1-LGt*s123+hGt*c123-
Ls*s12)*(-Ls*s12-LGt*s123+hGt*c123)+2*(Lf*s1
        -LGt*c123-hGt*s123-Ls*c12)*(-Ls*c12-LGt*c123-hGt*s123))+0.5*Iub+0.25*mub*(2*(-Ls*s12-Lf*c1+p6)*(-Ls*s12
        -Lt*s123+p8)+2*(-Ls*c12+Lf*s1+p5)*(-Ls*c12-Lt*c123+p7))+0.5*ms
        *(2*(-LGs*s12+hGs*c12-Lf*c1)*(-LGs*s12+hGs*c12)+2*(-LGs*c12-hGs*s12+Lf*s1)*(-LGs*c12-hGs*s12)))*ddq1
        + Tg[1] + Tcc[1] + Tf[1] - torques[0]; // ankle KE2_4dof.txt

    THM[2] = (0.25*mub*(2*(p7)*(-Lt*c123+p7)+2*(p8)*(-Lt*s123+p8))+0.5
        *Iub)*ddq4+(0.5*mt*(2*(-LGt*c123-hGt*s123)*(-LGt*c123-hGt*s123)+2*(-LGt*s123+hGt*c123)*(-
LGt*s123+hGt*c123))+It+0.25*mub*(2*(-Lt*c123+p7)*(-Lt*c123+p7)
        +2*(-Lt*s123+p8)*(-Lt*s123+p8))+0.5*Iub)*ddq3+(It+0.25*mub*(2*(p4)*(-Lt*c123+p7)
        +2*(p3)*(-Lt*s123+p8))+0.5*Iub+0.5*mt*(2*(-Ls*c12-LGt*c123-hGt*s123)*(-LGt*c123
        -hGt*s123)+2*(-Ls*s12-LGt*s123+hGt*c123)*(-LGt*s123+hGt*c123)))*ddq2+(It+0.5*mt*(2*(-Lf*c1-LGt*s123+hGt*c123-
Ls*s12)*(-LGt*s123+hGt
        *c123)+2*(Lf*s1-LGt*c123-hGt*s123-Ls*c12)*(-LGt*c123-hGt*s123))+0.5*Iub+0.25*mub*(2*(-Ls*s12-Lf*c1+p6)
        *(-Lt*s123+p8)+2*(-Ls*c12+Lf*s1+p5)*(-Lt*c123+p7)))*ddq1 + Tg[2]+ Tcc[2] + Tf[2] - torques[1];  // knee
KE3_4dof.txt

    THM[3] = (0.25*mub*(2*p7*p7+2*p8*p8)+0.5*Iub)*ddq4+(0.25*mub*(2*(p5)*(p7)+2*(-
Lt*s123+p8)*(p8))+0.5*Iub)*ddq3+(0.25*mub*(2*(p7
        -Ls*c12-Lt*c123)*(p7)+2*(p3)*(p8))+0.5*Iub)*ddq2+(0.25*mub*(2*(-Ls*s12-Lf*c1+p6)*(p8)+2*(-Ls*c12+Lf*s1+p5)*(-LGub
        *c1234-hGub*s1234))+0.5*Iub)*ddq1 + Tg[3] + Tcc[3] + Tf[3] - torques[2];    // hip KE4_4dof.txt
}


/* Function: GetJT44 - ABZ 2004-10-05
* ------------------------------------------------
* Calculates the transpose jacobian matrix for a 4dof leg and updates JT.
* The jacobian transforms an opertional force of the form [Fx_hip Fy_hip Tz_hip Tz_foot] into a torque vector [Tfoot Tankl
Tknee Thip]
* T = JT44 F
*/
void GetJT44(double             JT[][4],
             const BodyDataT *bodyData,
             const double     *trig){

    double Lf, Ls, Lt, c12, s12, c23, c123, s123, s3, c2;

    Lf   = bodyData->foot.length;
    Ls   = bodyData->shank.length;
    Lt   = bodyData->thigh.length;
    c12  = trig[C12RD];
    s12  = trig[S12RD];
    c23  = trig[C23RD];
    c123 = trig[C123RD];
    s123 = trig[S123RD];
    s3   = trig[S3RD];
    c2   = trig[C2RD];

    //[row][col]
    JT[0][0] = -(c123*Lf*c2-Lf*c23*c12+Ls*s3*c12+s3*Lt*c123)/s3;
    JT[0][1] = -(s123*Lf*c2-Lf*c23*s12+Ls*s3*s12+s3*Lt*s123)/s3;
    JT[0][2] = 1;
    JT[0][3] = 1;

    JT[1][0] = -Ls*c12-Lt*c123;
    JT[1][1] = -Ls*s12-Lt*s123;
    JT[1][2] = 1;
    JT[1][3] = 0;

    JT[2][0] = -c123*Lt;
    JT[2][1] = -s123*Lt;
    JT[2][2] = 1;
    JT[2][3] = 0;

    JT[3][0] = 0;
    JT[3][1] = 0;
    JT[3][2] = 1;
    JT[3][3] = 0;
}


/* Function: GetJinvT44
* ------------------------------------------------------------------------------------------------
* Calculates the inverse transpose jacobian matrix JinT for a 4dof leg. F = JinvT T where T is the
*joint torque vector and F is the operational force [Fx@hip Fy@hip Tz@hip Tfoot] .
* Equation is obtained from JRinvT in SingleRedModel2dof.m
*/
void GetJinvT44(double            JinvT[][4],
                const BodyDataT *bodyData,
                const int        heelContact,
                const double    *trig){

    double s12, c12, s123, c123, s3, c2, c23, Lf, Ls, Lt, s3_div, Ls_div, Lt_div;

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    if (heelContact == 1){
        Lf = bodyData->heel.length;
```

```
    }else{
        Lf = bodyData->foot.length;
    }

    c2   = trig[C2RD];
    c12  = trig[C12RD];    // get pre-computed trigonometric functions
    c123 = trig[C123RD];
    c23  = trig[C23RD];

    s3   = trig[S3RD];
    s12  = trig[S12RD];
    s123 = trig[S123RD];

    s3_div = 1/s3;
    Ls_div = 1/Ls;
    Lt_div = 1/Lt;

    JinvT[0][0] =  0;
    JinvT[0][1] = -s123*Ls_div*s3_div;
    JinvT[0][2] =  (Ls*s12+Lt*s123)*Lt_div*Ls_div*s3_div;
    JinvT[0][3] = -s12*Lt_div*s3_div;
    JinvT[1][0] =  0;
    JinvT[1][1] =  c123*Ls_div*s3_div;
    JinvT[1][2] = -(Ls*c12+Lt*c123)*Lt_div*Ls_div*s3_div;
    JinvT[1][3] =  c12*Lt_div*s3_div;
    JinvT[2][0] =  0;
    JinvT[2][1] =  0;
    JinvT[2][2] =  0;
    JinvT[2][3] =  1;
    JinvT[3][0] =  1;
    JinvT[3][1] =  (Lf*c23-Ls*s3)*Ls_div*s3_div;
    JinvT[3][2] =  (-Lf*Ls*c2-Lf*Lt*c23)*Lt_div*Ls_div*s3_div;
    JinvT[3][3] =  Lf*c2*Lt_div*s3_div;
}

/* Function: GetJp6T
* ------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp6T for a double support system with one redundancy.
* Tp6 = Jp6T Fp6 is the joint torque vector and F is the operational force system vector
* F = [FLx@hip FLy@hip FRx@hip FRy@hip Tz@hip Tz@foot] .
* see J6xT in TestForceDistr.m
*/
void GetJp6T(double               Jp6T[][6],
             const BodyDataT    *bodyData,
             const int           heelContact,
             const double        *trigRD,
             const double        *trigNR){

    double c1RD, s1RD, c23NR, s23NR, c12RD, s12RD, c123RD, s123RD, c2NR, s2NR,Lf,Ls,Lt,p1,p2;

    static double unfilteredAngle[4] = {0,0,0,0}; // Current A[0] and Previous values
    static double filteredAngle[4] = {0,0,0,0};    // used in lowpass filter

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    if (heelContact == 1){
        Lf = bodyData->heel.length;
    }else{
        Lf = bodyData->foot.length;
    }
    c1RD   = trigRD[C1RD];        s1RD   = trigRD[S1RD]; // get pre-computed trigonometric functions
    c12RD  = trigRD[C12RD];       s12RD  = trigRD[S12RD];
    c123RD = trigRD[C123RD];    s123RD = trigRD[S123RD];
    c2NR   = trigNR[C2];        s2NR   = trigNR[S2];
    c23NR  = trigNR[C23];       s23NR  = trigNR[S23];

    p1 =  Lf*s1RD-Ls*c12RD-Lt*c123RD; // compute reoccurring terms
    p2 = -Lf*c1RD -Ls*s12RD -Lt*s123RD;

//    LowpassFilter(unfilteredAngle, filteredAngle, &p2, filterCoeffs2nd1); // 1 Hz filter (makes system stable but
vibrates)

    Jp6T[0][0] = -Ls*c2NR-Lt*c23NR;
    Jp6T[0][1] = -Ls*s2NR-Lt*s23NR;
    Jp6T[0][2] =  p1;
    Jp6T[0][3] =  p2;
    Jp6T[0][4] =  1;
    Jp6T[0][5] =  1;
    Jp6T[1][0] = -Lt*c23NR;
    Jp6T[1][1] = -Lt*s23NR;
    Jp6T[1][2] =  p1;
    Jp6T[1][3] =  p2;
    Jp6T[1][4] =  1;
    Jp6T[1][5] =  1;
    Jp6T[2][0] =  0;
    Jp6T[2][1] =  0;
    Jp6T[2][2] =  p1; // set to 0
    Jp6T[2][3] =  p2; // set to 0
    Jp6T[2][4] =  1;
    Jp6T[2][5] =  1;
    Jp6T[3][0] =  0;
    Jp6T[3][1] =  0;
    Jp6T[3][2] = -Lf*s1RD;
    Jp6T[3][3] =  Lf*c1RD;
    Jp6T[3][4] =  0;
    Jp6T[3][5] = -1;
    Jp6T[4][0] =  0;
    Jp6T[4][1] =  0;
```

```
    Jp6T[4][2] =  Ls*c12RD-Lf*s1RD;
    Jp6T[4][3] =  Lf*c1RD+Ls*s12RD;
    Jp6T[4][4] =  0;
    Jp6T[4][5] = -1;
    Jp6T[5][0] =  0;
    Jp6T[5][1] =  0;
    Jp6T[5][2] = -p1; // set to 0
    Jp6T[5][3] = -p2; // set to 0
    Jp6T[5][4] =  0;
    Jp6T[5][5] = -1;
}


/* Function: GetJp6Tnew   (may 27 03)
* ------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp6T for a double support system with one redundancy.
* Tp6 = Jp6T Fp6 is the joint torque vector and F is the operational force system vector
* F = [FLx@hip FLy@hip FRx@hip FRy@hip Tz@hip Tz@foot] .
* see J6xT in TestForceDistr.m
*/
void GetJp6Tnew(double           Jp6T[][6],
                const BodyDataT *bodyData,
                const int        heelContact,
                const double    *trigRD,
                const double    *trigNR){

    double c1RD, s1RD, c23NR, s23NR, c12RD, s12RD, c123RD, s123RD, c2NR, s2NR,Lf,Ls,Lt,p1,p2,p3;

    static double unfilteredAngle[4] = {0,0,0,0}; // Current A[0] and Previous values
    static double filteredAngle[4]   = {0,0,0,0}; // used in lowpass filter

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    if (heelContact == 1){
        Lf = bodyData->heel.length;
    }else{
        Lf = bodyData->foot.length;
    }
    c1RD   = trigRD[C1RD];         s1RD   = trigRD[S1RD]; // get pre-computed trigonometric functions
    c12RD  = trigRD[C12RD];        s12RD  = trigRD[S12RD];
    c123RD = trigRD[C123RD];     s123RD = trigRD[S123RD];
    c2NR   = trigNR[C2];         s2NR   = trigNR[S2];
    c23NR  = trigNR[C23];         s23NR  = trigNR[S23];

    p1 = 1/(Ls*c12RD+Lt*c123RD-Lf*s1RD); // compute reoccurring terms
    p2 = Lt*s123RD+Lf*c1RD+Ls*s12RD;
    p3 = Ls*c2NR+Lt*c23NR;

    Jp6T[0][0] = -Ls*c2NR-Lt*c23NR;
    Jp6T[0][1] = -Ls*s2NR-Lt*s23NR;
    Jp6T[0][2] = -p3*p2*p1;
    Jp6T[0][3] =  1;
    Jp6T[0][4] =  p3*p1;
    Jp6T[0][5] =  p3*p1;

    Jp6T[1][0] = -Lt*c23NR;
    Jp6T[1][1] = -Lt*s23NR;
    Jp6T[1][2] = -c23NR*p2*Lt*p1;
    Jp6T[1][3] =  1;
    Jp6T[1][4] =  c23NR*Lt*p1;
    Jp6T[1][5] =  c23NR*Lt*p1;

    Jp6T[2][0] =  0;
    Jp6T[2][1] =  0;
    Jp6T[2][2] =  0;
    Jp6T[2][3] =  1;
    Jp6T[2][4] =  0;
    Jp6T[2][5] =  0;

    Jp6T[3][0] =  0;
    Jp6T[3][1] =  0;
    Jp6T[3][2] =  Lf*(s1RD*Ls*s12RD+Lt*s123RD*s1RD+Ls*c12RD*c1RD+Lt*c123RD*c1RD)*p1;
    Jp6T[3][3] =  0;
    Jp6T[3][4] = -s1RD*Lf*p1;
    Jp6T[3][5] = -(Ls*c12RD+Lt*c123RD)*p1;

    Jp6T[4][0] =  0;
    Jp6T[4][1] =  0;
    Jp6T[4][2] =  (s123RD*Lf*s1RD-s123RD*Ls*c12RD+c123RD*Lf*c1RD+c123RD*Ls*s12RD)*Lt*p1;
    Jp6T[4][3] =  0;
    Jp6T[4][4] =  (Ls*c12RD-Lf*s1RD)*p1;
    Jp6T[4][5] = -c123RD*Lt*p1;

    Jp6T[5][0] =  0;
    Jp6T[5][1] =  0;
    Jp6T[5][2] =  0;
    Jp6T[5][3] =  0;
    Jp6T[5][4] =  1;
    Jp6T[5][5] =  0;
}


/* Function: GetJp5xT
* ------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp5T for a system with a zero ankle torque at the non-redundant leg.
* Tp5 = Jp5xT Fp5 is the joint torque vector and F is the operational force system vector
* Fp5 = [FNRx@hip FRDx@hip Fy@hip Tz@hip Tz@foot]
```

```
* see J6xT2 in TestForceDistr.m
*/
void GetJp5xT(double            Jp5xT[][5],
              const BodyDataT   *bodyData,
              const int           heelContact,
              const double        *trigRD,
              const double        *trigNR){

    double c2NR, s2NR, c23NR, s23NR, c1RD, s1RD, c12RD, s12RD, c123RD, s123RD, Lf, Ls, Lt, jx, p1, p2;

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    if (heelContact == 1){
        Lf = bodyData->heel.length;
    }else{
        Lf = bodyData->foot.length;
    }

    c2NR   = trigNR[C2];        s2NR   = trigNR[S2]; // get pre-computed trigonometric functions
    c23NR  = trigNR[C23];       s23NR  = trigNR[S23];
    c1RD   = trigRD[C1RD];      s1RD   = trigRD[S1RD];
    c12RD  = trigRD[C12RD];     s12RD  = trigRD[S12RD];
    c123RD = trigRD[C123RD];    s123RD = trigRD[S123RD];

    p1 = Ls*s2NR+Lt*s23NR;
    p2 = Ls*c2NR+Lt*c23NR;

    jx = 1/ ( Ls*s2NR-Lf*c1RD+Lt*s23NR-Lt*s123RD-Ls*s12RD); // compute recurring terms

    Jp5xT[0][0] = -(c2NR*Lt*s123RD+c2NR*Ls*s12RD+c2NR*Lf*c1RD-Lt*s23NR*c2NR
        +Lt*c23NR*s2NR)*Ls*jx;
    Jp5xT[0][1] =  Ls*(Lf*s1RD-Ls*c12RD-Lt*c123RD)*s2NR*jx;
    Jp5xT[0][2] = -Ls*(Lt*s123RD+Ls*s12RD+Lf*c1RD)*s2NR*jx;
    Jp5xT[0][3] =  Ls*s2NR * jx;
    Jp5xT[0][4] =  Ls*s2NR * jx;
    Jp5xT[1][0] = -(Lt*s123RD+Ls*s12RD+Lf*c1RD)*p2*jx;
    Jp5xT[1][1] =  (Lf*s1RD-Ls*c12RD-Lt*c123RD)*p1*jx;
    Jp5xT[1][2] = -(Lt*s123RD+Ls*s12RD+Lf*c1RD)*p1*jx;
    Jp5xT[1][3] =  p1*jx;
    Jp5xT[1][4] =  p1*jx;
    Jp5xT[2][0] =  Lf*c1RD*p2*jx;
    Jp5xT[2][1] = -(Ls*s1RD*s2NR-Lt*s123RD*s1RD+Lt*s23NR*s1RD-s1RD*Ls*s12RD
        -Ls*c12RD*c1RD-Lt*c123RD*c1RD)*Lf*jx;
    Jp5xT[2][2] =  Lf*c1RD*p1*jx;
    Jp5xT[2][3] = -c1RD*Lf*jx;
    Jp5xT[2][4] = -(Ls*s2NR-Lt*s123RD+Lt*s23NR-Ls*s12RD)*jx;
    Jp5xT[3][0] =  (Ls*s12RD+Lf*c1RD)*p2*jx;

    Jp5xT[3][1] = -(-s2NR*Ls*Ls*c12RD+s2NR*Ls*s1RD*Lf-Lt*c123RD*Lf*c1RD+Lt*s23NR*s1RD*Lf-Lt*s23NR*Ls*c12RD
        -Lt*c123RD*Ls*s12RD-Lt*s123RD*Lf*s1RD+Lt*s123RD*Ls*c12RD)*jx;

    Jp5xT[3][2] =  (Ls*s12RD+Lf*c1RD)*p1*jx;
    Jp5xT[3][3] = -(Ls*s12RD+Lf*c1RD)*jx;
    Jp5xT[3][4] = -(Ls*s2NR-Lt*s123RD+Lt*s23NR)*jx;
    Jp5xT[4][0] =  (Lt*s123RD+Ls*s12RD+Lf*c1RD)*p2*jx;
    Jp5xT[4][1] = -(Lf*s1RD-Ls*c12RD-Lt*c123RD)*p1*jx;
    Jp5xT[4][2] =  (Lt*s123RD+Ls*s12RD+Lf*c1RD)*p1*jx;
    Jp5xT[4][3] = -(Lt*s123RD+Ls*s12RD+Lf*c1RD)*jx;
    Jp5xT[4][4] = -p1*jx;
}


/* Function: GetJp5xxT
* ------------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp5T for a system with a zero ankle torque at the redundant leg.
* Tp5 = Jp5xxT Fp5 is the joint torque vector and F is the operational force system vector
* Fp5 = [FNRx@hip FRDx@hip Fy@hip Tz@hip Tz@foot]  (see J5xxT in TestForceDistr.m)
*/
void GetJp5xxT(double            Jp5xxT[][5],
               const BodyDataT   *bodyData,
               const int         heelContact,
               const double        *trigRD,
               const double        *trigNR){

    double c2NR, s2NR, c23NR, s23NR, c1RD, s1RD, c12RD, s12RD, c123RD, s123RD, Lf, Ls, Lt, jx;

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    if (heelContact == 1){
        Lf = bodyData->heel.length;
    }else{
        Lf = bodyData->foot.length;
    }

    c2NR   = trigNR[C2];        s2NR   = trigNR[S2]; // get pre-computed trigonometric functions
    c23NR  = trigNR[C23];       s23NR  = trigNR[S23];
    c1RD   = trigRD[C1RD];      s1RD   = trigRD[S1RD];
    c12RD  = trigRD[C12RD];     s12RD  = trigRD[S12RD];
    c123RD = trigRD[C123RD];    s123RD = trigRD[S123RD];

    jx = 1/ (Lt*s123RD+Lf*c1RD+Ls*s12RD); // compute recurring terms

    Jp5xxT[0][0] = -Ls*c2NR-Lt*c23NR;
    Jp5xxT[0][1] = -(Ls*c12RD+Lt*c123RD-Lf*s1RD)*(Ls*s2NR+Lt*s23NR)*jx;
    Jp5xxT[0][2] = -Ls*s2NR-Lt*s23NR;
    Jp5xxT[0][3] =  1;
    Jp5xxT[0][4] =  (Ls*s2NR+Lt*s23NR)*jx;
```

```
    Jp5xxT[1][0] = -Lt*c23NR;
    Jp5xxT[1][1] =  (-Lt*(Ls*c12RD+Lt*c123RD-Lf*s1RD)*s23NR)*jx;
    Jp5xxT[1][2] = -Lt*s23NR;
    Jp5xxT[1][3] =  1;
    Jp5xxT[1][4] =  Lt*s23NR*jx;
    Jp5xxT[2][0] =  0;
    Jp5xxT[2][1] =  0;
    Jp5xxT[2][2] =  0;
    Jp5xxT[2][3] =  1;
    Jp5xxT[2][4] =  0;
    Jp5xxT[3][0] =  0;
    Jp5xxT[3][1] =  (-Lf*(s1RD*Ls*s12RD+Lt*s123RD*s1RD+Ls*c12RD*c1RD+Lt*c123RD*c1RD))*jx;
    Jp5xxT[3][2] =  0;
    Jp5xxT[3][3] =  0;
    Jp5xxT[3][4] =  (-(Ls*s12RD+Lt*s123RD))*jx;
    Jp5xxT[4][0] =  0;
    Jp5xxT[4][1] =  ((-s123RD*Lf*s1RD+s123RD*Ls*c12RD-c123RD*Lf*c1RD-c123RD*Ls*s12RD)*Lt)*jx;
    Jp5xxT[4][2] =  0;
    Jp5xxT[4][3] =  0;
    Jp5xxT[4][4] = -s123RD*Lt*jx;
}

/* Function: GetTRDfootOperational
* ------------------------------------------------------------------------------------------
* Calculates the foot operational torque for a 4dof leg. TRDfoot is the result and
* THMRD is the leg's joint torque vector.
*/
double GetTRDfootOperational(double            *THMRD,
                             const BodyDataT *bodyData,
                             const int        heelContact,
                             const double    *trig){

    double s3, c2, c23, Lf, Ls, Lt, TRDfoot;

