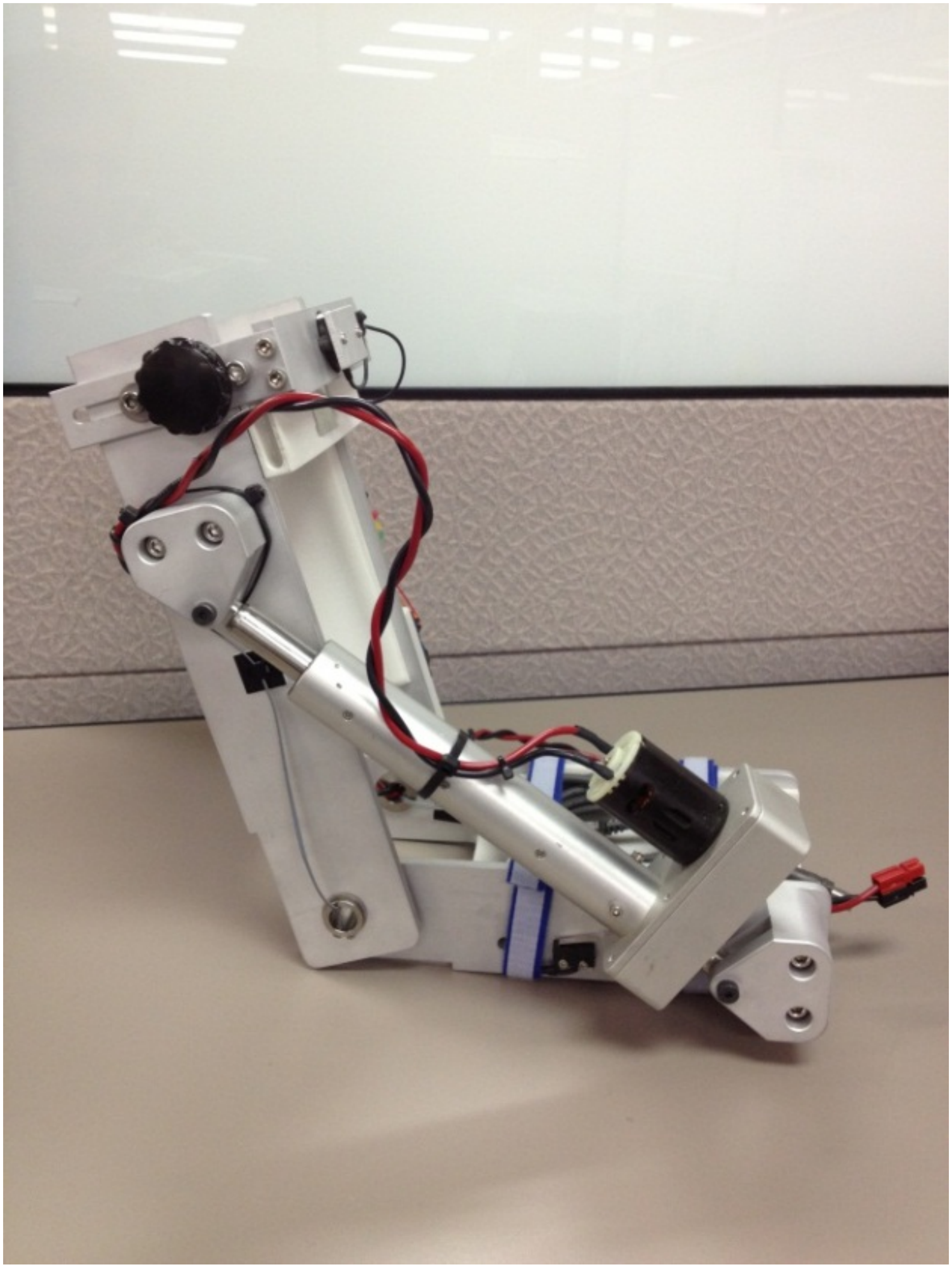
Development of a Rehabilitative Exoskeleton

Daniel Garcia
Victor Soto
Yojans Lurbe
Sabri Tosunoglu

Department of Mechanical and Materials Engineering
Florida International University
Miami, Florida 33174

April 2012



Contents

Figures 4
Tables 4
Introduction 5
    Problem Statement 5
    Literature Review 5
Conceptual Design 6
Theoretical Work 6
Major components 8
Design Drawings 9
Programming 9
Circuitry 10
Prototype Construction 10
Testing and Test Results 15
Conclusions and Future Work 16
References 16
Appendix A 17
Appendix B 21
Appendix C – Program 52
    Flow Chart 52
    Code 53
Appendix D - Circuit Chart 58

Figures

Figure 1 - Motion Schematic 6
Figure 2 - Static Equilibrium Equations 7
Figure 3 - Actuator Torque 7
Figure 4 - Dynamic Equations 8
Figure 5 - Major Components 9
Figure 6 - Frame Assembly 11
Figure 7 - Linear Actuator Drive 11
Figure 8 - Linear Actuator Drive Side 12
Figure 9 - Linear Actuator Mounted 12
Figure 10 - Frame Assembly Sandblasted 13
Figure 11 - Electronics Connected 13
Figure 12 - Electronic Connections 14
Figure 13 - Final Assembly Right Side 14
Figure 14 - Final Assembly Left Side 15
Figure 15 - Program Logic 52
Figure 16 - Circuit Diagram 58

Tables

Table 1 - Bill of Materials 8

Introduction

Problem Statement

Reduced motor functionality of the upper extremities is one of the most common impairments resulting from spinal cord injuries, occupational and sports injuries, strokes, and other diseases. Partial or even full sensorimotor recovery can be attained through intense and repetitive exercises. In recent years robotic assisted rehabilitation has been shown to improve treatment outcomes in these cases [1].

Literature Review

Current exoskeleton rehabilitation devices have multiple advantages over traditionally manual techniques, including [1]:

- Data tracking for performance feedback
- The ability to apply controlled forces at each joint as well as magnitude adjustment of such forces based on patient needs
- They can be adjusted for multiple limb sizes to fit different patients
- They can replicate the majority of the patient's upper limb healthy workspace, using multiple degrees of freedom.

Our device contains additional advantages over current devices. First of all it will be portable. It is going to address a very specific task, which makes it more user friendly, and last but not least it has a simple and cost effective design.

This bicep & tricep therapeutic device