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    if (heelContact == 1){
        Lf = bodyData->heel.length;
    }else{
        Lf = bodyData->foot.length;
    }

    s3  = trig[S3RD];
    c2  = trig[C2RD];
    c23 = trig[C23RD];
    // Equation is obtained from last section of SingleRedModel2dof.m
    TRDfoot = THMRD[0] +  THMRD[1]*(Lf*c23-Ls*s3)/ (Ls*s3)
        + THMRD[2]*(-Lf*Ls*c2-Lf*Lt*c23)/ (Lt*Ls*s3)
        + THMRD[3]*Lf*c2/(Lt*s3);

    return TRDfoot;
}


/* Function: GetJp4xT
* ------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp4T for a system with two zero ankle torques.
* Tp4 = Jp4xT Fp4 is the joint torque vector and F is the operational force system vector
* Fp4 = [Fx@hip Fy@hip Tz@hip Tz@foot]
* see J4xT in TestForceDistr.m
*/
void GetJp4xT(double            Jp4xT[][4],
              const double       *RDangles,
              const double       *NRangles,
              const BodyDataT    *bodyData,
              const int           heelContact,
              const double       *trigRD,
              const double       *trigNR){

    double q2NR, q3NR, q1RD, q2RD, q3RD, c2NR, c23NR, s23NR, c1RD, s1RD, c12RD, s12RD, c123RD, s123RD,
        c2RD,  c23RD,  s212, c123, s2123, s1223, s23123, c21, Lf, Ls, Lt, jdiv,  Lf_div, p1, p2, p3, p4;

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    q2NR = NRangles[0];  q3NR = NRangles[1];
    q3RD = RDangles[2];

    Lf = bodyData->foot.length;
    q1RD = RDangles[0] - ANKLE_TOE_HEEL_ANGLE;
    q2RD = RDangles[1] + ANKLE_TOE_HEEL_ANGLE;

    if (heelContact == 1){
        Lf = bodyData->heel.length;
        q1RD = q1RD - HEEL_ANKLE_TOE_ANGLE;
        q2RD = q2RD + HEEL_ANKLE_TOE_ANGLE;
    }

    c2NR   = trigNR[C2];
    c23NR  = trigNR[C23];        s23NR  = trigNR[S23]; // get pre-computed trigonometric functions
    c1RD   = trigRD[C1RD];       s1RD   = trigRD[S1RD];
    c12RD  = trigRD[C12RD];      s12RD  = trigRD[S12RD];
    c123RD = trigRD[C123RD];    s123RD = trigRD[S123RD];
    c2RD   = trigRD[C2RD];
    c23RD  = trigRD[C23RD];

    s212   = sin(q2NR-q1RD-q2RD);
    c123   = cos(-q1RD+q2NR+q3NR);
```

```
    s2123  = sin(q2NR-q1RD-q2RD-q3RD);
    s1223  = sin(-q1RD-q2RD+q2NR+q3NR);
    s23123 = sin(q2NR+q3NR-q1RD-q2RD-q3RD);
    c21    = cos(q2NR-q1RD);

    Lf_div = 1/Lf; // compute recurring terms

    jdiv = 1/(Lt*c123+Ls*c21-Lt*c23RD-Ls*c2RD); // compute recurring terms

    p1 = -(s1RD*Ls*s12RD+Lt*s123RD*s1RD+Ls*c12RD*c1RD+Lt*c123RD*c1RD)*(Ls*c2NR+Lt*c23NR)*jdiv; // compute recurring terms
    p2 =  (s1RD*s12RD+c12RD*c1RD)*Ls*(Ls*c2NR+Lt*c23NR)*jdiv;
    p3 =  Ls*sin(q2NR)+Lt*s23NR;
    p4 =  (Ls*c2RD+Lt*c23RD)*jdiv;

    Jp4xT[0][0] =  (-Ls*cos(-q2RD+q2NR)-Ls*cos(q2RD+q2NR)-Lt*cos(q2NR+q2RD+q3RD)-Lt*cos(q2NR-q2RD-q3RD)+Lt*cos(-q1RD+q3NR)-
Lt*cos(q1RD+q3NR))*0.5*Ls*jdiv;
    Jp4xT[0][1] =  (-Ls*sin(q2RD+q2NR)-Ls*sin(-q2RD+q2NR)-Lt*sin(q2NR-q2RD-q3RD)-Lt*sin(q2NR+q2RD+q3RD)-Lt*sin(q1RD+q3NR)-
Lt*sin(-q1RD+q3NR))*0.5*Ls*jdiv;
    Jp4xT[0][2] =  cos(q2NR-q1RD)*Ls*jdiv;
    Jp4xT[0][3] =  (Ls*s212+Lt*s2123+Lt*sin(q3NR))*Ls*jdiv*Lf_div;
    Jp4xT[1][0] =  p1;
    Jp4xT[1][1] = -p3*p4;
    Jp4xT[1][2] =  (Ls*c21+Lt*c123)*jdiv;
    Jp4xT[1][3] =  (Ls*Ls*s212+Ls*Lt*s2123+Ls*Lt*s1223+Lt*Lt*s23123)*jdiv*Lf_div;
    Jp4xT[2][0] =  p2;
    Jp4xT[2][1] =  p3*Ls*c2RD*jdiv;
    Jp4xT[2][2] = -(s1RD*s12RD+c12RD*c1RD)*Ls*jdiv;
    Jp4xT[2][3] =  (c2NR*Ls*s12RD-Lt*c12RD*s23NR+c12RD*Lt*s123RD-sin(q2NR)*Ls*c12RD+Lt*c23NR*s12RD-
s12RD*Lt*c123RD)*Ls*jdiv*Lf_div;
    Jp4xT[3][0] = -p1;
    Jp4xT[3][1] =  p3*p4;
    Jp4xT[3][2] =  (-Ls*c2RD-Lt*c23RD)*jdiv;
    Jp4xT[3][3] =  (-Ls*Ls*s212-Ls*Lt*s2123-Ls*Lt*s1223-Lt*Lt*s23123)*jdiv*Lf_div;
}

/* Function: ComputeTrig_1Red
* ---------------------------------------------------------------------------------------------
* Computes trigonometric sin and cos functions for the 1Redundancy Double Support state and stores them in an
* array.
* this function reduce the number of cos and sin to be computed in the 1Red double stance state by half.
* RDangles = [toe ankle knee hip]
* NRangles = [ankle knee hip]
*/
void ComputeTrig_1Red(double        *trigRD,
                      double        *trigNR,
                      const double   *RDangles,
                      const double   *NRangles,
                      const int       heelContact){

    double q2NR, q3NR, q4NR, q1RD, q2RD, q3RD, q4RD;

    q2NR = NRangles[0]; // non-redundant leg angles
    q3NR = NRangles[1];
    q4NR = NRangles[2];

    // redundant leg angles
    q1RD = RDangles[0] - ANKLE_TOE_HEEL_ANGLE;
    q2RD = RDangles[1] + ANKLE_TOE_HEEL_ANGLE;

    if (heelContact == 1){
        q1RD = q1RD - HEEL_ANKLE_TOE_ANGLE;
        q2RD = q2RD + HEEL_ANKLE_TOE_ANGLE;
    }

    q3RD = RDangles[2];
    q4RD = RDangles[3];

    // non-redundant leg
    trigNR[C2]   = cos(q2NR);
    trigNR[S2]   = sin(q2NR);
    trigNR[C23]  = cos(q2NR+q3NR);
    trigNR[S23]  = sin(q2NR+q3NR);
    trigNR[C234] = cos(q2NR+q3NR+q4NR);
    trigNR[S234] = sin(q2NR+q3NR+q4NR);
    trigNR[C3]   = cos(q3NR);
    trigNR[S3]   = sin(q3NR);
    trigNR[C4]   = cos(q4NR);
    trigNR[S4]   = sin(q4NR);


    // redundant leg
    trigRD[C1RD]    = cos(q1RD);
    trigRD[S1RD]    = sin(q1RD);
    trigRD[C12RD]   = cos(q1RD+q2RD);
    trigRD[S12RD]   = sin(q1RD+q2RD);
    trigRD[C123RD]  = cos(q1RD+q2RD+q3RD);
    trigRD[S123RD]  = sin(q1RD+q2RD+q3RD);
    trigRD[C1234RD] = cos(q1RD+q2RD+q3RD+q4RD);
    trigRD[S1234RD] = sin(q1RD+q2RD+q3RD+q4RD);
    trigRD[C2RD]    = cos(q2RD);
    trigRD[S3RD]    = sin(q3RD);
    trigRD[C23RD]   = cos(q2RD+q3RD);
    trigRD[C3RD]    = cos(q3RD);
    trigRD[C4RD]    = cos(q4RD);
    trigRD[S4RD]    = sin(q4RD);

}

/* Function: GetTfrictionRedtLeg
```

```
* ------------------------------------------------------------------------------------------
* Calculates the joint torques vector to counteract joint friction and stiffness for one leg. Updates Tf.
* The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
*/
void GetTfrictionRedtLeg(double                *Tf,
                         const double        *angles,
                         const char            side,
                         const SensorDataT    *sensorData){

    int heelSwitch  = 0; // heel switch is activated
    int heelContact = 0; // ONLY heel switch is activated
    int ballSwitch  = 0;
    int toeSwitch   = 0;

    //    if (side == 'L'){
    //        heelSwitch  = sensorData->Lfootswitch[HEEL];
    //        heelContact = sensorData->dynamicMode.leftHeelContact;
    //        ballSwitch  = sensorData->Lfootswitch[BALL];
    //        toeSwitch   = sensorData->Lfootswitch[TOE];
    //    }else{
    //        heelSwitch  = sensorData->Rfootswitch[HEEL];
    //        heelContact = sensorData->dynamicMode.rightHeelContact;
    //        ballSwitch  = sensorData->Rfootswitch[BALL];
    //        toeSwitch   = sensorData->Rfootswitch[TOE];
    //    }
    //
    //    Tf[ANKLE_T] = 12;    // for lifting foot up in toe contact
    //
    //    if(heelSwitch) {
    //        Tf[KNEE_T] = 25;
    //    }else{    // no heel
    //        Tf[KNEE_T] = 15;    // for lifting foot up
    //    }
    //    Tf[HIP_T] = 0;
    //
    //    if (ballSwitch || toeSwitch){ // added 08/08/03 by JRS, values detrmined using manual torque debug code found in
Fhm.c at the end of the file
    //        Tf[ANKLE_T] = Tf[ANKLE_T] + 10; // this was added to help lift the foot
    //        Tf[KNEE_T]  = Tf[KNEE_T]  + 20;
    //        Tf[HIP_T]   = Tf[HIP_T]   + 20;
    //
    ////        if (side == 'L'){
    ////            Tf[KNEE_T] = Tf[KNEE_T] + 10; // added 08-11-03, adds additonal manual torque to left knee only,
prevent knee from locking at heel off
    ////            Tf[HIP_T]  = Tf[HIP_T]  - 20; // added 08-11-03, removes manual torque for left hip only
    ////        }
    //
    //    }

    Tf[ANKLE_T] = 0;
    Tf[KNEE_T]  = 0;
    Tf[HIP_T]   = 0;

}


/* Function: GetTfrictionNRedtLeg
* ------------------------------------------------------------------------------------------
* Calculates the joint torques vector to counteract joint friction and stiffness for one leg. Updates Tf.
* The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
*/
void GetTfrictionNRedtLeg(double            *Tf,
                          const double        *angles,
                          const char        side,
                          const int            heelContact,
                          const double        dNR,
                          const double        dRD,
                          const BodyDataT    *bodyData){

    double footDist;

    //    Tf[ANKLE_T] = 0; // for lifting foot up in toe contact
    //
    //    if(heelContact && side == 'L') {
    //        Tf[KNEE_T] = -11;//-15;    // if other leg is in heel contact
    //    }else if(heelContact && side == 'R'){
    //        Tf[KNEE_T] = -11; // if other leg is in heel contact
    //    }else{
    //        Tf[KNEE_T] = -2;
    //    }
    //
    //    Tf[HIP_T] = -2;

    Tf[ANKLE_T] = 0;
    Tf[KNEE_T]  = 0;
    Tf[HIP_T]   = 0;

}
```

Appendix A.18 – DSup.h

```
/* Function: DoubleSupportTHM
 * ----------------------------------
 * Calculates the joint torque vector due to the human during double support and updates THM.
 * Calculates the gravity compensation joint torque vector during double support and updates Tg.
 * The vector is as follows:[TankleL TkneeL ThipL TankleR TkneeR ThipR]
 */
void DoubleSupportTHM(double *angles, double *velocities, double *accelerations,
            double * torques, const BodyDataT *bodyData, const double Kf, const double *torsoForces,
            ForceDistributionT *distrData, SensorDataT *sensorData,  const VirtualGuardT vguard,
            const SysPropertiesT *sysProperties);

/* Function: Get3dofTg
 * ---------------------------------------------
 * Calculates the joint torques due to gravity for a 3dof leg, and updates Tg.
 * The vector is as follows:[Tankle Tknee Thip]
 */
void Get3dofTg(double *Tg3dof, const BodyDataT *bodyData, const double *trig, const char side,
               SensorDataT *sensorData);

/* Function: Get3dofTcc
 * ---------------------------------------------
 * Calculates the joint torques vector due to the coriolis and centrifugal forces, and updates Tcc.
 * The vector is as follows:[Tankle Tknee Thip]
 */
void Get3dofTcc(double *Tcc, const double *velocities, const BodyDataT *bodyData, const double *trig);

/* Function: Get3dofTHM
 * ---------------------------------------------
 * Calculates the joint torques vector due to the human for a 3dof leg, and updates THM.
 * Does not use backpack force sensor.
 * The vector is as follows:[Tankle Tknee Thip]
 */
void Get3dofTHM(double *Tg, double *Tcc, double *Tf, double *THM, double *Tinertial, const double *velocities, const double
*accelerations,
                                const double *torques, const BodyDataT *bodyData, const double *trig);

/* Function: Get3dofTHMfast
 * ---------------------------------------------
 * Calculates the joint torques vector due to the human for a 3dof leg, and updates THM.
 * This function ignores any components of the torque due to angular velocity.
 * Does not use backpack force sensor.
 * The vector is as follows:[Tankle Tknee Thip]
 */
void Get3dofTHMfast(double *Tg, double *THM, const double *accelerations,
                              const double *torques, const BodyDataT *bodyData, const double *trig);

/* Function: GetJinvT33
 * ---------------------------------------------
 * Calculates the inverse transpose jacobian matrix for a 3dof leg and updates J.
 */
void GetJinvT33(double JinvT[][3], const BodyDataT *bodyData, const double *trig);

/* Function: MatVectMult
 * ---------------------------------------------
 * Computes c  = A * b  where A is a mxn matrix  and b is a nx1 vector.
 */
void MatVectMult(double *c, const double *a, const double *b, const int m, const int n);

/* Function: GetJT33
 * ---------------------------------------------
 * Calculates the transpose jacobian matrix for a 3dof leg and updates J.
 */
void GetJT33(double JT[][3], const BodyDataT *bodyData, const double *trig);

/* Function: GetFootDist2D
 * ---------------------------------------------
 * Calculates and returns the transverse plane distance between the machine CG and the ankle 'd' and the
 * sagittal-transverse horizontal distance between the machine CG and the ankle 'rx'.
 * Arguments are: angles = [ankle knee hip]; hipangles = [rotation abduction]
 * redundancy = 1 if the leg is 4dof, 0 otherwise.
 * heelContact = 1 if the heel is in contact for the redundant leg, 0 otherwise
 * flatFoot = 1 if foot is flat on the ground
 * set heelContact = 1 if the heel is on the ground
 * eq obtained from  FootDistance3DOFfixedTorso.m
 */
void GetFootDist2D(const double *hipAngles, const BodyDataT *bodyData, const int redundancy, const char side,
                   const double *trig, double *rx, double *d, const int doubleStance, const int heelContact,
                   const SensorDataT *sensorData);

/* Function: GetJp5T
 * ---------------------------------------------------------------------------------------------
 * Calculates the jacobian transpose matrix Jp5T for a system with a zero ankle torque at the right leg.
 * Tp5 = Jp5T Fp5 is the joint torque vector and F is the operational force system vector
 * F = [FLx@hip FLy@hip FRx@hip FRy@hip Tz@hip] .
 */
void GetJp5T(double J[][5], const BodyDataT *bodyData, const double *trig1, const double *trig2);

/* Function: GetJTsensor
 * ---------------------------------------------------------------
 * Calculates the transpose jacobian matrix of the backpack force sensor for a 3dof leg and updates JT.
 * [T2 T3 T4]' = JT * [Fx Fy Tz]_0
 * Jacobian is obtained from sensorJacobian.m
 */
void GetJTsensor(double JT[][3], const double *angles, const BodyDataT *bodyData);
```

```
/* Function: GetJp4T
 * ------------------------------------------------------------------------------------------
 * Calculates the jacobian transpose matrix Jp4T for a system with two zero ankle torques.
 * Tp4 = Jp4T Fp4 is the joint torque vector and F is the operational force system vector
 * F = [FLx@hip FRx@hip Fy@hip Tz@hip] .
 * Tp4 = [LkneeT LhipT RkneeT RhipT]
 */
void GetJp4T(double J[][4], const double *Langles, const double *Rangles, const BodyDataT *bodyData,
             const double *trigL, const double *trigR);

/* Function: GetHipRotationFactor
 * ------------------------------------------------------------------------------------------
 * Computes the factor by which to multiply the desired horizontal operational force when accounting for
 * hip rotation.
 * FHMx_actual = FHMx_desired * Krot  (Krot >= 1)
 *
 */
void GetHipRotationFactor(double *Krot, const double *angles);

/* Function: ComputeTrig_DSupport
 * ------------------------------------------------------------------------------------------
 * Computes trigonometric sin and cos functions for the Double Support state and stores them in an array
 * array = [c2L s2L c23L s23L c234L s234L c2R s2R c23R s23R c234R s234R]
 *
 */
void ComputeTrig_DSupport(double *trigL, double *trigR, const double *angles);

/* Function: GetTfrictionDStance
 * ------------------------------------------------
 * Calculates the joint torques vector to counteract joint friction and stiffness and updates Tf.
 * The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
 */
void GetTfrictionDStance(double *Tf, const double *angles, const char side, const SensorDataT *sensorData);
```

Appendix A.19 – DSup.c

```
#include <math.h>
#include <stdio.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"
#include "DSup.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];

extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];


/* Function: DoubleSupportTHM
* ------------------------------------------------------------------------------------------
* Calculates the joint torque vector due to the human during double support and updates THM.
* Calculates the gravity compensation joint torque vector during double support and updates Tg.
* The vector is as follows:[TankleL TkneeL ThipL TankleR TkneeR ThipR]
*/
void DoubleSupportTHM(double               *angles,
                      double               *velocities,
                      double               *accelerations,
                      double               * torques,
                      const BodyDataT       *bodyData,
                      const double           Kf,
                      const double           *torsoForces,
                      ForceDistributionT   *distrData,
                      SensorDataT           *sensorData,
                      const VirtualGuardT   vguard,
                      const SysPropertiesT   *sysProperties){

    int    i, sgn_dL;
    double *Langles, *Rangles, *LhipAngles, *RhipAngles; // pointers to joint angle vectors for each leg
    double *Lvel, *Rvel, *Lacc, *Racc; // pointers to joint velocities,acc., vectors for each leg
    double *Ltorques, *Rtorques; // measured input torques for each leg [Tankle Tknee Thip]
    double JLinvT[3][3], JRinvT[3][3], JLT[3][3], JRT[3][3], Jp5T[5][5], Jp4T[4][4]; // jacobians
    double FHMin[3]; //operational force on machine [Fx Fy Tz]@hip
    double FgLin[3], FgRin[3], FHMLin[3], FHMRin[3]; // operational forces distibuted to each leg [Fx Fy Tz]@hip
    double FHM5[5], FHM4[4];    // special case operational force vectors
    double THM5[5], THM4[4];
    double *THMLin, *THMRin; // torques to produce on each leg
    double Lsn, hsn; // x and y torso distance of the F/T sensor from the hip (m)
    double dL = 0; // distance from the torso CG to the left leg pressure point (m)
    double dR = 0; // distance from the torso CG to the right leg pressure point (m)
    double xL = 0; // sagittal plane distance from the torso CG to the left leg pressure point (m)
    double xR = 0; // sagittal plane distance from the torso CG to the right leg pressure point (m)
    double alpha;  // weight distribution factor
    double betaFg, betaFHM; // load distribution factor for Fg and FHM operational forces
    double Krot; // horizontal force hip rotation factor
    double FgL[3], FgR[3], FHML[3], FHMR[3];   // operational force computed on each leg [Fx Fy Tz]@hip
    double TgL[3], TgR[3], THML[3], THMR[3];
    double TinertialL[3], TinertialR[3];
    double *TinertialLin, *TinertialRin;
    double Tg5[5], Tg4[4];
    double THM[6], Tg[6]; //human machine and feedback linearization torques
    double Tinertial[6];
    double Fgin[3];
    double Fg5[5], Fg4[4];
    double *TgLin, *TgRin;
    double TccR[3] = {0,0,0}; // centrifugal and coriolis force vectors
    double TccL[3] = {0,0,0};
    double TfL[3], TfR[3]; // joint torque stiffness and friction vectors
    double TlinL[3], TlinR[3]; // feedback linearization torques
    double trigL[10], trigR[10]; // point to trig array for left or right leg only where
    // trig contains sin and cos functions used throughout this function
    // trig = [C2L S2L C23L S23L C234L S234L]
    double momentArms[3] = {0,0,0}; // actuator moment arms (m)
    double FHMinHip; // for debugging

    ComputeTrig_DSupport(trigL, trigR, angles); // compute trigonometric functions for each combination of angles

    // angles vector is as follows: [Ltoe Lankle Lknee Lhip Rtoe Rankle Rknee Rhip ]

    Langles = &angles[LANKLE]; // format leg  angle, vel, and acc. vectors in the form [ankle knee hip]
    Rangles = &angles[RANKLE];

    Langles[0] = angles[LTOE] + angles[LANKLE]; // include toe angle in ankle because feet are flat on the ground
    Rangles[0] = angles[RTOE] + angles[RANKLE]; //    so the toe angle actually represents the slope of the ground.
```

```
    LhipAngles = &angles[LHIP_ROT]; // [rotation abduction] angles
    RhipAngles = &angles[RHIP_ROT];

    // Compute gravity torques for feedback linearization

    TgLin = Tg; // assign local vars to point to appropriate Tg value  TgLin = [ankle knee hip]
    TgRin = &Tg[RANKLE_T];                                          //      TgRin = [ankle knee hip]

    Lvel = &velocities[LANKLE]; // create new velocity vectors for each leg
    Rvel = &velocities[RANKLE];

    GetTfrictionDStance(TfL, Langles,'L', sensorData); // get joint friction/stiffness torques (left)
    GetTfrictionDStance(TfR, Rangles,'R', sensorData); // ...(right)

    Get3dofTg(TgL, bodyData, trigL, 'L', sensorData); // compute torque due to gravity and store in Tg = [ankle knee hip]
    Get3dofTg(TgR, bodyData, trigR, 'R', sensorData); // for left (TgL) and right (TgR) legs

    for(i=0; i<3; i++){
        TlinL[i] = TgL[i] + TfL[i]; // compute linearizing torque by adding gravity and friction torques
        TlinR[i] = TgR[i] + TfR[i];
    }

    GetJinvT33(JLinvT, bodyData, trigL); // compute the Jacobian inverse transpose for each leg
    GetJinvT33(JRinvT, bodyData, trigR); // Fg = JLinvT Tg is the operational force due to Tg at the hip joint

    MatVectMult(FgL, &JLinvT[0][0], TlinL,3,3); // compute the operational force due to gravity on each leg
    MatVectMult(FgR, &JRinvT[0][0], TlinR,3,3);

    // **NOTE**: Fg, despite its name, incorporates all linearizing elements (Tcc,Tg and Tf). In the remainder of this
function
    // all variables with the 'g' suffix should be assumed to incorporate not only gravity, but velocity and friction
terms.

    // Compute force distribution factors

    GetFootDist2D(LhipAngles, bodyData, 0, 'L', trigL, &xL, &dL, 1 ,0, sensorData); // get transverse plane  distance
    GetFootDist2D(RhipAngles, bodyData, 0, 'R', trigR, &xR, &dR, 1, 0, sensorData); // from the ankle to the machine CG

    sgn_dL = (int) (dL / fabs(dL));    // compute the sign of dL

    alpha = fabs(dR)/(fabs(dL)+fabs(dR)); // compute the weight distribution factor

    for (i=0; i<3; i++){
        Fgin[i] = FgL[i] + FgR[i]; // compute the total operational force due to gravity
    }

//    betaFg = alpha - sgn_dL * fabs(dL - dR) * Kf * Fgin[0]; // Distribution factor (method 1)
//    //betaFg = alpha - sgn_dL * Fgin[0]/(fabs(Fgin[0])+ Kf*fabs(Fgin[1])) ; // (method 2)
//    //betaFg = alpha * (1  - sgn_dL * Kf * Fgin[0]); // (method 3)
//    //betaFg = 0.5 + (alpha-0.5)/fabs(alpha-0.5) * pow(fabs(0.5-alpha),/*Kf*/1); // (method 4)
//
////    if (betaFg < 0){ // saturate beta (0 means left leg gets all the load, 1 means right leg gets all the load)
////        betaFg = 0;  // values should not lie outside that range
////    }else if(betaFg > 1){
////        betaFg = 1;
////    }
//
//    // filter beta to ensure smooth transition between states
////LowpassFilter(distrData->unfilteredBetaFg, distrData->filteredBetaFg, &betaFg, filterCoeffs2nd05); // 1Hz
//
//    if (betaFg < 0){ // saturate beta
//        betaFg = 0;
//    }else if(betaFg > 1){
//        betaFg = 1;
//    }
//
    // ********************for DEBUGGING 2004-09-08*******
    betaFg = 0.5;
    //*****************************************************

    for (i=0; i<3; i++){
        FgLin[i] = betaFg * Fgin[i]; // Distribute the operational force
        FgRin[i] = Fgin[i] - FgLin[i];
    }

    //----------NEW METHOD ---------------------------------
    // Distribute Forces 07 06 03
    //FgLin[0] = betaFg * Fgin[0];
    //FgLin[1] = alpha  * Fgin[1]; // use CG location only in vertical y direction
    //FgLin[2] = betaFg * Fgin[2];
    //
    //for (i=0; i<3; i++){
    //    FgRin[i] = Fgin[i] - FgLin[i];
    //}
    //-------------------------------------------------------

    GetJT33(JLT, bodyData, trigL); // Compute the Jacobian transpose for each leg
    GetJT33(JRT, bodyData, trigR); // Tgin = JT Fgin   is the joint torque equivalent to Fgin

    MatVectMult(TgLin, &JLT[0][0], FgLin, 3, 3); // Compute the required joint torques on each legs
    MatVectMult(TgRin, &JRT[0][0], FgRin, 3, 3);

    // Compute Human-machine forces

    THMLin = THM; // assign local vars to point to appropriate THM value
    THMRin = &THM[RANKLE_T];

    TinertialLin = Tinertial;
```

```
    TinertialRin = &Tinertial[RANKLE_T];

    // Compute operational forces @ hip
    Lacc = &accelerations[LANKLE]; // create acceleration vectors for each leg
    Racc = &accelerations[RANKLE];

    Ltorques = torques;    //create a new torque vector for each leg in the form [ankle knee hip]
    Rtorques = &torques[RANKLE_T];

    Get3dofTcc(TccL, Lvel, bodyData, trigL); // compute torque due to coriolis and centrifugal forces: Tcc = [ankle knee
hip]
    Get3dofTcc(TccR, Rvel, bodyData, trigR); // for left (TccL) and right (TccR) legs

    Get3dofTHM(TgL, TccL, TfL, THML, TinertialL, Lvel, Lacc, Ltorques, bodyData, trigL); // Get the joint torques due to
the human
    Get3dofTHM(TgR, TccR, TfR, THMR, TinertialR, Rvel, Racc, Rtorques, bodyData, trigR);

    MatVectMult(FHML, &JLinvT[0][0], THML,3,3);    // Compute the operational force due to human for each leg
    MatVectMult(FHMR, &JRinvT[0][0], THMR,3,3);

    for (i=0; i<3; i++){
        FHMin[i] = FHML[i] + FHMR[i];  // Compute the total operational force due to the human
    }

    // compute virtual guard force if necessary
//    GetVGuardForces(sensorData, bodyData, vguard, xR, xL, trigL[C234], trigL[S234]);
//    FHMin[0] = FHMin[0] + sensorData->virtualGuardFx; // add horizontal VGuard force
//    FHMin[2] = FHMin[2] + sensorData->virtualGuardT;  // add hip moment caused by VGuard force at torso CG

    GetHipRotationFactor(&Krot, angles); //compute horizontal force hip rotation factor
//    LowpassFilter(distrData->unfilteredKrot, distrData->filteredKrot, &Krot, filterCoeffs2nd2); // filter Krot for smooth
transition

    // !!!!!!!!!!!!FOR DEBUGGING ONLY    !!!!!!!!!!!!!!!!!!
    Krot = 1; // ignore effects of hip rotation
    //-------------------------------------------------------

    // adjust horizontal operational force components on each leg depending on rotation of the hips.
    FHMin[0] = Krot*FHMin[0]; // more hip rotation leads to a smaller horizontal force component
    FHMin[2] = Krot*FHMin[2]; // more hip rotation leads to a smaller torque component (added may 13 03)


    betaFHM = alpha - sgn_dL * fabs(dL - dR) * Kf * FHMin[0];                 // Distribution factor (method 1)
    //betaFHM = alpha - sgn_dL * FHMin[0]/(fabs(FHMin[0])+Kf*fabs(FHMin[1])); // Distribution factor (method 2)
    //betaFHM = alpha * (1 - sgn_dL * Kf * FHMin[0]);                         // Distribution factor (method 3)

    if (betaFHM < 0){ // saturate beta (0 means left leg gets all the load, 1 means right leg gets all the load)
        betaFHM = 0;
    }else if(betaFHM > 1){
        betaFHM = 1;
    }

    // filter beta to ensure smooth transition between states
    //LowpassFilter(distrData->unfilteredBetaFHM, distrData->filteredBetaFHM, &betaFHM, filterCoeffs2nd1);

    if (betaFHM < 0){ // saturate beta
        betaFHM = 0;
    }else if(betaFHM > 1){
        betaFHM = 1;
    }

    // ********************for DEBUGGING 2004-09-08********
    betaFHM = 0.5;
    //*****************************************************


    for (i=0; i<3; i++){
        FHMLin[i] = betaFHM *  FHMin[i]; // Distribute the operational force between the two legs
        FHMRin[i] = FHMin[i] - FHMLin[i];
    }

    MatVectMult(THMLin, &JLT[0][0], FHMLin, 3, 3); // Compute the equivalent joint torques
    MatVectMult(THMRin, &JRT[0][0], FHMRin, 3, 3); // due to the human on each leg

    // update distrData array for GUI
    distrData->LankleDistance    = dL; // transverse plane distance from CG to ankles
    distrData->RankleDistance    = dR;
    distrData->weightDistrFactor = alpha;

    // update torque data for interface debugging
    sensorData->jointData[LANKLE].Tcc = TccL[ANKLE_T];
    sensorData->jointData[LKNEE].Tcc  = TccL[KNEE_T];
    sensorData->jointData[LHIP].Tcc   = TccL[HIP_T];

    sensorData->jointData[RANKLE].Tcc = TccR[ANKLE_T];
    sensorData->jointData[RKNEE].Tcc  = TccR[KNEE_T];
    sensorData->jointData[RHIP].Tcc   = TccR[HIP_T];

    sensorData->jointData[LANKLE].Tg = TgL[ANKLE_T]; // individual Tg on each half body, not the actual
    sensorData->jointData[LKNEE].Tg  = TgL[KNEE_T];     //   torques used in the controller to counteract gravity
    sensorData->jointData[LHIP].Tg   = TgL[HIP_T];

    sensorData->jointData[RANKLE].Tg = TgR[ANKLE_T];
    sensorData->jointData[RKNEE].Tg  = TgR[KNEE_T];
    sensorData->jointData[RHIP].Tg   = TgR[HIP_T];

    sensorData->jointData[LANKLE].Tf = TfL[ANKLE_T]; // individual Tf on each half body, not the actual
    sensorData->jointData[LKNEE].Tf  = TfL[KNEE_T];     // torques used in the controller to counteract friction
    sensorData->jointData[LHIP].Tf   = TfL[HIP_T];
```

```
    sensorData->jointData[RANKLE].Tf = TfR[ANKLE_T];
    sensorData->jointData[RKNEE].Tf  = TfR[KNEE_T];
    sensorData->jointData[RHIP].Tf   = TfR[HIP_T];

    sensorData->jointData[LANKLE].Thm = 0; //THMLin[ANKLE_T]; //0 - Ltorques[0]; //THM[LANKLE_T];
    sensorData->jointData[LKNEE].Thm  = 0; //THMLin[KNEE_T];  //0 - Ltorques[1]; //THM[LKNEE_T];
    sensorData->jointData[LHIP].Thm   = 0; //THMLin[HIP_T];   //0 - Ltorques[2]; //THM[LHIP_T];

    sensorData->jointData[RANKLE].Thm = 0; //THMRin[ANKLE_T]; //0 - Rtorques[0]; //THM[RANKLE_T];
    sensorData->jointData[RKNEE].Thm  = 0; //THMRin[KNEE_T];  //0 - Rtorques[1]; //THM[RKNEE_T];
    sensorData->jointData[RHIP].Thm   = 0; //THMRin[HIP_T];   //0 - Rtorques[2]; //THM[RHIP_T];

    sensorData->jointData[LANKLE].Tlin = TgLin[ANKLE_T]; // Applied TgLin compensation encompasses Tf as well
    sensorData->jointData[LKNEE].Tlin  = TgLin[KNEE_T];
    sensorData->jointData[LHIP].Tlin   = TgLin[HIP_T];

    sensorData->jointData[RANKLE].Tlin = TgRin[ANKLE_T]; // Applied TgLin compensation encompasses Tf as well
    sensorData->jointData[RKNEE].Tlin  = TgRin[KNEE_T];
    sensorData->jointData[RHIP].Tlin   = TgRin[HIP_T];

    sensorData->jointData[LANKLE].Tinertial = TinertialL[ANKLE_T]; // Ankle torque due to inertial forces, JRS, 2004-06-24
    sensorData->jointData[LKNEE].Tinertial  = TinertialL[KNEE_T];  // Knee  torque due to inertial forces, JRS, 2004-06-24
    sensorData->jointData[LHIP].Tinertial   = TinertialL[HIP_T];   // Hip   torque due to inertial forces, JRS, 2004-06-24

    sensorData->jointData[RANKLE].Tinertial = TinertialR[ANKLE_T]; // Ankle torque due to inertial forces, JRS, 2004-06-24
    sensorData->jointData[RKNEE].Tinertial  = TinertialR[KNEE_T];  // Knee  torque due to inertial forces, JRS, 2004-06-24
    sensorData->jointData[RHIP].Tinertial   = TinertialR[HIP_T];   // Hip   torque due to inertial forces, JRS, 2004-06-24

    // set toe torques to zero    (there's no actuator at the toe)
    sensorData->jointData[LTOE].Tg            = 0;
    sensorData->jointData[RTOE].Tg            = 0;
    sensorData->jointData[LTOE].Thm           = 0;
    sensorData->jointData[RTOE].Thm           = 0;
    sensorData->jointData[LTOE].Tlin        = 0;
    sensorData->jointData[RTOE].Tlin        = 0;
    sensorData->jointData[LTOE].Tcc           = 0;
    sensorData->jointData[RTOE].Tcc           = 0;
    sensorData->jointData[LTOE].Tf            = 0;
    sensorData->jointData[RTOE].Tf            = 0;
    sensorData->jointData[LTOE].Tinertial   = 0;
    sensorData->jointData[RTOE].Tinertial   = 0;
}


/* Function: Get3dofTg
* ------------------------------------------------
* Calculates the joint torques due to gravity for a 3dof leg, and updates Tg.
* The vector is as follows:[Tankle Tknee Thip]
*/
void Get3dofTg(double           *Tg3dof,
               const BodyDataT   *bodyData,
               const double       *trig,
               const char        side,
               SensorDataT       *sensorData){

    double ms, Ls, LGs, hGs, mt, Lt, LGt, hGt, mub, LGub, hGub; // segment properties
    double c2,s2,c23,c234, s23,s234;

    ms   = bodyData->shank.mass; // get mass and geometric properties
    Ls   = bodyData->shank.length;
    LGs  = bodyData->shank.Lcg;
    hGs  = bodyData->shank.hcg;
    mt   = bodyData->thigh.mass;
    Lt   = bodyData->thigh.length;
    LGt  = bodyData->thigh.Lcg;
    hGt  = bodyData->thigh.hcg;
    mub  = bodyData->upperBody.mass;
    LGub = bodyData->upperBody.Lcg;
    hGub = bodyData->upperBody.hcg;

    c2   = trig[C2];    s2   = trig[S2]; // get pre-computed trigonometric functions
    c23  = trig[C23];   s23  = trig[S23];
    c234 = trig[C234];  s234 = trig[S234];

    //  torques from OpSpaceModelHeels2dof.m
    // torque to support segment against gravity
    Tg3dof[ANKLE_T] = -(-ms*g*(-LGs*s2+hGs*c2)-mt*g*(-Ls*s2-LGt*s23+hGt*c23)-0.5*mub*g*(-Ls*s2-Lt*s23-
LGub*s234+hGub*c234)); // negative of ankle V2_3dof.txt

    Tg3dof[KNEE_T]  = -(-mt*g*(-LGt*s23+hGt*c23)-0.5*mub*g*(-Lt*s23-LGub*s234+hGub*c234)); // negative of knee V3_3dof.txt

    Tg3dof[HIP_T]   =  0.5*mub*g*(-LGub*s234+hGub*c234); // negative of hip V4_3dof.txt
}


/* Function: Get3dofTcc
* ------------------------------------------------------------------------------------------
* Calculates the joint torques vector due to the coriolis and centrifugal forces, and updates Tcc.
* The vector is as follows:[Tankle Tknee Thip]
*/
void Get3dofTcc(double           *Tcc,
                const double       *velocities,
                const BodyDataT *bodyData,
                const double       *trig){

    double ms, Ls, LGs, hGs, mt, Lt, LGt, hGt, mub, LGub, hGub; // segment properties
```

```
    double dq2,dq3,dq4,  dq23,dq234;
    double c2,s2,c23,c234,s23,s234,  p1,p2,p3;

    ms   = bodyData->shank.mass; // get mass and geometric properties
    Ls   = bodyData->shank.length;
    LGs  = bodyData->shank.Lcg;
    hGs  = bodyData->shank.hcg;
    mt   = bodyData->thigh.mass;
    Lt   = bodyData->thigh.length;
    LGt  = bodyData->thigh.Lcg;
    hGt  = bodyData->thigh.hcg;
    mub  = bodyData->upperBody.mass;
    LGub = bodyData->upperBody.Lcg;
    hGub = bodyData->upperBody.hcg;

    c2   = trig[C2];    s2   = trig[S2]; // get pre-computed trigonometric functions
    c23  = trig[C23];    s23  = trig[S23];
    c234 = trig[C234];    s234 = trig[S234];

    // get velocities
    dq2   = velocities[0];
    dq3   = velocities[1];
    dq4   = velocities[2];
    dq23  = dq2+dq3;
    dq234 = dq2+dq3+dq4;

    // torques from OpSpaceModelHeels2dof.m
    p1 = -Ls*s2-Lt*s23-LGub*s234+hGub*c234; // compute recurring terms
    p2 = -Ls*c2-Lt*c23-LGub*c234-hGub*s234;
    p3 = -LGub*c234*dq234-hGub*s234*dq234;

    Tcc[ANKLE_T] = 0.5*ms*(4*dq2*(LGs*(c2)+hGs*(s2))*(-LGs*(s2)*dq2+hGs*(c2)*dq2)+4*dq2*(-LGs*(s2)+hGs*(c2))*(-
LGs*(c2)*dq2-hGs*(s2)*dq2))-0.5*mt*(2*(-dq2*Ls*(c2)-((dq23))
        *(LGt*(c23)+hGt*(s23)))*(Ls*(s2)*dq2-((dq23))*(-LGt*(s23)+hGt*(c23)))+2*(-Ls*(s2)*dq2+((dq23))*(-
LGt*(s23)+hGt*(c23)))*(-dq2*Ls*(c2)+((dq23))*(-LGt*(c23)
        -hGt*(s23))))+0.5*mt*(2*(-((dq23))*(-LGt*(s23)*((dq23))+hGt*(c23)*((dq23)))+dq2*dq2*Ls*(s2))*(-Ls*(c2)-LGt*(c23)-
hGt*(s23))+2*(-Ls*(s2)*dq2+((dq23))
        *(-LGt*(s23)+hGt*(c23)))*(-dq2*Ls*(c2)-LGt*(c23)*((dq23))-hGt*(s23)*((dq23)))+2*(-dq2*Ls*(c2)-
((dq23))*(LGt*(c23)+hGt*(s23)))*(Ls*(s2)*dq2+LGt*(s23)*((dq23))
        -hGt*(c23)*((dq23)))+2*(((dq23))*(-LGt*(c23)*((dq23))-hGt*(s23)*((dq23)))-dq2*dq2*Ls*(c2))*(-Ls*(s2)-
LGt*(s23)+hGt*(c23)))+0.25*mub*(2*(dq23*dq23
        *Lt*(s23)+dq2*dq2*Ls*(s2)-(dq234)*(-LGub*(s234)*(dq234)+hGub*(c234)*(dq234)))*(p2)
        +2*(-Ls*(s2)*dq2-Lt*(s23)*((dq23))+(dq234)*(-LGub*(s234)+hGub*(c234)))*(-dq2*Ls*(c2)-((dq23))*Lt*(c23)-
LGub*(c234)*(dq234)-hGub*(s234)
        *(dq234))+2*(-dq2*Ls*(c2)-((dq23))*Lt*(c23)-
(dq234)*(LGub*(c234)+hGub*(s234)))*(Ls*(s2)*dq2+Lt*(s23)*((dq23))+LGub*(s234)*(((dq4+dq3
        +dq2)))-hGub*(c234)*(dq234))+2*(-dq23*dq23*Lt*(c23)-dq2*dq2*Ls*(c2)+(dq234)*(-LGub*(c234)*(dq234)-
hGub*(s234)*(((dq4
        +dq3+dq2)))))*(p1))-0.25*mub*(2*(-dq2*Ls*(c2)-((dq23))*Lt*(c23)-(dq234)*(LGub*(c234)+hGub*(s234)))*(Ls*(s2)
        *dq2+Lt*(s23)*((dq23))-(dq234)*(-LGub*(s234)+hGub*(c234)))+2*(-Ls*(s2)*dq2-Lt*(s23)*((dq23))+(dq234)*(-
LGub*(s234)+hGub*(c234)))*(-dq2
        *Ls*(c2)-((dq23))*Lt*(c23)+(dq234)*(-LGub*(c234)-hGub*(s234))))-0.5*ms*(2*dq2*dq2*(LGs*(c2)+hGs*(s2))*(-
LGs*(s2)+hGs*(c2))+2*dq2*dq2*(-LGs
        *(s2)+hGs*(c2))*(-LGs*(c2)-hGs*(s2))); // ankle KE2_3dof.txt

    Tcc[KNEE_T] = 0.5*mt*(2*(-((dq23))*(-LGt*(s23)*((dq23))+hGt*(c23)*((dq23)))+dq2*dq2*Ls*(s2))*(-LGt*(c23)-hGt*(s23))+2*(
        -dq2*Ls*(s2)+((dq23))*(-LGt*(s23)+hGt*(c23))))*(-LGt*(c23)*((dq23))-hGt*(s23)*((dq23)))+2*(-dq2*Ls*(c2)-
((dq23))*(LGt*(c23)+hGt*(s23)))*(LGt*(s23)*((dq23))
        -hGt*(c23)*((dq23)))+2*(((dq23))*(-LGt*(c23)*((dq23))-hGt*(s23)*((dq23)))-dq2*dq2*Ls*(c2))*(-LGt*(s23)+hGt*(c23)))-
0.25*mub*(2*(-dq2*Ls*(c2)-((dq23))
        *Lt*(c23)-(dq234)*(LGub*(c234)+hGub*(s234)))*(Lt*(s23)*((dq23))-(dq234)*(-LGub*(s234)+hGub*(c234)))+2*(-
dq2*Ls*(s2)-Lt*(s23)*((dq23))
        +(dq234)*(-LGub*(s234)+hGub*(c234)))*(-((dq23))*Lt*(c23)+(dq234)*(-LGub*(c234)-
hGub*(s234))))+0.25*mub*(2*(dq23*dq23*Lt*(s23)+dq2*dq2
        *Ls*(s2)-(dq234)*(-LGub*(s234)*(dq234)+hGub*(c234)*(dq234)))*(-Lt*(c23)-LGub*(c234)-hGub*(s234))+2*(-dq2*Ls*(s2)-
Lt*(s23)
        *((dq23))+(dq234)*(-LGub*(s234)+hGub*(c234)))*(-((dq23))*Lt*(c23)+p3)+2*(-dq2*Ls*(c2)
        -((dq23))*Lt*(c23)-(dq234)*(LGub*(c234)+hGub*(s234)))*(Lt*(s23)*((dq23))+LGub*(s234)*(dq234)-
hGub*(c234)*(dq234))+2*(-dq23*dq23
        *Lt*(c23)-dq2*dq2*Ls*(c2)+(dq234)*(p3))*(-Lt*(s23)-LGub*(s234)+hGub*(c234)))-0.5
        *mt*(-2*(-dq2*Ls*(c2)-((dq23))*(LGt*(c23)+hGt*(s23)))*((dq23))*(-LGt*(s23)+hGt*(c23))+2*(-dq2*Ls*(s2)+((dq23))*(-
LGt*(s23)+hGt*(c23)))*((dq23))*(-LGt*(c23)-hGt*(s23))); // knee KE3_3dof.txt

    Tcc[HIP_T] = 0.25*mub*(2*(dq23*dq23*Lt*(s23)
        +dq2*dq2*Ls*(s2)-(dq234)*(-LGub*(s234)*(dq234)+hGub*(c234)*(dq234)))*(-LGub*(c234)-hGub*(s234))+2*(-dq2*Ls*(s2)-
((dq23))
        *Lt*(s23)+(dq234)*(-LGub*(s234)+hGub*(c234)))*(p3)+2*(-dq2*Ls*(c2)-((dq23))*Lt*(c23)
        -(dq234)*(LGub*(c234)+hGub*(s234)))*(LGub*(s234)*(dq234)-hGub*(c234)*(dq234))+2*(-dq23*dq23*Lt*(c23)-
dq2*dq2*Ls*(c2)
        +(dq234)*(p3))*(-LGub*(s234)+hGub*(c234)))-0.25*mub*(-2*(-dq2*Ls*(c2)-((dq23))*Lt*(c23)
        -(dq234)*(LGub*(c234)+hGub*(s234)))*(dq234)*(-LGub*(s234)+hGub*(c234))+2*(-dq2*Ls*(s2)-((dq23))*Lt*(s23)+(dq234)*(-
LGub*(s234)
        +hGub*(c234)))*(dq234)*(-LGub*(c234)-hGub*(s234))); // hip KE4_3dof.txt
}


/* Function: Get3dofTHM
* ------------------------------------------------------------------------------------------
* Calculates the joint torques vector due to the human for a 3dof leg, and updates THM.
* Does not use backpack force sensor. NOTE: This Thm includes Tcc!!
* The vector is as follows:[Tankle Tknee Thip]
*/
void Get3dofTHM(double           *Tg,
                double           *Tcc,
                double           *Tf,
                double           *THM,
                double           *Tinertial,
                const double       *velocities,
                const double       *accelerations,
```

```
                const double        *torques,
                const BodyDataT *bodyData,
                const double        *trig){

    double ms, Is, Ls, LGs, hGs, mt, It, Lt, LGt, hGt, mub, Iub, LGub, hGub; // segment properties
    double ddq2,ddq3,ddq4;
    double c2,s2,c23,c234,s23,s234, p1,p2;

    ms   = bodyData->shank.mass; // get mass and geometric properties
    Is   = bodyData->shank.inertia;
    Ls   = bodyData->shank.length;
    LGs  = bodyData->shank.Lcg;
    hGs  = bodyData->shank.hcg;
    mt   = bodyData->thigh.mass;
    It   = bodyData->thigh.inertia;
    Lt   = bodyData->thigh.length;
    LGt  = bodyData->thigh.Lcg;
    hGt  = bodyData->thigh.hcg;
    mub  = bodyData->upperBody.mass;
    Iub  = bodyData->upperBody.inertia;
    LGub = bodyData->upperBody.Lcg;
    hGub = bodyData->upperBody.hcg;

    c2   = trig[C2];    s2   = trig[S2]; // get pre-computed trigonometric functions
    c23  = trig[C23];   s23  = trig[S23];
    c234 = trig[C234];  s234 = trig[S234];

    ddq2 = accelerations[0];
    ddq3 = accelerations[1];
    ddq4 = accelerations[2];

    // torques from OpSpaceModelHeels2dof.m
    p1 = -Ls*s2-Lt*s23-LGub*s234+hGub*c234; // compute recurring terms
    p2 = -Ls*c2-Lt*c23-LGub*c234-hGub*s234;

    Tinertial[ANKLE_T] = (0.25*mub*(2*(-LGub*(c234)-hGub*(s234))*(p2)+2*(-LGub*(s234)+hGub*(c234))*(p1))
        +0.5*Iub)*ddq4+(0.5*mt*(2*(-LGt*(c23)-hGt*(s23))*(-Ls*(c2)-LGt*(c23)-hGt*(s23))+2*(-LGt*(s23)+hGt*(c23))*(-Ls*(s2)-
LGt*(s23)+hGt*(c23)))+0.25*mub*(2*(-Lt
        *(c23)-LGub*(c234)-hGub*(s234))*(p2)+2*(-Lt*(s23)-LGub*(s234)+hGub*(c234))*(p1))
        +It+0.5*Iub)*ddq3+(Is+It+0.5*mt*(2*(-Ls*(s2)-LGt*(s23)+hGt*(c23))*(-Ls*(s2)-LGt*(s23)+hGt*(c23))+2*(-Ls*(c2)-
LGt*(c23)-hGt*(s23))*(-Ls*(c2)-LGt*(c23)-hGt*(s23)))+0.5*Iub+0.25*mub
        *(2*p1*p1+2*p2*p2)+0.5*ms*(2*(LGs*(c2)+hGs*(s2))*(LGs*(c2)+hGs*(s2))+2*(-LGs*(s2)+hGs*(c2))*(-LGs*(s2)+hGs*(c2))))
        *ddq2; // + Tcc[0] + Tg[0] + Tf[0] - torques[0]; // ankle KE2_3dof.txt

    Tinertial[KNEE_T] = (0.25*mub*(2*(-LGub*(c234)-hGub*(s234))*(-Lt*(c23)-LGub*(c234)-hGub*(s234))+2*(-
LGub*(s234)+hGub*(c234))*(-Lt*(s23)-LGub*(s234)+hGub*(c234)))+0.5*Iub)*ddq4
        +(0.5*mt*(2*(-LGt*(c23)-hGt*(s23))*(-LGt*(c23)-hGt*(s23))+2*(-LGt*(s23)+hGt*(c23))*(-
LGt*(s23)+hGt*(c23)))+It+0.25*mub*(2*(-Lt*(c23)-LGub*(c234)-hGub*(s234))*(-Lt*(c23)-LGub*(c234)-hGub*(s234))+2
        *(-Lt*(s23)-LGub*(s234)+hGub*(c234))*(-Lt*(s23)-LGub*(s234)+hGub*(c234)))+0.5*Iub)*ddq3+(It+0.5*mt*(2*(-Ls*(s2)-
LGt*(s23)+hGt*(c23))*(-LGt*(s23)+hGt*(c23))+2*(-LGt*(c23)-hGt*(s23)-Ls*(c2))*(-LGt*(c23)
        -hGt*(s23)))+0.5*Iub+0.25*mub*(2*(p1)*(-Lt*(s23)-LGub*(s234)+hGub*(c234))+2*(-Ls*(c2)-LGub*(c234)-hGub*(s234)-
Lt*(c23))
        *(-Lt*(c23)-LGub*(c234)-hGub*(s234))))*ddq2; // + Tcc[1] + Tg[1] + Tf[1] - torques[1]; // knee KE3_3dof.txt

    Tinertial[HIP_T] = (0.25*mub*(2*(-LGub*(c234)-hGub*(s234))*(-LGub*(c234)-hGub*(s234)) +2*(-LGub*(s234)+hGub*(c234))*(-
LGub*(s234)+hGub*(c234)))+0.5*Iub)*ddq4+(0.25*mub*(2*(-Lt*(c23)-LGub*(c234)-hGub*(s234))*(-LGub
        *(c234)-hGub*(s234))+2*(-Lt*(s23)-LGub*(s234)+hGub*(c234))*(-
LGub*(s234)+hGub*(c234)))+0.5*Iub)*ddq3+(0.25*mub*(2*(-Ls*(s2)-LGub*(s234)+hGub*(c234)-Lt*(s23))
        *(-LGub*(s234)+hGub*(c234))+2*(-Ls*(c2)-LGub*(c234)-hGub*(s234)-Lt*(c23))*(-LGub*(c234)-
hGub*(s234)))+0.5*Iub)*ddq2; // + Tcc[2] + Tg[2] + Tf[2] - torques[2]; // hip KE4_3dof.txt

    THM[ANKLE_T] = Tinertial[ANKLE_T] + Tcc[ANKLE_T] + Tg[0] + Tf[0] - torques[0];
    THM[KNEE_T]  = Tinertial[KNEE_T]  + Tcc[KNEE_T]  + Tg[1] + Tf[1] - torques[1];
    THM[HIP_T]   = Tinertial[HIP_T]   + Tcc[HIP_T]   + Tg[2] + Tf[2] - torques[2];
}


/* Function: GetJinvT33
* ------------------------------------------------------------------------------------------
* Calculates the inverse transpose jacobian matrix JinT for a 3dof leg. F = JinvT T where T is the
*joint torque vector and F is the operational force at the hip.
* This is JLinvT in OpSpaceModelHeels2dof.m
*/
void GetJinvT33(double            JinvT[][3],
                const BodyDataT *bodyData,
                const double        *trig){

    double s2, c2, s23, c23, Ls, Lt, p, Ls_div, Lt_div;

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    c2  = trig[C2];
    s2  = trig[S2];
    s23 = trig[S23];
    c23 = trig[C23];

    p      = 1/(s23*c2-c23*s2); // = 1/s3
    Ls_div = 1/Ls;
    Lt_div = 1/Lt;

    //[row][col]
    JinvT[0][0] = -s23*Ls_div*p;
    JinvT[0][1] =  (Ls*s2+Lt*s23)*Lt_div*Ls_div*p;
    JinvT[0][2] = -s2*Lt_div*p;
    JinvT[1][0] =  c23*Ls_div*p;
    JinvT[1][1] = -(Ls*c2+Lt*c23)*Lt_div*Ls_div*p;
    JinvT[1][2] =  c2*Lt_div*p;
    JinvT[2][0] =  0;
```

```
    JinvT[2][1] =  0;
    JinvT[2][2] =  1;
}


/* Function: GetJT33
* ------------------------------------------------------------------------------------------
* Calculates the transpose jacobian matrix for a 3dof leg and updates JT.
* The jacobian transforms an opertional force of the form [Fx_hip Fy_hip Tz_hip] into a torque vector [Tankl Tknee Thip]
* T = JT33 F
*/
void GetJT33(double             JT[][3],
             const BodyDataT *bodyData,
             const double     *trig){

    double Ls, Lt, c23, s23;

    Ls  = bodyData->shank.length;
    Lt  = bodyData->thigh.length;

    c23 = trig[C23];
    s23 = trig[S23];

    //[row][col]
    JT[0][0] = -Ls*trig[C2]-Lt*c23;
    JT[0][1] = -Ls*trig[S2]-Lt*s23;
    JT[0][2] =     1;
    JT[1][0] = -Lt*c23;
    JT[1][1] = -Lt*s23;
    JT[1][2] =  1;
    JT[2][0] =  0;
    JT[2][1] =  0;
    JT[2][2] =  1;
}


/* Function: MatVectMult
* ------------------------------------------------
* Computes c  = A * b  where A is a mxn matrix  and b is a nx1 vector.
*/
void MatVectMult(double         *c,
                 const double *a,
                 const double *b,
                 const int     m,
                 const int     n){

    double sum = 0.0;
    int i,j;

    for(i=0; i<m; i++){                 // m = number of rows in matrix
        for(j=0 ;j<n; j++)             // n = number of columns in matrix = nber of rows in vector
            sum = sum + A(i,j) * b[j]; // A(row, col)
        c[i] = sum;
        sum = 0.0;
    }
}


/* Function: GetFootDist2D
* ------------------------------------------------
* Calculates and returns the transverse plane distance between the machine CG and the ankle 'd' and the
* sagittal-transverse horizontal distance between the machine CG and the ankle 'rx'.
* Arguments are: angles = [ankle knee hip]; hipangles = [rotation abduction]
* redundancy = 1 if the leg is 4dof, 0 otherwise.
* heelContact = 1 if the heel is in contact for the redundant leg, 0 otherwise
* flatFoot = 1 if foot is flat on the ground
* set heelContact = 1 if the heel is on the ground
* eq obtained from  FootDistance3DOFfixedTorso.m
*/
void GetFootDist2D(const double         *hipAngles,
                   const BodyDataT     *bodyData,
                   const int          redundancy,
                   const char          side,
                   const double         *trig,
                   double              *rx,
                   double              *d,
                   const int          flatFoot,
                   const int          heelContact,
                   const SensorDataT *sensorData){

    double q5, q6, Ls, Lt, LGub, hGub, s1234, c1234,c3,s3,c4,s4, c6, s6, s5, c5, s12, s123, rz;
    double rx_local;
    double pPointDist = 0; // number of active footswitches

    Ls   = bodyData->shank.length;
    Lt   = bodyData->thigh.length;
    LGub = bodyData->upperBody.Lcg;
    hGub = bodyData->upperBody.hcg;

    // assume the leg is 3dof
    q5 = hipAngles[0];
    q6 = hipAngles[1]; // unactuated hip joint angles

    // compute non-sagittal plane trigonometric functions
    s5 = sin(q5);    c5 = cos(q5);
    c6 = cos(q6);    s6 = sin(q6);

    // get pre-computed sagittal plane trigonometric functions
    if (redundancy == 0){ // 3 dof leg
```

```
        c1234 = trig[C234];         s1234 = trig[S234]; // use appropriate index for 'trig'
        s12   = trig[S2];          s123  = trig[S23];
        c3    = trig[C3];          s3    = trig[S3];
        c4    = trig[C4];          s4    = trig[S4];
    }else{                  // 4dof leg
        c1234 = trig[C1234RD];     s1234 = trig[S1234RD];
        s12   = trig[S12RD];     s123  = trig[S123RD];
        c3    = trig[C3RD];        s3    = trig[S3RD];
        c4    = trig[C4RD];        s4    = trig[S4RD];
    }

    // Note: torso is assumed to lie in sagittal plane (not tilted sideways)
    if (side == 'R'){ // z- axis distance computation depends on leg being considered
        rx_local = Ls*(((c1234*c5-s1234*s6*s5)*c4+s1234*c6*s4)*s3+((c1234*c5-s1234*s6*s5)*s4-s1234*c6*c4)*c3)
                   +Lt*((c1234*c5-s1234*s6*s5)*s4-s1234*c6*c4)+LABD*s1234*s6+hGub*c1234-LGub*s1234;
        rz       = Ls*((c6*s5*c4+s6*s4)*s3+(c6*s5*s4-s6*c4)*c3)+Lt*(c6*s5*s4-s6*c4)-LABD*c6-DGUB;
    }else{
        rx_local = Ls*(((c1234*c5-s1234*s6*s5)*c4+s1234*c6*s4)*s3+((c1234*c5-s1234*s6*s5)*s4-s1234*c6*c4)*c3)
                   +Lt*((c1234*c5-s1234*s6*s5)*s4-s1234*c6*c4)-LABD*s1234*s6+hGub*c1234-LGub*s1234;
        rz       = Ls*((c6*s5*c4+s6*s4)*s3+(c6*s5*s4-s6*c4)*c3)+Lt*(c6*s5*s4-s6*c4)+LABD*c6+DGUB;
    }

    rx_local = - rx_local; // switch sign

    // add average distance to pressure point of foot to rx_ankle
    // use footswitches to determine pressure point. A more accurate estimation of the pressure point position
    // makes the system more stable
    if(side == 'L'){
        pPointDist = ( sensorData->Lfootswitch[HEEL] * HEEL_DIST/2
                     + sensorData->Lfootswitch[MIDFOOT] * (MIDFOOT_DIST/2 + HEEL_DIST)
                     + sensorData->Lfootswitch[BALL] * (BALL_DIST/2 + MIDFOOT_DIST + HEEL_DIST)
                     + sensorData->Lfootswitch[TOE] * (TOE_DIST/2 + BALL_DIST + MIDFOOT_DIST+ HEEL_DIST) )
                     / ( sensorData->Lfootswitch[HEEL] + sensorData->Lfootswitch[MIDFOOT]
                     + sensorData->Lfootswitch[BALL] + sensorData->Lfootswitch[TOE]);
        rx_local   = rx_local + pPointDist * cos(sensorData->jointData[LTOE].position);
    }else{
        pPointDist = ( sensorData->Rfootswitch[HEEL] * HEEL_DIST/2
                     + sensorData->Rfootswitch[MIDFOOT] * (MIDFOOT_DIST/2 + HEEL_DIST)
                     + sensorData->Rfootswitch[BALL] * (BALL_DIST/2 + MIDFOOT_DIST + BALL_DIST + HEEL_DIST)
                     + sensorData->Rfootswitch[TOE] * (TOE_DIST/2 + MIDFOOT_DIST + BALL_DIST + HEEL_DIST) )
                     / ( sensorData->Rfootswitch[HEEL] + sensorData->Rfootswitch[MIDFOOT]
                     + sensorData->Rfootswitch[BALL] + sensorData->Rfootswitch[TOE]);
        rx_local   = rx_local + pPointDist * cos(sensorData->jointData[RTOE].position);
    }

    *rx = rx_local;
    //*d = sqrt(rx_local*rx_local + rz*rz)*fabs(rx_local)/rx_local; // tranverse plane distance (m), define sign of
distance as sign of rx
    *d = rx_local; // for debugging use only distance in the sagittal plane
}


/* Function: GetJp5T
* ------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp5T for a system with a zero ankle torque.
* Tp5 = Jp5T Fp5 is the joint torque vector and F is the operational force system vector
* F = [FLx@hip FLy@hip FRx@hip FRy@hip Tz@hip]. (see J5xT in TestForceDistr.m)
*/
void GetJp5T(double             J[][5],
             const BodyDataT *bodyData,
             const double     *trig1,
             const double     *trig2){

    double c23L, s23L, c23R,s23R, Ls, Lt, p1, p2;

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    c23R = trig2[C23];    s23R = trig2[S23];
    c23L = trig1[C23];    s23L = trig1[S23];

    p1 = Ls*trig1[C2]+Lt*c23L;
    p2 = Ls*trig1[S2]+Lt*s23L;

    //[row][col]
    J[0][0] =  Ls*trig1[C2];
    J[0][1] =  Ls*trig1[S2];
    J[0][2] =  0;
    J[0][3] =  0;
    J[0][4] =  0;
    J[1][0] =  p1;
    J[1][1] =  p2;
    J[1][2] =  0;
    J[1][3] =  0;
    J[1][4] =  0;
    J[2][0] = -p1;
    J[2][1] = -p2;
    J[2][2] = -Ls*trig2[C2]-Lt*c23R;
    J[2][3] = -Ls*trig2[S2]-Lt*s23R;
    J[2][4] =  1;
    J[3][0] = -p1;
    J[3][1] = -p2;
    J[3][2] = -Lt*c23R;
    J[3][3] = -Lt*s23R;
    J[3][4] =  1;
    J[4][0] = -p1;
    J[4][1] = -p2;
    J[4][2] =  0;
    J[4][3] =  0;
```

```
    J[4][4] =  1;
}


/* Function: GetJp4T
* ------------------------------------------------------------------------------------------
* Calculates the jacobian transpose matrix Jp4T for a system with two zero ankle torques.
* Tp4 = Jp4T Fp4 is the joint torque vector and F is the operational force system vector
* F = [FLx@hip FRx@hip Fy@hip Tz@hip] .
* Tp4 = [LkneeT LhipT RkneeT RhipT]
* (J4xT in TestForceDistr.m)
*/
void GetJp4T(double             J[][4],
             const double     *Langles,
             const double     *Rangles,
             const BodyDataT *bodyData,
             const double     *trigL,
             const double     *trigR){

    double q3L, q3R, c2L, s2L, c2R, s2R, c23L, s23L, c23R,s23R, Ls, Lt, p, p1, p2, p3, p4;

    Ls = bodyData->shank.length;
    Lt = bodyData->thigh.length;

    q3L = Langles[1];
    q3R = Rangles[1];

    s2L  = trigL[S2];    c2L  = trigL[C2];
    c23L = trigL[C23];    s23L = trigL[S23];
    s2R  = trigR[S2];    c2R  = trigR[C2];
    c23R = trigR[C23];    s23R = trigR[S23];

    p1 = (Ls*c2R+Lt*c23R);
    p2 = (Ls*c2L+Lt*c23L);
    p3 = (Ls*s2R+Lt*s23R);
    p4 = (Ls*s2L+Lt*s23L);
    p  = 1/(Lt*s23L+Ls*s2L-Ls*s2R-Lt*s23R);

    //[row][col]
    J[0][0] =  (Lt*sin(q3L)-c2L*Ls*s2R-c2L*Lt*s23R)*Ls*p;
    J[0][1] = -s2L*p1*Ls*p;
    J[0][2] = -s2L*p3*Ls*p;
    J[0][3] =  s2L*Ls*p;
    J[1][0] = -p2*p3*p;
    J[1][1] = -p4*p1*p;
    J[1][2] = -p4*p3*p;
    J[1][3] =  p4*p;
    J[2][0] =  p2*Ls*s2R*p;
    J[2][1] =  (Lt*c2R*s23L+Ls*s2L*c2R-Lt*sin(q3R))*p;
    J[2][2] =  p4*Ls*s2R*p;
    J[2][3] = -s2R*Ls*p;
    J[3][0] =  p2*p3*p;
    J[3][1] =  p4*p1*p;
    J[3][2] =  p4*p3*p;
    J[3][3] = -p3*p;
}


/* Function: ComputeTrig_DSupport
* ------------------------------------------------------------------------------------------
* Computes trigonometric sin and cos functions for the Double Support state and stores them in an array.
* array = [c2L s2L c23L s23L c234L s234L c2R s2R c23R s23R c234R s234R]
* this function reduce the number of cos and sin to be computed in the double stance state by half.
*/
void ComputeTrig_DSupport(double          *trigL,
                          double          *trigR,
                          const double *angles){

    double q2L, q3L, q4L, q2R, q3R, q4R;

    q2L = angles[LANKLE] + angles[LTOE]; // include toe angle because foot is flat on the ground
    q3L = angles[LKNEE];
    q4L = angles[LHIP];

    q2R = angles[RANKLE] + angles[RTOE];
    q3R = angles[RKNEE];
    q4R = angles[RHIP];

    // left leg
    trigL[0]    = cos(q2L);
    trigL[S2]   = sin(q2L);
    trigL[C23]  = cos(q2L+q3L);
    trigL[S23]  = sin(q2L+q3L);
    trigL[C234] = cos(q2L+q3L+q4L);
    trigL[S234] = sin(q2L+q3L+q4L);
    trigL[C3]   = cos(q3L);
    trigL[S3]   = sin(q3L);
    trigL[C4]   = cos(q4L);
    trigL[S4]   = sin(q4L);

    // right leg
    trigR[0]    = cos(q2R);
    trigR[S2]   = sin(q2R);
    trigR[C23]  = cos(q2R+q3R);
    trigR[S23]  = sin(q2R+q3R);
    trigR[C234] = cos(q2R+q3R+q4R);
    trigR[S234] = sin(q2R+q3R+q4R);
    trigR[C3]   = cos(q3R);
    trigR[S3]   = sin(q3R);
```

```
    trigR[C4]   = cos(q4R);
    trigR[S4]   = sin(q4R);
}


/* Function: GetHipRotationFactor
* ------------------------------------------------------------------------------------------
* Computes the factor by which to multiply the desired horizontal operational force when accounting for
* hip rotation.
* FHMx_actual = FHMx_desired * Krot  (Krot >= 1)
*
*/
void GetHipRotationFactor(double          *Krot,
                          const double *angles){

    double qrot;

    qrot = fabs(angles[RHIP_ROT] - angles[LHIP_ROT]);

    if( qrot > QROT1 ){ // if the absolute hip rotation difference is large
        *Krot = qrot*KROT_SLOPE + KROT_OFFSET; // decrease the horizontal force factor    ( Krot = (qrot - qrot1)/(qrot8 -
qrot1) )
        if(*Krot < 0)
            *Krot = 0; // saturate Krot at 0
    }else{ // if hip rotations don't deviate much keep Krot at 1
        *Krot = 1;
    }
}


/* Function: GetTfrictionDStance
* ------------------------------------------------------------------------------------------
* Calculates the joint torques vector to counteract joint friction and stiffness for one leg. Updates Tf.
* The vector is as follows:[Tankle Tknee Thip] and represents the torque of the distal segment on the proximal segment.
* Equations are obtained from Excel documents 'rankle stiffness.xls', 'rknee stiffness.xls', 'rhip stiffness.xls'.
*/
void GetTfrictionDStance(double               *Tf,
                         const double        *angles,
                         const char           side,
                         const SensorDataT    *sensorData){

    int heelContact    = 0;
    int midfootContact = 0;

    if (side == 'L'){
        heelContact    = sensorData->Lfootswitch[HEEL];
        midfootContact = sensorData->Lfootswitch[MIDFOOT];
    }else{ // R side
        heelContact    = sensorData->Rfootswitch[HEEL];
        midfootContact = sensorData->Rfootswitch[MIDFOOT];
    }

    //Commented out for EXO2 on 04/23/2004 by JRS
    //    Tf[ANKLE_T] = 0;  // +8Nm added 00/00/03
    //
    //    if(!heelContact && midfootContact){ // if not heel but midfoot
    //        if(side == 'L'){    // L side
    //            Tf[KNEE_T] = 10; // 13;
    //            Tf[HIP_T] = -12; //-10;
    //        }else{
    //            Tf[KNEE_T] = 10; // R side
    //            Tf[HIP_T] = -12;
    //        }
    //    }else{
    //        Tf[KNEE_T] = -2;
    //        Tf[HIP_T] = -2;
    //    }
    //     Tf[KNEE_T] = Tf[KNEE_T] - 10; //-5Nm added 00/00/03

    Tf[ANKLE_T] = 0;
    Tf[KNEE_T]  = 0;
    Tf[HIP_T]   = 0;
}
```

Appendix A.20 – Filters.h

```
/* Function: LowpassFilter
 * ------------------------------------------------------------------------------------------
 * Second order filter for the values contained in unfiltered array and outputs the filtered data to filteredArray.
 * Arrays are in the form [data(k) data(k-1) data(k-2)]
 * cutoff specifies the filter cutoff frequency in Hz
 */
void LowpassFilter(double *unfilteredArray, double *filteredArray, double *dataPoint, double *filterCoeffp);

/* Function: DifferentiatingFilter
 * ------------------------------------------------------------------------------------------
 * 3rd order differentiating filter for the values contained in unfiltered array and outputs the filtered data to
filteredArray.
 * Arrays are in the form [data(k) data(k-1) data(k-2) data(k-3)]
 * cutoff specifies the filter cutoff frequency in Hz
 * coefficients are computed in MATLAB using 'differentiating_filter_zoh.m'
 */
void DifferentiatingFilter(double *unfilteredArray, double *filteredArray, double *dataPoint, double *filterCoeffp);

/* Function: LowpassFiltertest
 * ------------------------------------------------------------------------------------------
 * Second order filter for the values contained in unfiltered array and outputs the filtered data to filteredArray.
 * Arrays are in the form [data(k) data(k-1) data(k-2)]
 * cutoff specifies the filter cutoff frequency in Hz
 */
double LowpassFiltertest(double *unfilteredArray, double *filteredArray, double *dataPoint, double *filterCoeffp);
```

Appendix A.21 – Filt2k.h

```
// Generate filter coefficients using command "butter" in Matlab

// define lowpass filter coefs that essentially turn the filter "off" -- i.e. output is exactly = input without phase lag
double filterCoeffsOFF[5]     = {1, 0, 0, 0, 0};


// define Lowpass 1st order filter coefficients {a1, a2, a3, b2, b3} @ 2 kHz sampling time
double filterCoeffs1st025[5]  = {3.925449498358158e-004, 3.925449498358158e-004, 0, -9.992149101003284e-001, 0};
double filterCoeffs1st05[5]   = {7.847819584529403e-004, 7.847819584529403e-004, 0, -9.984304360830941e-001, 0};
double filterCoeffs1st1[5]    = {1.568334083281053e-003, 1.568334083281053e-003, 0, -9.968633318334379e-001, 0};
double filterCoeffs1st2[5]    = {3.131764229192782e-003, 3.131764229192782e-003, 0, -9.937364715416146e-001, 0};
double filterCoeffs1st5[5]    = {7.792936291951547e-003, 7.792936291951547e-003, 0, -9.844141274160969e-001, 0};
double filterCoeffs1st10[5]   = {1.546629140310335e-002, 1.546629140310335e-002, 0, -9.690674171937933e-001, 0};
double filterCoeffs1st20[5]   = {3.046874709125380e-002, 3.046874709125380e-002, 0, -9.390625058174924e-001, 0};
double filterCoeffs1st50[5]   = {7.295965726826670e-002, 7.295965726826670e-002, 0, -8.548066854634670e-001, 0};
double filterCoeffs1st80[5]   = {1.121682444751930e-001, 1.121682444751930e-001, 0, -7.756795110496130e-001, 0};
double filterCoeffs1st100[5]  = {1.367287359973190e-001, 1.367287359973190e-001, 0, -7.265425288053610e-001, 0};
double filterCoeffs1st120[5]  = {1.682003508877370e-001, 1.682003508877370e-001, 0, -6.795992982452680e-001, 0};
double filterCoeffs1st140[5]  = {1.826983512279260e-001, 1.826983512279260e-001, 0, -6.346192975441480e-001, 0};
double filterCoeffs1st160[5]  = {2.043008243802640e-001, 2.043008243802640e-001, 0, -5.913983513994710e-001, 0};
double filterCoeffs1st180[5]  = {2.251226739036150e-001, 2.251226739036150e-001, 0, -5.497546521927700e-001, 0};
double filterCoeffs1st200[5]  = {2.452372752527860e-001, 2.452372752527860e-001, 0, -5.095254494944290e-001, 0};


// define Lowpass 2nd order filter coefficients { a1, a2, a3, b2, b3} @ 2 kHz sampling time
double filterCoeffs2nd025[5]  = {1.54126965e-007,     3.0825393e-007,      1.54126965e-007,     -1.998889279379557,
0.998889895887416};
double filterCoeffs2nd05[5]   = {6.1616576e-007,      1.232331521e-006,    6.1616576e-007,      -1.997778559442931,
0.997781024105973};
double filterCoeffs2nd1[5]    = {0.24619300464102e-5, 0.49238680928203e-5, 0.24619300464102e-5, -1.99555712434579,
0.995566972865986};
double filterCoeffs2nd2[5]    = {0.89825916820472e-4, 0.19651833640943e-4, 0.89825916820472e-4, -1.99111429220165,
0.991153595866894};
double filterCoeffs2nd5[5]    = {0.86180617875887e-3, 0.12281235751613e-3, 0.86180617875887e-3, -1.97778648377676,
0.978883850849180};
double filterCoeffs2nd10[5]   = {0.24135904904196e-3, 0.48271809888392e-3, 0.24135904904196e-3, -1.95557824031504,
0.956543676511200};
double filterCoeffs2nd20[5]   = {0.88094469184384,    0.88188938368768,    0.88094469184384,    -1.91119706742607,
0.914975834801430};
double filterCoeffs2nd50[5]   = {0.80554271721028,    0.81108543442056,    0.80554271721028,    -1.77863177782458,
0.800802646665710};
double filterCoeffs2nd80[5]   = {0.81335920802786,    0.82671840805571,    0.81335920802786,    -1.64745998107698,
0.708896781188400};
double filterCoeffs2nd100[5]  = {0.82088336556421,    0.84816673112842,    0.82088336556421,    -1.56101807580872,
0.641351538056756};
double filterCoeffs2nd120[5]  = {0.82785976611714,    0.85571953223427,    0.82785976611714,    -1.47548044359265,
0.586919508061190};
double filterCoeffs2nd140[5]  = {0.83657483584393,    0.87314967168786,    0.83657483584393,    -1.39889528142539,
0.537194624801100};
double filterCoeffs2nd160[5]  = {0.84613180209331,    0.89226360418663,    0.84613180209331,    -1.30728502884932,
0.491812237222580};
double filterCoeffs2nd180[5]  = {0.85644846226074,    0.91289692452147,    0.85644846226074,    -1.22465158101310,
0.458445430056040};
double filterCoeffs2nd200[5]  = {0.86745527388907,    0.93491054777814,    0.86745527388907,    -1.14298850253990,
0.412801590896190};
double filterCoeffs2nd250[5]  = {0.89763187293782,    0.99526214587564,    0.89763187293782,    -0.94288904158286,
0.333333333333330};
double filterCoeffs2nd300[5]  = {0.93110643991663,    0.06221287983325,    0.93110643991663,    -0.74778917825850,
0.272214937925010};
double filterCoeffs2nd350[5]  = {0.96748380012702,    0.93496760025403,    0.96748380012702,    -0.55703899731175,
0.226966197810020};
double filterCoeffs2nd400[5]  = {0.00657208382615,    0.01314416765230,    0.00657208382615,    -0.36952737735124,
0.195815712655030};
double filterCoeffs2nd500[5]  = {0.09289321881345,    0.585786437626980,   0.09289321881345,    -0.00000000000000,
0.171572875253810};
double filterCoeffs2nd1000[5] = {0.06745527388907,    0.13491054777814,    0.06745527388907,    -1.14298850253990,
0.412801590896190};


// define differentiating Lowpass 3rd order filter coefficients { a1, a2, a3, a4, b2, b3, b4} @ 2 kHz sampling time
// coefficients are computed in MATLAB using 'differentiating_filter_zoh.m'
double DfilterCoeffs1[7]   = {0,  0.12493751562240e-6, -0.00006245314322e-6, -0.12487506247917e-6, -2.99850037493751,
2.99700149950013,  -0.99850112443772};
double DfilterCoeffs2[7]   = {0,  0.99900049983337e-6, -0.00099850116684e-6, -0.99800199866733e-6, -2.99700149950012,
2.99400599600200,  -0.99700449550337};
double DfilterCoeffs5[7]   = {0,  0.15585986287460e-4, -0.00038916300075e-4, -0.15547069987386e-4, -2.99250936719238,
2.98503743757804,  -0.99252805481914};
double DfilterCoeffs10[7]  = {0,  0.12437655989909e-3, -0.00062033068044e-3, -0.12375622921865e-3, -2.98503743757805,
2.97014950124751,  -0.98511193960307};
double DfilterCoeffs20[7]  = {0,  0.99004983374917e-3, -0.00985116044241e-3, -0.98019867330675e-3, -2.97014950124751,
2.94059601992027,  -0.97044553354851};
double DfilterCoeffs50[7]  = {0,  0.01523921737544,    -0.00037625761762,    -0.01486295975782,    -2.92592973688508,
2.85368827350215,  -0.92774348632855};
double DfilterCoeffs100[7] = {0,  0.11890367806259,    -0.00579900080809,    -0.11310467725449,    -2.85368827350214,
2.71451225410788,  -0.86078797642506};
double DfilterCoeffs140[7] = {0,  0.31981108022774,    -0.02162120548595,    -0.29818987474179,    -2.79718145971785,
2.60807470619643,  -0.81858424597019};
double DfilterCoeffs160[7] = {0,  0.47263556934996,    -0.03633794939926,    -0.43629761995070,    -2.76934983915991,
2.55643136689864,  -0.78662786106656};
double DfilterCoeffs200[7] = {0,  0.90483741803596,    -0.08610666495798,    -0.81873075307798,    -2.71451225410788,
2.45619225923395,  -0.74881822068172};
double DfilterCoeffs250[7] = {0,  1.72362676286054,    -0.20253148342420,    -1.52109527943633,    -2.64749870775378,
2.33648234921420,  -0.68728927879897};
```

```
double DfilterCoeffs300[7] = {0,  2.90488942043457,  -0.48462792563377,  -2.58026149480080,  -2.58212392927517,
2.22245466204515,  -0.63762815162177};
double DfilterCoeffs350[7] = {0,  4.49896497068497,  -0.72227723984874,  -3.77668773083623,  -2.51837186238762,
2.11486426915614,  -0.59155536436682};
double DfilterCoeffs400[7] = {0,  6.54984682462385,  -1.18728565633873,  -5.36256036828511,  -2.45619225923394,
2.01096013818692,  -0.54881163609403};
double DfilterCoeffs500[7] = {0, 12.16876223549070,  -2.69172867748078,  -9.47784155800988,  -2.33648234921422,
1.81959197913798,  -0.47236655274182};
```

Appendix A.22 – Filters.c

```
#include <math.h>
#include <stdio.h>

#include "ExoMain.h"
#include "Defines.h"
#include "Filters.h"

extern double filterCoeffsOFF[5];

extern double filterCoeffs1st025[5], filterCoeffs1st05[5],  filterCoeffs1st1[5],   filterCoeffs1st2[5],
              filterCoeffs1st5[5],   filterCoeffs1st10[5],  filterCoeffs1st20[5],  filterCoeffs1st50[5],
              filterCoeffs1st80[5],  filterCoeffs1st100[5], filterCoeffs1st120[5], filterCoeffs1st140[5],
              filterCoeffs1st160[5], filterCoeffs1st180[5], filterCoeffs1st200[5];

extern double filterCoeffs2nd025[5], filterCoeffs2nd05[5],  filterCoeffs2nd1[5],   filterCoeffs2nd2[5],
filterCoeffs2nd5[5],   filterCoeffs2nd10[5],
              filterCoeffs2nd20[5],  filterCoeffs2nd50[5],  filterCoeffs2nd80[5],  filterCoeffs2nd100[5],
filterCoeffs2nd120[5], filterCoeffs2nd140[5],
              filterCoeffs2nd160[5], filterCoeffs2nd180[5], filterCoeffs2nd200[5], filterCoeffs2nd250[5],
filterCoeffs2nd300[5], filterCoeffs2nd350[5],
              filterCoeffs2nd400[5], filterCoeffs2nd500[5],
              filterCoeffs2nd1000[5];

extern double DfilterCoeffs1[7],   DfilterCoeffs2[7],   DfilterCoeffs5[7],   DfilterCoeffs10[7],
              DfilterCoeffs20[7],  DfilterCoeffs50[7],  DfilterCoeffs100[7], DfilterCoeffs140[7], DfilterCoeffs160[7],
              DfilterCoeffs200[7], DfilterCoeffs250[7], DfilterCoeffs300[7], DfilterCoeffs350[7], DfilterCoeffs400[7],
              DfilterCoeffs500[7];


/* Function: LowpassFilter
* ------------------------------------------------------------------------------------------
* Second order butterworth filter for the values contained in unfiltered array and outputs the filtered data to
filteredArray.
* Arrays are in the form [data(k) data(k-1) data(k-2)]
* cutoff specifies the filter cutoff frequency in Hz
* Filter can be designed using MATLAB->Launch Pad->Signal Processing Toolbox->FDA Tool
* or via matlab command line with the "butter" command: [b,a] = butter(<filter order> , <Fcutoff/(Fsample/2)>);
*/
void LowpassFilter(double *unfilteredArray,
                   double *filteredArray,
                   double *dataPoint,
                   double *filterCoeffp){

    double b2, b3, a1, a2, a3; // filter coefficients

    a1 = filterCoeffp[A1]; // 0
    a2 = filterCoeffp[A2]; // 1
    a3 = filterCoeffp[A3]; // 2
    b2 = filterCoeffp[B2]; // 3
    b3 = filterCoeffp[B3]; // 4

    unfilteredArray[2] = unfilteredArray[1]; // update unfiltered data array
    unfilteredArray[1] = unfilteredArray[0];
    unfilteredArray[0] = *dataPoint; // assign current data point to latest value in filter array

    filteredArray[2] = filteredArray[1]; // update filtered data array
    filteredArray[1] = filteredArray[0];

    filteredArray[0] = - b2*filteredArray[1]
                       - b3*filteredArray[2]
                       + a1*unfilteredArray[0]
                       + a2*unfilteredArray[1]
                       + a3*unfilteredArray[2];

    *dataPoint = filteredArray[0]; // filter data point
}


/* Function: DifferentiatingFilter
* ------------------------------------------------------------------------------------------
* 3rd order differentiating filter for the values contained in unfiltered array and outputs the filtered data to
filteredArray.
* Arrays are in the form [data(k) data(k-1) data(k-2) data(k-3)]
* cutoff specifies the filter cutoff frequency in Hz
* coefficients are computed in MATLAB using 'differentiating_filter_zoh.m'
*/
void DifferentiatingFilter(double *unfilteredArray,
                           double *filteredArray,
                           double *dataPoint,
                           double *filterCoeffp){

    double b2, b3, b4, a1, a2, a3, a4; // filter coefficients

    a1 = filterCoeffp[0];
    a2 = filterCoeffp[1];
    a3 = filterCoeffp[2];
    a4 = filterCoeffp[3];

    b2 = filterCoeffp[4];
    b3 = filterCoeffp[5];
    b4 = filterCoeffp[6];

    unfilteredArray[3] = unfilteredArray[2]; // update unfiltered data array
    unfilteredArray[2] = unfilteredArray[1];
    unfilteredArray[1] = unfilteredArray[0];
```

```
    unfilteredArray[0] = *dataPoint; // assign current data point to latest value in filter array

    filteredArray[3] = filteredArray[2]; // update filtered data array
    filteredArray[2] = filteredArray[1];
    filteredArray[1] = filteredArray[0];

    filteredArray[0] = - b2*filteredArray[1]
                       - b3*filteredArray[2]
                       - b4*filteredArray[3]
                       + a1*unfilteredArray[0]
                       + a2*unfilteredArray[1]
                       + a3*unfilteredArray[2]
                       + a4*unfilteredArray[3];

    *dataPoint = filteredArray[0]; // filter data point
}


/* Function: LowpassFiltertest
* ------------------------------------------------------------------------------------------
* Second order filter for the values contained in unfiltered array and outputs the filtered data to filteredArray.
* Arrays are in the form [data(k) data(k-1) data(k-2)]
* cutoff specifies the filter cutoff frequency in Hz
*/
double LowpassFiltertest(double *unfilteredArray,
                         double *filteredArray,
                         double *dataPoint,
                         double *filterCoeffp){

    double b2, b3, a1, a2, a3; // filter coefficients

    a1 =  0.89865221073380e-6; // filterCoeffp[A1];
    a2 =  0.19730442146759e-6; // filterCoeffp[A2];
    a3 =  0.89865221073380e-6; // filterCoeffp[A3];
    b2 = -1.99911142347080;    // filterCoeffp[B2];
    b3 =  0.99911181807964;    // filterCoeffp[B3];

    unfilteredArray[2] = unfilteredArray[1]; // update unfiltered data array
    unfilteredArray[1] = unfilteredArray[0];
    unfilteredArray[0] = *dataPoint; // assign current data point to latest value in filter array

    filteredArray[2] = filteredArray[1]; // update filtered data array
    filteredArray[1] = filteredArray[0];

    filteredArray[0] = - b2*filteredArray[1]
                       - b3*filteredArray[2]
                       + a1*unfilteredArray[0]
                       + a2*unfilteredArray[1]
                       + a3*unfilteredArray[2];

    //    *dataPoint = filteredArray[0]; // filter data point

    return filteredArray[0];
}
```

Appendix A.23 – PCI.h

```
/* Function: InitComm
 *-------------------------
 * This function keeps running until it receives a flag from the Supervisor I/O board indicating that it is communicating
 * through the PCI FPGA. It returns the  register address 'iobase' of the PCI i/o register
 */
int InitComm(long *bufaddrPtr,
             SensorDataT *sensorData);


/* Function: WaitForCounterUpdate
 *-------------------------
 * This function keeps running until it the communication counter of the PCI FPGA has been updated. This should
 * happen every 80 kHz. When the counter has been updated the function returns 1. If the counter has not been
 * updated after 1 sec the function times out and returns a 0.
 */
int WaitForCounterUpdate(long bufaddr,
                         SensorDataT *sensorData);


/* Function: UpdateDataArray
 * -------------------------------
 * Updates the sensor data in dataArray.
 */
int UpdateDataArray(long bufaddr,
                    short *dataArray);


/* Function: UpdateDACs
 * ----------------------------
 * write desired valve voltage to the PCI FPGA DAC
 */
void UpdateDACs(long bufaddr,
                SysPropertiesT *sysProperties,
                SensorDataT *sensorData);


/* Function: CalibrateEncoders
 *---------------------------------
 * This function puts the RIOM in encoder calibration mode and auto-calibrate the system if GUI flag indicates
 * this user option.
 */
void CalibrateEncoders(int flag,
                       SensorDataT *sensorData,
                       long bufaddr);


/* Function: StopPCIcommunication
 * ------------------------------
 * Sets enable bit to 0 in the control register reg_control to stop communication with the PCI FPGA
 */
void StopPCIcommunication(long bufaddr);

/* Function: UpdateRegControl
 * ------------------------------
 * Sets enable bit to 1 in the control register reg_control and sets all other bits to 0
 * Puts the RIOM in encoder calibration mode or auto-calibrate if required by the GUI
 * Switches the request flag if required by the GUI
 */
void UpdateRegControl(long bufaddr,
                      int calibrationFlag,
                      int switch_drequestDone);

/* Function: GUIcommunication
 * ----------------------
 * Receive a command from the GUI via the PCI bus and store it in the sysProperties structure
 */
int GUIcommunication(long bufaddr,
                     SysPropertiesT *sysProperties,
                     SensorDataT *sensorData,
                     short counter,
                     BodyDataT *bodyData);

/* Function: StructToArray
 * ----------------------
 * Takes data from sensorDataT structure and saves it into an array reg_var[]
 */
void StructToArray(SensorDataT *sensorData,
                   float *reg_var,
                   SysPropertiesT *sysProperties);

/* Function: makeAllJointControlTypesMatchMainOperationMode
 * ------------------------------------------------------------------------------------------
 * when the user switches the main control type in the gui (ex: position to VFC), this function goes through the joints
 * and makes sure the individual control type matches the chosen main control type
 */
void makeAllJointControlTypesMatchMainOperationMode(SysPropertiesT *sysProperties);
```

Appendix A.24 – PCI.c

```
#ifdef DJGPP
    #include <dpmi.h>
    #include <pc.h>
    #include <sys/movedata.h>
    #include "_PCILIB.h"
#else
    #include "SimPCI.h"
#endif

#include <malloc.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>

#include "ExoMain.h"
#include "Defines.h"
#include "PCI.h"

#define outl(p,v)   outportl(p,v)
#define inl(p)      inportl(p)
#define DEFAULT_VENDOR_ID     0xEDDD
#define DEFAULT_DEVICE_ID     0xABCA
#define DEFAULT_DEVICE_INDEX  0x00
#define DOTCYCLES              0x100000
#define TIMECYCLES              0x10000000
#define CMDWR                  0x00000001

// import global variables from exoMainTrig.cpp

/*************************************************************************
* init_pcidev(val,devfun,vendor_id,device_id)                            *
* Description: Initializes PF3100 PCI device                             *
*                                                                        *
* 1. Read PCI Command Register                                           *
* 2. Read PCI Base Address Register                                      *
* 3. Determine size of I/O Address block                                 *
* 4. Enable IO and Master capability                                     *
* 5. Return Base Address                                                 *
*                                                                        *
*************************************************************************/
int init_pcidev(int             bus,
                unsigned int    devfun,
                unsigned int    vendor_id,
                unsigned int    device_id){

    int val  = 0;
    int    bar0 = 0;
    printf("Initializing PF3100 PCI Device\n");

    /* Read Base Address Register 0 (BAR0) */
    /* This is allocated by the host's PCI BIOS */
    //      pcibios_read_config_dword(bus,(int) devfun,PCI_BASE_ADDRESS_0,&val);
    printf("   Current BAR_0=%x\n",val);

    /* Save value of BAR0 assigned by PCI BIOS */
    bar0 = val;

    /* Return Base Address for BAR0 */
    return(bar0&0xfffe);
}


/* Function: InitComm
* ------------------------------------------------------------------------------------------
* This function keeps running until it receives a flag from the Supervisor I/O board indicating that it is communicating
* through the PCI FPGA. It returns the  register address 'iobase' of the PCI i/o register
*/
int InitComm(long           *bufaddrPtr,
             SensorDataT    *sensorData){

    int vendor_id    = DEFAULT_VENDOR_ID;
    int device_id    = DEFAULT_DEVICE_ID;
    int device_index = DEFAULT_DEVICE_INDEX;
    int val = 2; // this is the bus number!!
    int sel, read0;
    long bufaddr;
    unsigned short int reg_dac[14];
    int iobase, i;

    #ifdef VISUAL_C_PROJECT
        short read_dac, reg_control, reg_num_var, reg_commtime;
    #else
        unsigned short int read_dac, reg_control, reg_num_var, reg_commtime;
    #endif

    // Find PF3100 PCI Device
    //if (pci_device_find(device_id, vendor_id, device_index, &val, &devfun)==1){
    //   printf("PF3100 PCI device not found ...exiting.\n");
    //   return -1;
    //}
    iobase = 0xe000; // init_pcidev(val, devfun, vendor_id, device_id); // find iobase address
    printf("IOBase is %0x\n", iobase);
    printf("Allocating DOS memory block\n");
    bufaddr  = __dpmi_allocate_dos_memory(8192,&sel); // get buffer address, prev 68=1K, 4096=65KB, 8192=131K
```

```
    printf("bufaddr=%x, sel=%x\n",bufaddr, sel);
    bufaddr  = ((bufaddr + 3)/4)*4;    // word align address
    printf("Word aligned bufaddr=%x, sel=%x\n",bufaddr, sel);
    outl(iobase+20, bufaddr); // write buffer address to IO register
    reg_commtime = (short) (20e6/FREQ); // compute com time as defined by sunny's protocol
    outl(iobase+16, reg_commtime); // set new communication time
    read0 = inl(iobase+20); // read register to make sure address was written correctly
    printf("baseaddr=%x\n", read0);
    for(i=14; i; i--) reg_dac[i-1]=0; // initialize 14 DAC values for valve output data
    outl(iobase+12, 0x00000000); // initialize reg_control to 0
    reg_control = (short) 0x00000000;
    dosmemput(&reg_control, 4, bufaddr); // write reg_control with enable bit = 1 to bufaddr
    dosmemput(reg_dac, 28, bufaddr+4); // set DAC to 0
    reg_num_var = GUI_DATA_ARRAY_SIZE; // 300 variables as of 06-02-2004
    outl(iobase+24, reg_num_var);// set the number of variables sent to the debug port GUI
    *bufaddrPtr = bufaddr; // assign value to argument
    outl(iobase+12, 0x00000000); // set enable flag in IO register, system starts !!!
    return iobase;
}


/* Function: WaitForCounterUpdate
* ------------------------------------------------------------------------------------------
* This function keeps running until it the communication counter of the PCI FPGA has been updated. This should
* happen every 80 kHz. When the counter has been updated the function returns 1. If the counter has not been
* updated after 1 sec the function times out and returns a 0.
*/
int WaitForCounterUpdate(long         bufaddr,
                         SensorDataT *sensorData){

    int updated_flag = 0; // indicates whether the FPGA counter is updating
    int timedOut = 0;
    static unsigned short int reg_counter_old = 0;
    unsigned short int reg_counter = 0;
    int i = 0;


#ifndef VISUAL_C_PROJECT
        dosmemget(bufaddr+202, 2, &reg_counter); // get the counter data (2 bytes)
        if(reg_counter != reg_counter_old){
            updated_flag = 1; // if the counter has been incremented, use new data
        }

        while(updated_flag!=1 && timedOut == 0){ // loop until FPGA data is ready or user quits
            dosmemget(bufaddr+202, 2, &reg_counter); // get the counter data (2 bytes)
            if(reg_counter != reg_counter_old){
                updated_flag = 1; // if the counter has been incremented, quit the loop
            }
        }

        // if you miss a counter, update the lostCommunication counter

        if((reg_counter - reg_counter_old) > 1){
            sensorData->lostCommunication = sensorData->lostCommunication + 1;
            if(sensorData->lostCommunication >= 32000)
                sensorData->lostCommunication = 32000; // saturate
        }
#endif
#ifdef VISUAL_C_PROJECT
            updated_flag = 1; // compiling in Visual C use this dummy value

#endif

    reg_counter_old = reg_counter; // set counter value to previous

    if (updated_flag == 1){
        return 1; // communication is ok
    }else{
        return 0; // communication is lost
    }
}


/* Function: UpdateDataArray
* ------------------------------------------------------------------------------------------
* Updates the sensor data in dataArray using the variables from the PCI port.
*/
int UpdateDataArray(long    bufaddr,
                    short   dataArray[84]){

    int i;

    #ifndef VISUAL_C_PROJECT
        dosmemget(bufaddr+32, 168, dataArray); // get all sensor data (168 bytes = 84 values @ 2 bytes each)
    #else
        for(i=84; i; i--){
            dataArray[i-1] = 0; // if compiling in Visual C set all the data to 0.
        }
    #endif

    return 1;
}


/* Function: UpdateDACs
* ------------------------------------------------------------------------------------------
* write desired valve voltage to the PCI FPGA DAC
*/
void UpdateDACs(long            bufaddr,
```

```
                SysPropertiesT    *sysProperties,
                SensorDataT        *sensorData){

    int cutoff, i, signalDir, valveNumberInJointData;
    static double unfilteredVoltage[6][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // Current A[0]
and Previous values
    static double filteredVoltage[6][4]   = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}}; // used in
lowpass filter
    short reg_dac[10] = {0,0,0,0,0,0,0,0,0,0}; // stores DAC data
    short DACdata[6];
    double voltage[6];
    double saturation = UMAX; // voltage saturation (V)
    double dac_gain[6]     = {LANKLE_DAC_GAIN,      LKNEE_DAC_GAIN,      LHIP_DAC_GAIN,      RANKLE_DAC_GAIN,
RKNEE_DAC_GAIN,      RHIP_DAC_GAIN };
    double dac_offset[6]   = {LANKLE_DAC_OFFSET,    LKNEE_DAC_OFFSET,    LHIP_DAC_OFFSET,    RANKLE_DAC_OFFSET,
RKNEE_DAC_OFFSET,    RHIP_DAC_OFFSET};
    double valve_offset[6] = {LANKLE_VALVE_OFFSET, LKNEE_VALVE_OFFSET, LHIP_VALVE_OFFSET, RANKLE_VALVE_OFFSET,
RKNEE_VALVE_OFFSET, RHIP_VALVE_OFFSET};

    cutoff = 100; // cutoff frequency (Hz)

    for (i=0; i<6; i++){
         switch(i){ // change array index number so it matches convention in jointData array
              case LANKLE_T:
                  signalDir = 1;     // all signalDir signs were changed going from Riom23.bit to Riom31.bit Apr 15
                  valveNumberInJointData = LANKLE;
                  break;
              case LKNEE_T:
                  signalDir = -1;
                  valveNumberInJointData = LKNEE;
                  break;
              case LHIP_T:
                  signalDir = -1;
                  valveNumberInJointData = LHIP;
                  break;
              case RANKLE_T:
                  signalDir = -1;
                  valveNumberInJointData = RANKLE;
                  break;
              case RKNEE_T:
                  signalDir = 1;
                  valveNumberInJointData = RKNEE;
                  break;
              case RHIP_T:
                  signalDir = 1;
                  valveNumberInJointData = RHIP;
                  break;
          }

         // saturate output voltages at +/- 4.5 volts
         if (sensorData->jointData[valveNumberInJointData].valveVoltage > saturation){
             sensorData->jointData[valveNumberInJointData].valveVoltage = saturation;
         }else if(sensorData->jointData[valveNumberInJointData].valveVoltage < -saturation){
             sensorData->jointData[valveNumberInJointData].valveVoltage = -saturation;
         }

         // if valve is on assign voltage
         if(sysProperties->jointControl[i].OperationMode == 0){
             sensorData->jointData[valveNumberInJointData].valveVoltage = DEFAULT_VALVE_IN;
         }

         voltage[i] = (sensorData->jointData[valveNumberInJointData].valveVoltage + valve_offset[i]) * dac_gain[i] +
dac_offset[i];

         DACdata[i] = (short) (voltage[i] * signalDir * D_TO_A_CONVERSION); // Adjust voltage sign since valve ports may be
    }                                                                        // connected to cylinder chambers differently
                                                                             // for each joint

    reg_dac[0] = (short) DACdata[RHIP_T]; // convert to short since the communication protocol only handles
    reg_dac[1] = (short) DACdata[RKNEE_T]; // this type of data
    reg_dac[2] = (short) DACdata[RANKLE_T];
    reg_dac[7] = (short) DACdata[LHIP_T];
    reg_dac[8] = (short) DACdata[LKNEE_T];
    reg_dac[9] = (short) DACdata[LANKLE_T];

    // update DAC data values for PCI FPGA
    dosmemput(reg_dac, 20, bufaddr+4);   // set 20 bytes of dacs
}


/* Function: UpdateRegControl
* ------------------------------------------------------------------------------------------
* Sets enable bit to 1 in the control register reg_control and sets all other bits to 0
* Puts the RIOM in encoder calibration mode or auto-calibrate if required by the GUI
* Switches the request flag if required by the GUI
*/
void UpdateRegControl(long    bufaddr,
                      int    calibrationFlag,
                      int    switch_drequestDone){

    unsigned short int reg_control, reg_control0, reg_control1;

    dosmemget(bufaddr, 2, &reg_control0); // read the current control register

    if(switch_drequestDone) reg_control0 = reg_control0 ^ 0x0001; // switch drequest_done flag to signal that vars have
been sent

    reg_control1 = 0x8000; // set enable bit to 1 and all the rest to 0
```

```
    if(calibrationFlag == 10 || calibrationFlag == 11)    reg_control1 = 0xA380; //  1 0100 0 111 000 000 0
                                                                                 // set enable =1; cmd type = 0100 to calibrate
encoders;
                                                                                 // target addr = 111 (calibrates all encoders)
                                                                                 // target fpga = 0;

    reg_control = (reg_control0 & 0x007F) | (reg_control1 & 0xFF80); // get bits 0-6 from original value and
                                                                                  // bits 8-15 from reg_control1
    dosmemput(&reg_control, 2, bufaddr);
}


/* Function: StopPCIcommunication
* ------------------------------------------------------------------------------------------
* Sets enable bit to 0 in the control register reg_control to stop communication with the PCI FPGA
*/
void StopPCIcommunication(long bufaddr){

    unsigned short int reg_control;

    reg_control = 0x00000000;
    dosmemput(&reg_control, 4, bufaddr); // set enable to 0
}


/* Function: GUIcommunication
* ------------------------------------------------------------------------------------------
* Receives a command from the GUI debug port via the PCI bus and store it in the sysProperties structure.
* Sends variables to the GUI debug port via the PCI bus if request was posted.
* Returns 1 if drequestDone should be switched in reg_control from the debug port to communicate.
*/
int GUIcommunication(long            bufaddr,
                     SysPropertiesT *sysProperties,
                     SensorDataT    *sensorData,
                     short           loopPeriod,
                     BodyDataT       *bodyData){

    int commandID = 0;
    int newCommand = 0;  // commands from GUI
    double commandValue = 0;
    int jointID, subID, i;
    unsigned short int reg_debug;
    short reg_id[2], reg_var[GUI_DATA_ARRAY_SIZE]; // note: reg_id[] can have 4 elements if needed
    float reg_var_float[GUI_DATA_ARRAY_SIZE];
    unsigned short int reg_status, drequestflag, dcommflag, dcmdtype;
    static unsigned short int cmdCounter_old = 0;
    static unsigned short int varCounter_old = 0;
    unsigned short int cmdCounter = 0;
    unsigned short int varCounter = 0;
    unsigned short int reg_control = 0x0000;
    static short oldLoopPeriod = 0;
    int switch_drequestDone = 0;

    dosmemget(bufaddr+200, 2, &reg_status); // read the status register
    drequestflag = (reg_status>>1) & 0x0001; // read data request flag from debug port in status register

    if(drequestflag){ // if the debug port has issued a communication request and is ready to communicate
        dosmemget(bufaddr+204, 2, &reg_debug);    // read reg_debug register which contains the debug system status info
        dcommflag = (reg_debug>>9) & 0x0001; // check if the debug port is communicating
        switch_drequestDone = 1;

        if(dcommflag){ // if the debug port is communicating
            dcmdtype = (reg_debug>>7) & 0x0003; // check cmd type: 1 = request variables, 2 = post command

            if(dcmdtype == 2){
                varCounter = (reg_debug>>4) & 0x0007; // read the read variable counter
                if(varCounter != varCounter_old){ // if counter value has been changed
                    sensorData->loopPeriod = loopPeriod; //counter - oldCounter; // update sampling between two read_vars
                    oldLoopPeriod = loopPeriod;
                    StructToArray(sensorData, reg_var_float, sysProperties); // assign value to variable array to be sent
to debug port
                    for(i=GUI_DATA_ARRAY_SIZE; i; i--){
                        reg_var[i-1] = (short) (reg_var_float[i-1]*1000); // Multiply by 1000 so that you
                                                                          // can pass 3 decimal places in a 16-bit number
                    }
                    reg_var[257] = (short) sensorData->error;
                    reg_var[258] = (short) sensorData->lostCommunication ; // number of times communication with PCI was
lost
                    reg_var[259] = (short) sensorData->loopPeriod; // Supervisor loop period (usec)

                    reg_var[270] = (short) (((long) (sensorData->CounterTicks & 65535)) - 32767); // lower 16 bits
                    reg_var[272] = (short) (((long) (sensorData->CounterTicks >> 16))   - 32767); // higher 16 bits
                    reg_var[274] = sysProperties->recordFlag;

                    dosmemput(reg_var, (GUI_DATA_ARRAY_SIZE*2), bufaddr+216); // save these variables where they're
accessible to the FPGA, why "+216" ?
                    varCounter_old = varCounter;
                }
            }

            if(dcmdtype == 1){  // if a command has been posted
                cmdCounter = (reg_debug>>1) & 0x0007; // check the command counter register
                if(cmdCounter != cmdCounter_old){ // if the counter has been updated
                    dosmemget(bufaddr+208, 4, reg_id); // read the new command (2 cmds, 2 bytes each = 4 bytes)
                    cmdCounter_old = cmdCounter; // reset debug port command counter
                    newCommand = 1;
                }
            }
```

```
        } // if(dcommflag)
    }//    if( drequestflag )

    if(kbhit()){
        printf("reg status 0x%04X, reg_debug 0x%04X \n", reg_status, reg_debug);
    }

    if(newCommand){ // if a new command has been issued, read it and interpret it

        #ifdef VISUAL_C_PROJECT
            reg_id[0] = 0; //  for debugging
            reg_id[1] = 1;
        #endif

        commandID = reg_id[0];
        commandValue = ((double) reg_id[1])/1000; // Convert to double with two decimal places of accuracy
                                    // CommandValue was multiplied by 1000 on the GUI side so as to keep 3 decimal places
                                    // of accuracy when passing it through a 16-bit number via the PCI port

        //printf("reg status 0x%04X, reg_debug 0x%04X, regid0 0x%04X, regid1 0x%04X \n", reg_status, reg_debug, reg_id[0],
reg_id[1]);
        //printf("\n\ncmdID %d, cmdval %6.4f \n", commandID, commandValue);

        if (commandID < 600){ // jointControl commmand
            jointID = commandID/100; // 0 = LANKLE, 1 = LKNEE, 2 = LHIP, 3 = RANKLE, 4 = RKNEE, 5 = RHIP
            subID   = commandID - (jointID*100); // 2 digit number

            switch( subID ){
                case 1:
                    sysProperties->jointControl[jointID].OperationMode = (int) commandValue; // e.g. for lankle commandID
is 001
                    break;
                case 2:
                    sysProperties->jointControl[jointID].kp = commandValue*100; // e.g. for lknee commandID is 102
                    break;
                case 3:
                    sysProperties->jointControl[jointID].kv = commandValue;
                    break;
                case 4:
                    sysProperties->jointControl[jointID].lambda1 = commandValue*1000; // lambda's are large and were not
multiplied
                    break;                                                              // by 1000 on the GUI side
                case 5:
                    sysProperties->jointControl[jointID].lambda2 = commandValue*1000;
                    break;
                case 6:
                    sysProperties->jointControl[jointID].Ci = commandValue*1000;
                    break;
                case 7:
                    sysProperties->jointControl[jointID].eta1 = commandValue;
                    break;
                case 8:
                    sysProperties->jointControl[jointID].eta2 = commandValue*1e-3;
                    break;
                case 9:
                    sysProperties->jointControl[jointID].phi1_inv = commandValue;
                    break;
                case 10:
                    sysProperties->jointControl[jointID].phi2_inv = commandValue;
                    break;
                case 11:
                    sysProperties->jointControl[jointID].ro1 = commandValue;
                    break;
                case 12:
                    sysProperties->jointControl[jointID].ro2_inv = commandValue;
                    break;
                case 13:
                    sysProperties->jointControl[jointID].manualTorque = commandValue*100; // This value was multiplied by
10, not 1000, on the GUI side
                    break;
                case 14:
                    sysProperties->jointControl[jointID].manualValveInput = commandValue;
                    break;
                case 15:
                    sysProperties->jointControl[jointID].desiredPosition = commandValue;
                    break;
                case 16:
                    sysProperties->jointControl[jointID].frequency = commandValue;
                    break;
                case 17:
                    sysProperties->jointControl[jointID].amplitude = commandValue;
                    break;
                case 18:
                    sysProperties->jointControl[jointID].kpFHM = commandValue;
                    break;
                case 19:
                    sysProperties->jointControl[jointID].jointControlType = (int) commandValue;
                    break;
            } // switch

        }else{ // if(commandID < 600){
            switch( commandID){
                case 630:
                    sysProperties->virtualGuard.activation  = commandValue;
                    break;
                case 631:
                    sysProperties->virtualLimit.activation = commandValue;
                    break;
                case 632:
```

```
                    sysProperties->torqueControl.controlFHM = (int) commandValue;
                    break;
                case 640:
                    sysProperties->virtualGuard.position = commandValue;
                    break;
                case 641:
                    sysProperties->virtualGuard.saturation  = commandValue;
                    break;
                case 642:
                    sysProperties->virtualGuard.stiffness  = commandValue;
                    break;
                case 643:
                    sysProperties->virtualGuard.damping  = commandValue;
                    break;
                case 645:
                    sysProperties->virtualLimit.position = commandValue;
                    break;
                case 646:
                    sysProperties->virtualLimit.saturation = commandValue*10; // this value was multiplied by 100, not
1000,
                    break;                                                    // on the GUI side
                case 647:
                    sysProperties->virtualLimit.stiffness = commandValue*10; // this value was multiplied by 100, not 1000
                    break;
                case 648:
                    sysProperties->virtualLimit.damping = commandValue*10; // this value was multiplied by 100, not 1000
                    break;
                case 650:
                    sysProperties->calibration.selection[0] = (int) commandValue;
                    break;
                case 651:
                    sysProperties->calibration.selection[1] = (int) commandValue;
                    break;
                case 652:
                    sysProperties->calibration.selection[2] = (int) commandValue;
                    break;
                case 653:
                    sysProperties->calibration.selection[3] = (int) commandValue;
                    break;
                case 654:
                    sysProperties->calibration.selection[4] = (int) commandValue;
                    break;
                case 655:
                    sysProperties->calibration.selection[5] = (int) commandValue;
                    break;
                case 656:
                    sysProperties->calibration.selection[6] = (int) commandValue;
                    break;
                case 657:
                    sysProperties->calibration.GUIflag = (int) commandValue;
                    break;
                case 660:
                    sysProperties->GUIping = (int) commandValue;
                    break;
                case 670:
                    sysProperties->mainOperationMode = (int) commandValue;
                    makeAllJointControlTypesMatchMainOperationMode(sysProperties); // if you switch the main control mode
in the gui,this makes sure all joints start out using the same mode
                    break;
                case 680:
                    sysProperties->DynDistributionFactor = commandValue;
                    break;
                case 681:
                    sysProperties->ditherAmplitude = commandValue*100; // dither amplitude
                    break;
                case 682:
                    bodyData->heel.mass = commandValue*100; // dither freq.
                    break;
                case 683:
                    sysProperties->recordFlag = (int) commandValue;
                    break;
                case 684:
                    sysProperties->resetTimer = (int) commandValue;
                    break;
                case 686:
                    sysProperties->debuggingControls.activateAnkleSpring = (int) (commandValue*10);    //
activateAnkleSpring
                    break;
                case 687:
                    sysProperties->debuggingControls.springRate = commandValue*10; // springRate
                    break;
                case 688:
                    sysProperties->debuggingControls.centerAngle = commandValue*10; // centerAngle
                    break;
                case 689:
                    sysProperties->debuggingControls.activateKneeDamper = (int) (commandValue*10); // activateKneeDamper
                    break;
                case 690:
                    sysProperties->debuggingControls.flexionDampingCoeff = commandValue*10; // flexionDampingCoefficient
                    break;
                case 691:
                    sysProperties->debuggingControls.extensionDampingCoeff = commandValue*10; //
extensionDampingCoefficient
                    break;
                case 692:
                    sysProperties->debuggingControls.test3 = (int) (commandValue*10); // test3
                    break;
                case 693:
                    sysProperties->debuggingControls.test4 = (int) (commandValue*10); // test4
```

```
                    break;
                } // switch
            } // if(commandID < 600)
        } // if(newCommand)
        return switch_drequestDone;
}


/* Function: StructToArray
* ------------------------------------------------------------------------------------------
* Takes data from sensorDataT structure and saves it into an array reg_var[]
*/
void StructToArray(SensorDataT        *sensorData,
                   float              *reg_var,
                   SysPropertiesT     *sysProperties){

    int i,j;

    reg_var[0] = sensorData->dynamicMode.Mode;              // 0:jump; 1:sgl sup; 2:dbl sup; 3: dbl sup sgl red; 4:dbl sup
dbl red
    reg_var[1] = sensorData->dynamicMode.groundedLeg;        // 1 if the left leg is in contact with the ground
    reg_var[2] = sensorData->dynamicMode.redundantLeg;       // 1 if the left leg is the redundant one (4 dof)
    reg_var[3] = sensorData->dynamicMode.leftHeelContact  ;  // 1 if the left heel is in contact with the ground
    reg_var[4] = sensorData->dynamicMode.rightHeelContact  ; // 1 if the right heel is in contact with the ground

    for (i=0; i<8; i++) {
        j = 18*i;
        reg_var[5+j]  = sensorData->jointData[i].sensorForce/100; // Cylinder force sensor reading (N)
        reg_var[6+j]  = sensorData->jointData[i].position;          // arrays are as follows: [Ltoe Lankle Lknee Lhip Rtoe
Rankle Rknee Rhip ]
        reg_var[7+j]  = sensorData->jointData[i].velocity;
        reg_var[8+j]  = sensorData->jointData[i].acceleration/10;
        reg_var[9+j]  = sensorData->jointData[i].momentArm;       // m
        reg_var[10+j] = sensorData->jointData[i].pistonPosition;  // piston distance from xp0 reference position (m)
        reg_var[11+j] = sensorData->jointData[i].pistonVelocity;  // m/s
        reg_var[12+j] = sensorData->jointData[i].torque/10;       // torque caused by actuator of distal on proximal
segment(N.m)
        reg_var[13+j] = sensorData->jointData[i].Thm/10;          // Joint Torque of human on machine (N.m)
        reg_var[14+j] = sensorData->jointData[i].Tg/10;           // Joint Torque needed to compensate gravity (N.m)
        reg_var[15+j] = sensorData->jointData[i].Tdes/10;         // Desired Joint Torque vector
        reg_var[16+j] = sensorData->jointData[i].Tcc/10;          // Joint torque needed to compensate centrifugal and
coriolis forces (N.m)
        reg_var[17+j] = sensorData->jointData[i].Tf/10;             // Joint friction torque - includes hosing and cable
stiffness (N.m)
        reg_var[18+j] = sensorData->jointData[i].Tlin/10;          // Linearizing torque Tlin = Tg + Tcc + Tf (N.m)
        reg_var[19+j] = sensorData->jointData[i].Tvguard/10;      // Virtual guard torque
        reg_var[20+j] = sensorData->jointData[i].Tvlimit/10;      // Virtual limit torque
        reg_var[21+j] = sensorData->jointData[i].valveVoltage;      // input voltage on the valve
        reg_var[22+j] = sensorData->jointData[i].indexPulse;      // encoder index pulse
    }

    for (i=0; i<5; i++) {
        reg_var[149+2*i] = sensorData->Lfootswitch[i];            // footswitch binary value
        reg_var[150+2*i] = sensorData->Rfootswitch[i];
    }

    // Define these values as constants in define.h for improved speed
    for (i=0; i<7; i++) {   // bodyAccel[ LFOOT LSHANK LTHIGH RFOOT RSHANK RTHIGH UPPERBODY]
        j = 3*i;
        reg_var[159+j] = sensorData->bodyAccel[i].angular_accel;
        reg_var[160+j] = sensorData->bodyAccel[i].lin_accel1;     // linear accelerometer outputs
        reg_var[161+j] = sensorData->bodyAccel[i].lin_accel2;
        //reg_var[156+j] = sensorData->bodyAccel[i].offset1;      // linear accelerometer offsets
        //reg_var[157+j] = sensorData->bodyAccel[i].offset2;
        //reg_var[158+j] = sensorData->bodyAccel[i].gain1;        // linear accelerometer gains
        //reg_var[159+j] = sensorData->bodyAccel[i].gain2;
    }

    reg_var[202] = sensorData->TorsoTilt; // torso inertial sensor output.
    reg_var[203] = sensorData->torsoVelocity;

    for (i=0; i<6; i++){
        reg_var[204+2*i] = sensorData->torsoForce.SG[i];  // torso FT sensor straingage outputs
        reg_var[205+2*i] = sensorData->torsoForce.SGt[i]; // temperature-compensated straingage outputs
    }

    reg_var[204] = sensorData->jointData[LANKLE].againstStop;
    reg_var[206] = sensorData->jointData[LKNEE].againstStop;
    reg_var[208] = sensorData->jointData[LHIP].againstStop;
    reg_var[210] = sensorData->jointData[RANKLE].againstStop;
    reg_var[212] = sensorData->jointData[RKNEE].againstStop;
    reg_var[214] = sensorData->jointData[RHIP].againstStop;

    //reg_var[216] = sensorData->torsoForce.thermister;
    //reg_var[217] = sensorData->torsoForce.Fx/100;  // torso FT sensor straingage outputs
    //reg_var[218] = sensorData->torsoForce.Fy/100;
    //reg_var[219] = sensorData->torsoForce.T/10;

    //debugging Fhm equation changes, added by JRS, 2004-07-20
    reg_var[216] = sensorData->torsoForce.thermister;
    reg_var[217] = sysProperties->jointControl[RANKLE_T].kpFHM; //sensorData->torsoForce.Fx/100; // torso FT sensor
straingage outputs
    reg_var[218] = sysProperties->jointControl[RANKLE_T].kv; //sensorData->torsoForce.Fy/100;
    reg_var[219] = sensorData->torsoForce.T/10;

    reg_var[220] = sensorData->hipData.L_abduction_indexP; // temperature compensated straingauge bias (V)
    reg_var[221] = sensorData->hipData.L_rotation_indexP;
    reg_var[222] = sensorData->hipData.R_abduction_indexP;
    reg_var[223] = sensorData->hipData.R_rotation_indexP;
```

```
    //reg_var[220] = sensorData->torsoForce.SGt_bias[0]; // temperature compensated straingauge bias (V)
    //reg_var[221] = sensorData->torsoForce.SGt_bias[1];
    //reg_var[222] = sensorData->torsoForce.SGt_bias[2];
    //reg_var[223] = sensorData->torsoForce.SGt_bias[3];
    //reg_var[224] = sensorData->torsoForce.SGt_bias[4];
    //reg_var[225] = sensorData->torsoForce.SGt_bias[5];

    reg_var[226] = sensorData->hipData.R_abduction; // unactuated hip joint angles
    reg_var[227] = sensorData->hipData.L_abduction;
    reg_var[228] = sensorData->hipData.R_rotation;
    reg_var[229] = sensorData->hipData.L_rotation;

    reg_var[230] = sensorData->forceDistribution.LankleDistance; // transverse plane distance from CG to ankles
    reg_var[231] = sensorData->forceDistribution.RankleDistance;
    reg_var[232] = sensorData->forceDistribution.weightDistrFactor; // force factor due to gravity (alpha)

    reg_var[233] = sensorData->forceDistribution.filteredBetaFg[0]; // last 4 elements of the filtered load distribution
factor Beta (A[0] = most recent)
    reg_var[234] = sensorData->forceDistribution.unfilteredBetaFg[0]; // ... unfiltered ...
    reg_var[235] = sensorData->forceDistribution.filteredBetaFHM[0];
    reg_var[236] = sensorData->forceDistribution.unfilteredBetaFHM[0];
    reg_var[237] = sensorData->forceDistribution.filteredKrot[0]; // filtered horiz. force factor due to hip rotation
    reg_var[238] = sensorData->forceDistribution.unfilteredKrot[0];

    reg_var[260] = sensorData->GUIping;            // GUI ping response
    reg_var[261] = sensorData->calibrationFlag; // indicates current controller state of accelerometer calibration
    reg_var[262] = sensorData->virtualGuardFx;    // horizontal force caused by virtual guard at upper body CG
    reg_var[263] = sensorData->virtualGuardT;    // hip moment cause by virtualGuardFx

    reg_var[276] = sysProperties->debuggingControls.activateAnkleSpring/10;
    reg_var[278] = sysProperties->debuggingControls.springRate/10;
    reg_var[280] = sysProperties->debuggingControls.centerAngle/10;
    reg_var[282] = sysProperties->debuggingControls.flexionDampingCoeff/10;
    reg_var[284] = sysProperties->debuggingControls.extensionDampingCoeff/10;
    reg_var[286] = sysProperties->debuggingControls.activateKneeDamper/10;
    reg_var[288] = sysProperties->debuggingControls.test3/10;
    reg_var[290] = sysProperties->debuggingControls.test4/10;
}


/* Function: makeAllJointControlTypesMatchMainOperationMode
* ------------------------------------------------------------------------------------------
* when the user switches the main control type in the gui (ex: position to VFC), this function goes through the joints
* and makes sure the individual control type matches the chosen main control type
*/
void makeAllJointControlTypesMatchMainOperationMode(SysPropertiesT *sysProperties){

    int i;

    switch (sysProperties->mainOperationMode){
    case AUTO_TORQUE_CTL:
        for (i=0; i<6; i++){
            sysProperties->jointControl[i].jointControlType = AUTO_TORQUE_CTL;
        }
        break;
    case POSITION_CTL:
        for (i=0; i<6; i++){
            sysProperties->jointControl[i].jointControlType = POSITION_CTL;
        }
        break;

    case MANUAL_TORQUE_CTL:
        for (i=0; i<6; i++){
            sysProperties->jointControl[i].jointControlType = MANUAL_TORQUE_CTL;
        }
        break;
    case VALVE_CTL:
        for (i=0; i<6; i++){
            sysProperties->jointControl[i].jointControlType = VALVE_CTL;
        }
    }
}
```

Appendix A.25 – Record.h

```
/* Function: RecordDataToMem
 * ---------------------------
 * recordsData to arrays for each joint
 */
int InitRecordDataToMem(FILE *FileArray[]);

/* Function: CloseFiles
* ------------------------------------------------------------------------------------------------
* closes all the files used by record
*/
int CloseFiles(FILE *FileArray[]);

/* Function: RecordDataToMem
 * ---------------------------
 * recordsData to arrays for each joint
 */
int RecordDataToMem(RecordedDataArrayT *recordedDataArray, SysPropertiesT *sysProperties, SensorDataT *sensorData,
BodyDataT *bodyData);

/* Function: WriteDataToFlash
 * ---------------------------
 * recordsData to arrays for each joint
 */
int WriteDataToFlash(RecordedDataArrayT *recordedDataArray, FILE *FileArray[]);
```

Appendix A.26 – Record.c

```
#include <math.h>
#include <stdio.h>

#include "ExoMain.h"
#include "Defines.h"


/* Function: RecordDataToMem
* ------------------------------------------------------------------------------------------
* recordsData to arrays for each joint
*/
int InitRecordDataToMem(FILE *FileArray[]){

    FileArray[0] = fopen("../EXODATA/Lankle.exo","a");
    FileArray[1] = fopen("../EXODATA/Lknee.exo","a");
    FileArray[2] = fopen("../EXODATA/Lhip.exo","a");
    FileArray[3] = fopen("../EXODATA/Rankle.exo","a");
    FileArray[4] = fopen("../EXODATA/Rknee.exo","a");
    FileArray[5] = fopen("../EXODATA/Rhip.exo","a");
    FileArray[6] = fopen("../EXODATA/global.exo","a");

    return 1;
}


/* Function: CloseFiles
* ------------------------------------------------------------------------------------------
* closes all the files used by record
*/
int CloseFiles(FILE *FileArray[]){

    fclose(FileArray[0]);
    fclose(FileArray[1]);
    fclose(FileArray[2]);
    fclose(FileArray[3]);
    fclose(FileArray[4]);
    fclose(FileArray[5]);
    fclose(FileArray[6]);

    return 1;
}


/* Function: RecordDataToMem
* ------------------------------------------------------------------------------------------
* recordsData to arrays for each joint
*/
int RecordDataToMem(RecordedDataArrayT    *recordedDataArray,
                    SysPropertiesT        *sysProperties,
                    SensorDataT            *sensorData,
                    BodyDataT             *bodyData){

    int i,j,k;

    for (i=1; i<8; i++) {
        recordedDataArray->jointData[i].indexPulse     = sensorData->jointData[i].indexPulse;
        recordedDataArray->jointData[i].position       = sensorData->jointData[i].position;
        recordedDataArray->jointData[i].velocity       = sensorData->jointData[i].velocity;
        recordedDataArray->jointData[i].acceleration   = sensorData->jointData[i].acceleration;
        recordedDataArray->jointData[i].sensorForce    = sensorData->jointData[i].sensorForce;
        recordedDataArray->jointData[i].momentArm      = sensorData->jointData[i].momentArm;
        recordedDataArray->jointData[i].torque         = sensorData->jointData[i].torque;
        recordedDataArray->jointData[i].Tcc            = sensorData->jointData[i].Tcc;
        recordedDataArray->jointData[i].Tdes           = sensorData->jointData[i].Tdes;
        recordedDataArray->jointData[i].Tf             = sensorData->jointData[i].Tf;
        recordedDataArray->jointData[i].Tg             = sensorData->jointData[i].Tg;
        recordedDataArray->jointData[i].Thm            = sensorData->jointData[i].Thm;
        recordedDataArray->jointData[i].Tlin           = sensorData->jointData[i].Tlin;
        recordedDataArray->jointData[i].valveVoltage   = sensorData->jointData[i].valveVoltage;
        recordedDataArray->jointData[i].pistonPosition = sensorData->jointData[i].pistonPosition;
        recordedDataArray->jointData[i].pistonVelocity = sensorData->jointData[i].pistonVelocity;
    }

    for (j=0; j<6; j++) {
        recordedDataArray->jointControl[j].lambda1      = sysProperties->jointControl[j].lambda1;
        recordedDataArray->jointControl[j].lambda2      = sysProperties->jointControl[j].lambda2;
        recordedDataArray->jointControl[j].Ci          = sysProperties->jointControl[j].Ci;
    }

    recordedDataArray->CounterTicks                   = sensorData->CounterTicks;
    recordedDataArray->dynamicMode.Mode               = sensorData->dynamicMode.Mode;
    recordedDataArray->dynamicMode.groundedLeg          = sensorData->dynamicMode.groundedLeg;
    recordedDataArray->dynamicMode.redundantLeg          = sensorData->dynamicMode.redundantLeg;
    recordedDataArray->dynamicMode.leftHeelContact      = sensorData->dynamicMode.leftHeelContact;
    recordedDataArray->dynamicMode.rightHeelContact   = sensorData->dynamicMode.rightHeelContact;
    recordedDataArray->TorsoTilt                      = sensorData->TorsoTilt;

    for (k=0; k<5; k++) {
        recordedDataArray->Lfootswitch[k] = sensorData->Lfootswitch[k];
        recordedDataArray->Rfootswitch[k] = sensorData->Rfootswitch[k];
    }

    return 1;
}
```

```
/* Function: WriteDataToFlash
* ------------------------------------------------------------------------------------------
* recordsData to arrays for each joint
*/
int WriteDataToFlash(RecordedDataArrayT *recordedDataArray, FILE *FileArray[]){

    //FILE *FileArray[NUM_RECORDED_FILES]; // currently = 7, JRS, 2004-10-28
    int i,j;

//    FileArray[0] = fopen("../EXODATA/Lankle.exo","a");
//    FileArray[1] = fopen("../EXODATA/Lknee.exo","a");
//    FileArray[2] = fopen("../EXODATA/Lhip.exo","a");
    for(i=1;i<=3;i++){
    fprintf(FileArray[i-1],"%f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t
%f\n",
                        recordedDataArray->jointData[i].indexPulse,
                        recordedDataArray->jointData[i].position,
                        recordedDataArray->jointData[i].velocity,
                        recordedDataArray->jointData[i].acceleration,
                        recordedDataArray->jointData[i].sensorForce,
                        recordedDataArray->jointData[i].momentArm,
                        recordedDataArray->jointData[i].torque,
                        recordedDataArray->jointData[i].Tcc,
                        recordedDataArray->jointData[i].Tdes,
                        recordedDataArray->jointData[i].Tf,
                        recordedDataArray->jointData[i].Tg,
                        recordedDataArray->jointData[i].Thm,
                        recordedDataArray->jointData[i].Tlin,
                        recordedDataArray->jointData[i].valveVoltage,
                        recordedDataArray->jointData[i].pistonPosition,
                        recordedDataArray->jointData[i].pistonVelocity,
                        recordedDataArray->jointControl[i-1].lambda1,
                        recordedDataArray->jointControl[i-1].lambda2,
                        recordedDataArray->jointControl[i-1].Ci);
    }
//    fclose(FileArray[0]);
//    fclose(FileArray[1]);
//    fclose(FileArray[2]);

//    FileArray[3] = fopen("../EXODATA/Rankle.exo","a");
//    FileArray[4] = fopen("../EXODATA/Rknee.exo","a");
//    FileArray[5] = fopen("../EXODATA/Rhip.exo","a");
    for(j=5;j<=7;j++){
    fprintf(FileArray[j-2],"%f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t %f\t
%f\n",
                        recordedDataArray->jointData[j].indexPulse,
                        recordedDataArray->jointData[j].position,
                        recordedDataArray->jointData[j].velocity,
                        recordedDataArray->jointData[j].acceleration,
                        recordedDataArray->jointData[j].sensorForce,
                        recordedDataArray->jointData[j].momentArm,
                        recordedDataArray->jointData[j].torque,
                        recordedDataArray->jointData[j].Tcc,
                        recordedDataArray->jointData[j].Tdes,
                        recordedDataArray->jointData[j].Tf,
                        recordedDataArray->jointData[j].Tg,
                        recordedDataArray->jointData[j].Thm,
                        recordedDataArray->jointData[j].Tlin,
                        recordedDataArray->jointData[j].valveVoltage,
                        recordedDataArray->jointData[j].pistonPosition,
                        recordedDataArray->jointData[j].pistonVelocity,
                        recordedDataArray->jointControl[j-2].lambda1,
                        recordedDataArray->jointControl[j-2].lambda2,
                        recordedDataArray->jointControl[j-2].Ci);
    }
//    fclose(FileArray[3]);
//    fclose(FileArray[4]);
//    fclose(FileArray[5]);

//    FileArray[6] = fopen("../EXODATA/globalExo.exo","a"); // added by JRS, 2004-10-28
    fprintf(FileArray[6],"%U\t %d\t %d\t %d\t %d\t %d\t %f\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\t %d\n",
        recordedDataArray->CounterTicks,
        recordedDataArray->dynamicMode.Mode,
        recordedDataArray->dynamicMode.groundedLeg,
        recordedDataArray->dynamicMode.redundantLeg,
        recordedDataArray->dynamicMode.leftHeelContact,
        recordedDataArray->dynamicMode.rightHeelContact,
        recordedDataArray->TorsoTilt,
        recordedDataArray->Lfootswitch[TIP],
        recordedDataArray->Rfootswitch[TOE],
        recordedDataArray->Rfootswitch[BALL],
        recordedDataArray->Rfootswitch[MIDFOOT],
        recordedDataArray->Rfootswitch[HEEL],
        recordedDataArray->Lfootswitch[TIP],
        recordedDataArray->Lfootswitch[TOE],
        recordedDataArray->Lfootswitch[BALL],
        recordedDataArray->Lfootswitch[MIDFOOT],
        recordedDataArray->Lfootswitch[HEEL]);
//    fclose(FileArray[6]);

//    FileArray[7] = fopen("../EXODATA/TorsoData.exo","a"); // not yet used, 06-10-2004

return 1;
}
```

Appendix A.27 – exogpp.bat (make file and compile script)

```
CLS
@ECHO OFF
ECHO ************************************************************************
ECHO *                      Human Engineering Laboratory                    *
ECHO *                  University of California, Berkeley                  *
ECHO *                              2000-2004                               *
ECHO *                                                                      *
ECHO *            EXOGCC.BAT control code compile script for the            *
ECHO *              1st generation UCB-DARPA Human Exoskeleton               *
ECHO ************************************************************************
ECHO *                                                                      *
ECHO *            Naming convention: yymmdd##.exe                           *
ECHO *                  yy = 2 digit year                                   *
ECHO *                  mm = 2 digit month                                  *
ECHO *                  dd = 2 digit day                                    *
ECHO *                  ## = 2 digit build # for the day, starting at 01    *
ECHO *                                                                      *
ECHO *                  ex: 05101327 = 10/13/2005 program #27               *
ECHO ************************************************************************


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Set the drive letter for the compact flash drive here
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
SET flashdrv=F


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Display today's date and prompt for version number
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
XECHO #VToday is: #W, #$N/#$X/#Y # #
IF NOT EXIST c:\xecho\nul MD c:\xecho
IF EXIST c:\xecho\xhelp.bat DEL c:\xecho\xhelp.bat
IF EXIST c:\xecho\xecho_0.bat DEL c:\xecho\xecho_0.bat
XECHO #VInput today's 2 digit code version number and press ENTER: #:0
XECHO #QA#+0>c:\xecho\xhelp.bat
CALL c:\xecho\xhelp
XECHO SET fname=#$Y#$N#$X%NAME%>c:\xecho\xhelp.bat
CALL c:\xecho\xhelp
IF EXIST c:\xecho\xhelp.bat DEL c:\xecho\xhelp.bat
IF EXIST c:\xecho\xecho_0.bat DEL c:\xecho\xecho_0.bat
IF EXIST c:\xecho\nul RMDIR c:\xecho
@ECHO OFF
ECHO.
ECHO File name is: %fname%.exe
ECHO.
PAUSE
@ECHO ON


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: compile individual files into object files
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c ExoMain.c  -o debug\ExoMain.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c Sensors.c  -o debug\Sensors.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c JointCtl.c -o debug\JointCtl.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c Fhm.c      -o debug\Fhm.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c Jump.c     -o debug\Jump.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c SSup.c     -o debug\SSup.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c 1Red.c     -o debug\1Red.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c 2Red.c     -o debug\2Red.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c DSup.c     -o debug\DSup.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c Accel.c    -o debug\Accel.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c Filters.c  -o debug\Filters.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c PCI.c      -o debug\PCI.o
GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer -c Record.c   -o debug\Record.o

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: create a library file
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
CD debug
AR rsc libexo.a ..\_PCILIB.o ExoMain.o Sensors.o Accel.o Filters.o PCI.o Record.o
AR rsc libexo.a JointCtl.o Fhm.o Jump.o SSup.o 1Red.o 2Red.o DSup.o
CD ..

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: link and build entire project
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: GCC -DDOS ExoMain.c exolib.a -O3 -mcpu=pentium -o exe\%fname%.exe

GPP -DDOS -O3 -mcpu=pentium -ffast-math -fomit-frame-pointer debug\libexo.a -o exe\%fname%.exe

::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Copy to floppy disk (uncomment to activate)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: COPY exe\%fname%.exe a:


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Copy to compact flash disk
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
COPY exe\%fname%.exe %flashdrv%:\EXOCODE\
@ECHO OFF
ECHO.
ECHO File copied to COMPACT FLASH on the %flashdrv%:\ drive...
```

```
ECHO.


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Write the runexo batch file on the compact flash
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
@ECHO OFF
ECHO CD exocode >%flashdrv%:\runexo.bat
ECHO %fname% >>%flashdrv%:\runexo.bat


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: Copy to server /archive/exe/ folder (uncomment to activate)
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

:: COPY exe\%fname%.exe w:\DARPAE~1\Control\MainCo~1\Archive\exe\


::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
:: e.g. gcc -DDOS exomain.c exolib.a -O3 -mcpu=pentium -o exe\exodac.exe
:: use -O2 -O3 or -O6 before -o for optimization
:: use -fomit-frame-pointer before -ffast-math (but it might slow down the program)
:: or try -mcpu=pentium after -O2
:: try -funroll-loops
:: exofast3 -O3  works fine 400 usec
:: exofast4 -O3 -funroll-loops  works fine  400 usec (ie funroll-loops is useless)
:: exofast5 -O3 -ffast-math fast 153 usec but doesn't respond to stop
:: exofast6 -O3 -mcpu=pentium  works fine  400 usec (ie -mcpu=pentium is useless)
:: exofast6 -O3 -fomit-frame-pointer doesn't run
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
ECHO.
PAUSE
```

Appendix B

BLEEX Proposed Testing Protocol

The following section is an initial draft of a proposed human testing protocol for BLEEX. The protocol was developed in the Spring of 2005 and follows the guidelines set by the University of California at Berkeley Committee for Protection of Human Subjects. This human subject testing plan was not implemented because the decision was made to delay the human testing portion of the project until the design of second generation exoskeleton hardware was complete in 2006. In addition, the decision was made to have human subject testing performed by an outside indepentant evaluator. This protocol has been included in this appendix for reference purposes only.

1. Title:

Lower Extremity Enhancer

2. Related Projects

None

3. Nature and Purpose

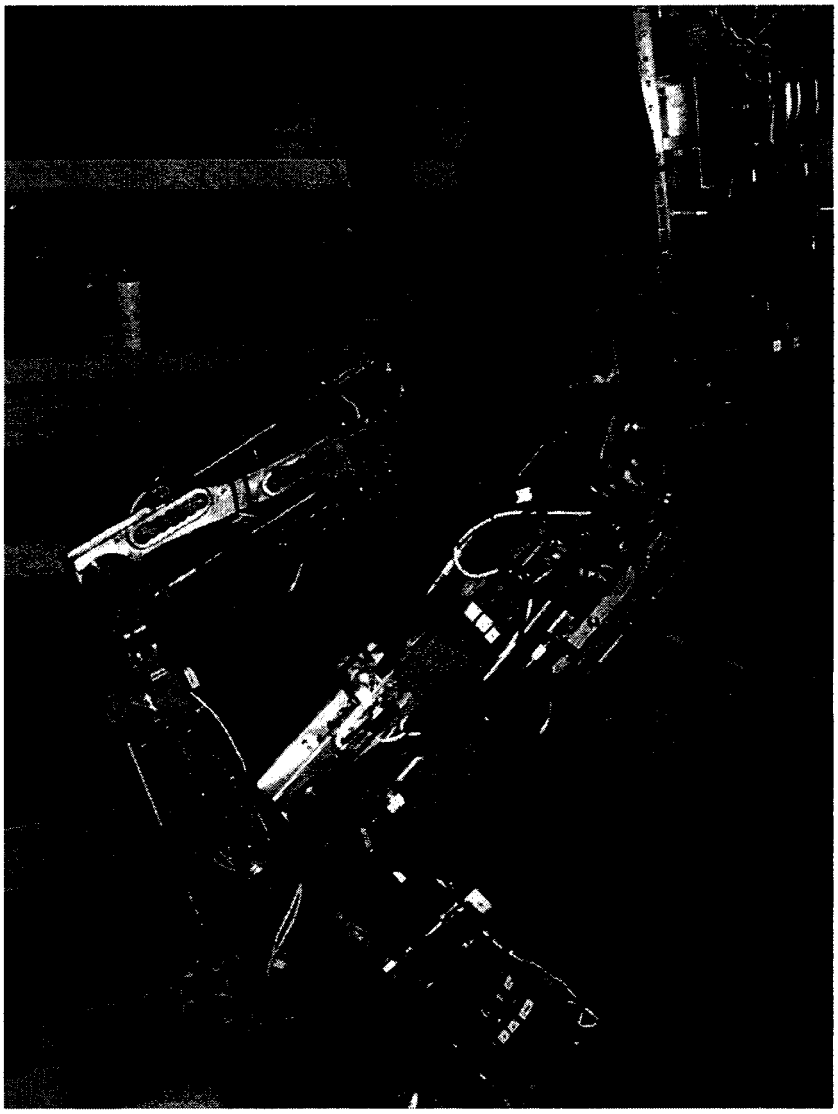
The objective of our research work is to design, construct and demonstrate a Lower Extremity Enhancer for strength and endurance enhancement of individuals carrying heavy backpacks such as soldiers and hikers. The proposed device will provide a person with the ability to carry loads with minimal effort over any type of terrain for extended periods of time. The Lower Extremity Enhancer will allow a person the degrees of freedom to comfortably squat, bend, swing from side to side, twist and walk on ascending and descending slopes, as well as to step over and under obstructions while

carrying equipment and supplies. Using the Lower Extremity Enhancer a person should be able to walk while carrying significant loads over considerable distances without reducing his/her agility.

4. Subjects

The performance of the Enhancer will be quantified with the help of a set of experimental results during the 5th year of the research work (2005). We already have a current CPHS approval (2003-10-108) for comfort evaluation of the device. The graduate students who designed and built the system in the lab needed to wear the device to test the device components and evaluate the device's comfort using the current CPHS approval (2003-10-108). We recently finished the construction of a new experimental Enhancer (see next page). We now need to evaluate the Enhancer performance via a larger population of the test subjects. This document describes the protocol for the evaluation of the Enhancer. The major difference between this application and (2003-10-108) is the experimental procedure described in Section 7 and the letter of consent. Dr. George Brooks will be advising us during this experimental evaluation.

To evaluate the Enhancer's performance, we will use qualified student volunteers as research subjects. Specifically, the research subjects will be a sample of male students between 18 and 25 years of age. The individuals should all be in good to excellent physical condition and not suffer from any heart, respiratory, or back related problems that could pose a danger to their health. Individuals must be comfortable with prolonged endurance tests and be able to carry significant backpack loads. Viable candidates should be able to walk with backpack loads of at most 50% of their bodyweight for short durations (10-20 minutes). The subjects were previously asked to carry 90 lbs with only 10 lbs of force on their shoulders.



Experimental Lower Extremity Enhancer

5. Recruitment

We prefer to recruit ROTC students because of their usual physical training. The study involves significant physical exertion to near exhaustion and participants should be accustomed to such rigors. I intend to discuss this experimental research with the campus ROTC officer and give him this protocol (in particular the Consent Letter to Subjects). Interested subjects will be instructed to contact Professor Kazerooni. The subjects will be asked about their existing physical fitness and their willingness to operate the Enhancer

according to the protocol described below.

6. Screening Procedure

All potential subjects with joint pain, history of problems or surgery in the neck, back and lower extremities, history of back strain, hip injury or hip replacement will be disqualified. Persons with asthma, angina and/or other possible cardiac and/or respiratory problems in their general health history will be disqualified as participant in this study. Participants must be students between 18 and 25 years old in age. Participants will be English-speaking. A separate screening consent form is prepared to qualify potential subjects in greater detail.

7. Procedure

5 test subjects will be chosen to participate in all experiments described below (Experiment A4 is optional). There will be at least a three-day rest period between each experiment. Should any of the subjects show any symptoms of cardio-respiratory distress or other injury, the testing will cease immediately and emergency personnel notified if necessary. Furthermore, if at any stage in testing the subject shows signs that continued testing at higher payloads may be unsafe, testing will cease.

Experiment A1 (about 2 hours)

Each subject is asked to walk on a treadmill for the period of 20 minutes while carrying a backpack with approximately 30% of their bodyweight at the speed of 2 mph. During this experiment, the oxygen consumption and heart rate will be measured. Immediately after the completion of this period, the subject is asked to run on a treadmill at 1.7 mph at a 10% incline (stage 1) without any backpack or enhancer. After 3 minutes the speed will be increased to 2.5 mph and the slope increased to 12% incline (stage 2).

After another 3 minutes the speed will be increased to 3.4 mph and the slope increased to 14% incline (stage 3). After another 3 minutes the speed will be increased to 4.2 mph and the slope increased to 16% incline (stage 4). After another 3 minutes the speed will be increased to 5 mph and the slope increased to 18% incline (stage 5). After another 3 minutes the speed will be increased to 5.5 mph and the slope increased to 20% incline (stage 6). The test will advance through either all 10 stages or until the subject no longer wishes to continue, whichever occurs first (Bruce Treadmill Test).

Experiment A2 (about 2 hours)

The same as experiment A1, except the backpack will weigh 40% of bodyweight.

Experiment A3 (about 2 hours)

The same as experiment A1, except the backpack will weigh 50% of bodyweight.

Experiment B1 (about 2 hours)

Each subject is asked to walk on a treadmill for the period of 20 minutes while carrying a backpack with approximately 30% of their bodyweight at the speed of 2 mph using an enhancer. During this experiment, the oxygen consumption and heart rate will be measured. Immediately after the completion of this period, the subject is asked to run on a treadmill at 1.7 mph at a 10% incline (stage 1) without any backpack or enhancer. After 3 minutes the speed will be increased to 2.5 mph and the slope increased to 12% incline (stage 2). After another 3 minutes the speed will be increased to 3.4mph and the slope increased to 14% incline (stage 3). After another 3 minutes the speed will be increased to 4.2 mph and the slope increased to 16% incline (stage 4). After another 3 minutes the speed will be increased to 5mph and the slope increased to 18% incline (stage 5). After another 3 minutes the speed will be increased to 5.5 mph and the slope increased

to 20% incline (stage 6). The test will advance through either all 6 stages or until the subject no longer wishes to continue, whichever occurs first (Bruce Treadmill Test).

Experiment B2 (about 2 hours)

The same as experiment B1, except the backpack will weigh 40% of bodyweight.

Experiment B3 (about 2 hours)

The same as experiment B1, except the backpack will weigh 50% of bodyweight.

Experiment B4 (Optional, about 2 hours)

The same as experiment B1, except the backpack will weigh 60% of bodyweight. This experiment will be performed only if the trend observed in experiments B1-B3 indicate that the energy expenditure expected when carrying 60% of bodyweight with the Enhancer will be less than or comparable to that carrying 50% of bodyweight without Enhancer (experiment A3) and the subject consents.

The table below shows the summary of the experiments. Each experiment with setup time will take about two hours. Each subject will spend approximately 14 hours for this research. There will be a three-day rest period between each experiment.

Payload (% of bodyweight)	Experiment	Time
30%	A1	About two hours
30% + Exo	B1	About two hours
40%	A2	About two hours
40% + Exo	B2	About two hours
50%	A3	About two hours
50% + Exo	B3	About two hours
60% + Exo	B4 (optional)	About two hours
Total		About 14 hours

Since the screening procedures exclude any individual(s) with joint pain, joint replacement surgery of the back and lower extremities, prior back and neck problems, prior back and neck surgery, asthma, angina and/or other possible cardiac and/or respiratory problems, a physician need not be present in the laboratory during experimentation. All experiments are carried out on a voluntary basis – hence the subject may end the test at the onset of any significant discomfort that could be an indicator of cardio-respiratory distress or other injury. Participants will be advised to report any symptoms of possible cardio-respiratory distress including chest pains, shortness of breath, arm, back, jaw or neck pain, nausea, light-headedness, sudden numbness or weakness (especially in one side of the body), dizziness, loss of balance, sudden confusion, trouble seeing, or sudden headache. The testing will be discontinued and emergency personnel contacted immediately should the subject exhibit any of these symptoms. As a precaution, at least one researcher with a current American Red Cross accreditation in CPR for the Professional Rescuer will be present in the laboratory during all testing procedures.

Orientation

Enhancer evaluation.

Expected Findings

With the experimental data, we will make histograms of the physiological statistics (mean and variance for oxygen consumption, minute ventilation, and heart rate) for the two modes of operations: with and without the help of the Enhancer. The comparative results from these analyses will evaluate the Enhancer's performance. For example, we learn the percentage decrease in heart rate or oxygen consumption when the Enhancer is used compared to when it is not used.

Several previous studies have demonstrated that the metabolic cost of backpack load carriage (measured by oxygen consumption) increases steadily with payload until about 40% of bodyweight, after which it increases sharply. This documented sharp energy cost increase has been cited by previous studies as the basis for limiting continuous or laboratory environment controlled load carriage to 40% of bodyweight. A limit of 50% bodyweight is used for intermittent carrying. (The more conservative rule of thumb in outside laboratory, continuous, long duration experiments has been 1/3 of bodyweight.) Several studies run by the military have included load carriage of much greater than 50% of bodyweight for extended distances since operational requirements often force soldiers to perform similar tasks.

> *1. Body weight*
>
> **That the maximum comfort load should be related to body weight is an idea of long standing. The weight carried by the soldier steadily increased during World War I up to 85% of body weight (Renbourn, 1954b). On the basis of the energy cost of load carriage, which rose steeply above 40% of body weight, Cathcart *et al* (1923) recommended that under laboratory conditions the maximum load for the maintenance of efficiency and health should be 40% of body weight, and for service conditions they accepted the traditional limit of one-third of body weight. Marshall (1950) cites the British and other studies to recommend an optimal marching load of not more than one-third of body weight.**
>
> Kinoshita (1985) cites a number of studies to support the statement that the weight of the individual load should not exceed 40% of body weight for continuous carrying, and 50% for intermittent or occasional carrying. McFarland (1969), reviewing the safety of carrying objects in occupational tasks, considered that objects carried with hand grips become 'heavy' at about 35% of body weight.

We expect to find that the use of the Enhancer may result in slightly increased metabolic costs at lower payloads (due to the weight of the Enhancer itself), but will

mitigate the dramatic energy cost increase normally seen at loads in excess of 40% bodyweight – thus it is vital that the testing protocol compare the energetic costs of motion at payloads exceeding the 40% of bodyweight threshold at which energetic cost typically increases sharply. It is expected that the energy cost of carrying large payloads will be significantly reduced by the Lower Extremity Enhancer. The Enhancer should therefore increase the maximum safe load carrying capacity of the wearer.

An opposing school of thought supported by performance based testing indicates that load carriage ability is not limited by energetic costs, but by localized fatigue. The reasoning is that a person cannot physically carry heavy loads at a given speed not so much due to the large energetic costs associated with the load, but due to fatigue of individual muscle groups. This explains the results of some studies in which individuals perform better and prefer to move in ways or at speeds that are documented to have higher metabolic costs. By performing a post-load-carriage graduated Bruce Treadmill Test, the relative level of localized muscle fatigue can be indirectly gauged from the performance of the subject. It is expected that by supporting a fraction of the payload mass from the ground independent from the wearer (much like a hand-truck), the Enhancer will reduce the muscular fatigue on the subject and increase his/her post-load-carriage performance.

8. Benefits

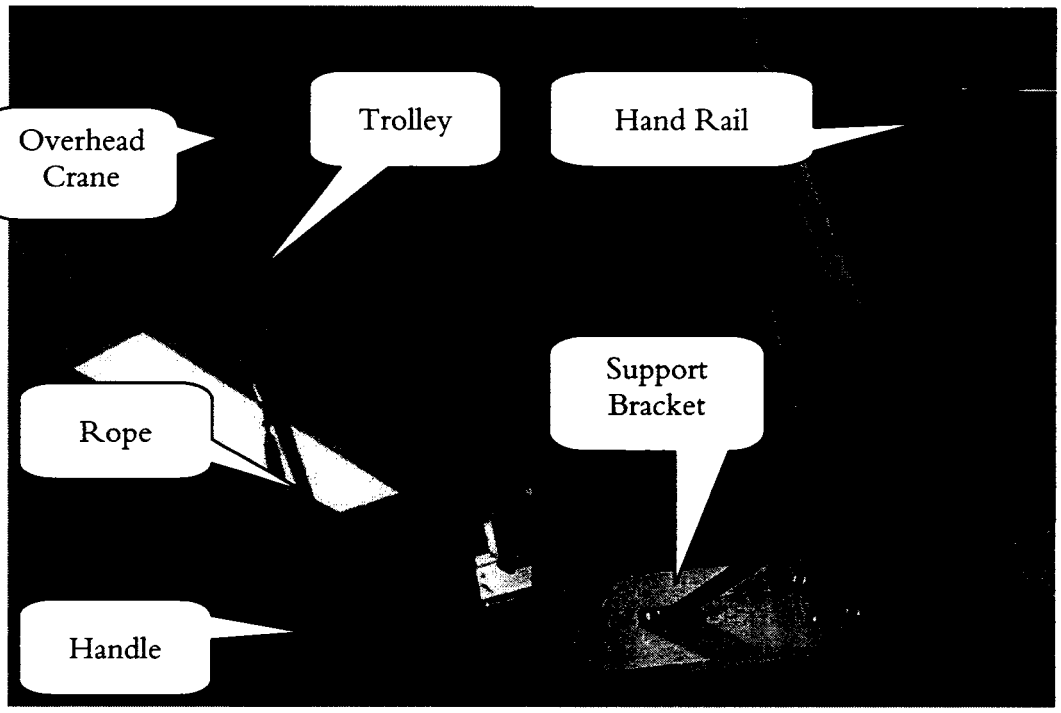
We hope that the research will benefit society. In particular we hope the results of this research work will lead to a technology that can help people carry loads without injuries.

9. Risk

Several safety features have been designed and built into this experimental system. The system has been designed so that the test subject can easily disable and exit from the

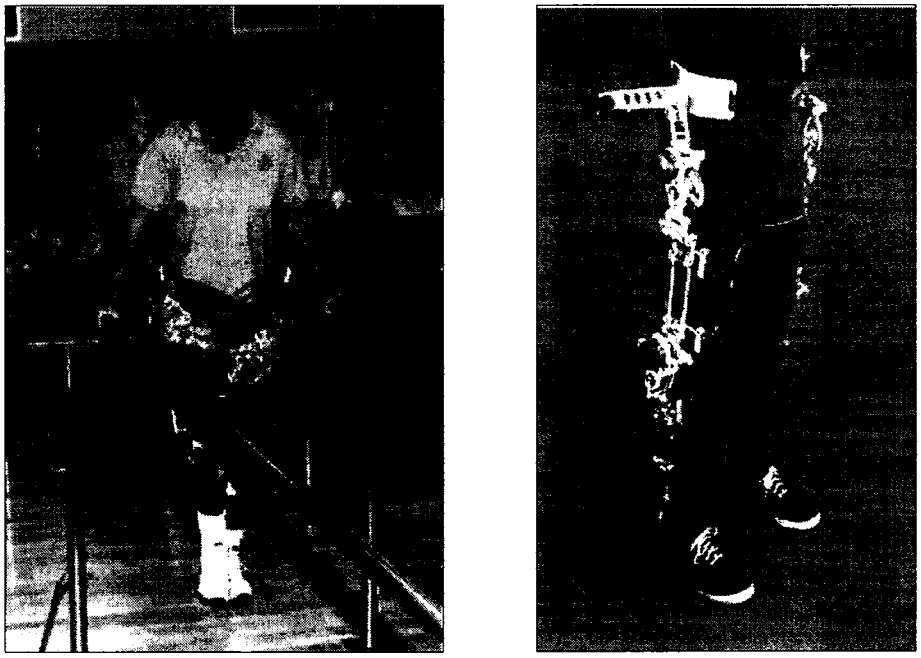
device. We have installed mechanical stops to prevent the device mechanisms from moving beyond the maneuvering space deemed humanly possible. A panic button is installed at a location close to the person monitoring the subject. The subject will also have a panic button, which allows him to stop the treadmill and exit from it at will. We have installed an overhead crane so the device is always hung from ceiling. This overhead crane allows the wearer to walk normally along the crane, but it does not allow the wearer to fall. Additionally two hand rails are installed on the floor that the subject can hold on. These features are shown on the next page. All persons conducting and participating in the experiments will be instructed about these safety features and means of activating them prior to each test cycle. Should an accident happen while the Enhancer is being tested, we will call 911 immediately.

The testing protocol involves physical exertion until exhaustion and thus involves some risk. This risk is mitigated by pre-screening to eliminate candidates with contributing risk factors and by having CPR trained personnel present at all times. Candidate subjects should all be physically fit, young and healthy individuals who strenuously exercise regularly, hence the risk should be minimal. Subjects will be advised to end the test at any time they feel their health is threatened.



The Photo on the left shows the overhead crane, trolley and a rope to hold the device. This system guarantees that the subject will not fall down. The trolley travels smoothly along the red rail when the subject walks. The system also has a handle that allows the rope to be slack or tight.

The Photo on the right shows the rail system (yellow) and its support bracket (Aluminum). The support bracket and the metal rail create a rigid frame where the wearer can always hold on with is hands during experiments. These safety features are quite common in many biomechanics laboratories as shown below.



Computer-controlled Orthotic device for paraplegic individuals, MIT Biomechanics Laboratory

10. Confidentiality

No test subject's name nor any identifying information, particularly personal information about a test subject's health during the qualification process, will be published in any reports and articles nor disclosed by any other means. I, Professor Kazerooni, am the only person to come in contact with the test subjects' health and qualification data. I will not show qualification information of the test subjects to anybody, nor will I disclose this information to any person or organization. I will destroy the Screening Consent letters of individuals who do not participate in the study. All participants will be identified by numbers (e.g. subject 1, subject 2, ...). The data (oxygen consumption and heart beats) will be correlated to the subject number rather than name or other personal information.

11A. SCREENING Consent Form
(To be signed by subjects at the time of screening).

My name is Hami Kazerooni. I am a faculty member in the mechanical engineering at the University of California at Berkeley. I would like you to take part in our research, which deals with assist devices used by individuals carrying heavy backpacks. To qualify you for this experimental study, please provide answers to all of the following questions:

What is your age? _____Years

What is your height? _____Feet_____Inches

Have you had joint pain? Yes ___ No___

Have you had invasive procedures or surgery (e.g., lumbar punctures)

performed on your back Yes ___ No___

Do you have any history of back strain? Yes ___ No___

Have you had joint replacement surgery in your lower extremities? Yes ___ No___

Have you had hip injury or replacement surgery? Yes ___ No___

Have you had prior back problems or surgery? Yes ___ No___

Have you had prior neck problems or surgery? Yes ___ No___

Have you had prior knees problems or surgery? Yes ___ No___

Have you had prior feet problems or surgery? Yes ___ No___

Have you had prior ankle problems or surgery? Yes ___ No___

Have you had prior leg problems or surgery? Yes ___ No___

Have you had asthma Yes ___ No___

Have you had angina Yes ___ No___

Have you had any respiratory conditions or problems Yes ___ No___

Have you had any heart (cardiac) conditions or problems? Yes ___ No___

Have you had any issues in your general health history, which may prevent your participation in this study? Yes ___ No___

Please explain:_____

Do you exercise regularly Yes ___ No___

Do you exercise regularly to exhaustion? Yes ___ No___

Would you feel comfortable attempting to carry a backpack with 50% of your bodyweight? Yes ___ No___

What is the heaviest backpack you feel you could safely carry for 20 min? _____lb

Your name and any identifying information, in particular your health data provided in this form, will not be published or disclosed in any reports and articles. I will never make copies of this form nor show this information to anybody. I will be the only person who comes in contact with this information and I will not disclose this information to any person or organization. If you do not participate in the study for any reason, I will destroy your consent form for this study. Your qualification or disqualification for this research will have no influence on your standing in school or on your ROTC status.

The study you are being asked to participate in involves carrying large backpack loads for short durations (~20 min at a time) and may involve considerable exertion. You should be comfortable carrying considerable backpack loads (up to 50% of your bodyweight) if

you wish to participate. You should also be confident that you can safely exert yourself to near exhaustion without any dangerous health complications. You will be required to wear a personal disposable oxygen sensor mask and a heart rate measuring chest strap during testing. If at any time you feel unable to safely continue or you exhibit any symptoms of cardio-respiratory distress or other injury, the test will immediately cease and emergency personnel contacted if necessary. Symptoms may include chest pains, shortness of breath, arm, back, jaw or neck pain, nausea, light-headedness, sudden numbness or weakness (especially in one side of the body), dizziness, loss of balance, sudden confusion, trouble seeing, or sudden headache. There will be at least one researcher trained and certified in CPR by the American Red Cross present at all times.

If you have any questions regarding the way in which you will be treated as a human subject in this experiment, you may consult Committee for Protection of Human Subjects, 101 Wheeler Hall, University of California, Berkeley, CA 94720-1340, 510-642-7461.

I have read this consent form and I have answered the above questions.

Signature: _____ Date: _____

11B. Consent Letter to Subjects (with University letterhead)

[omitted]

12. Financial Aspects

Each student will be paid a flat rate of $200.

13. Written Materials

None

14. Signatures

H. Kazerooni

15 Telephone Numbers

(510) 642-2964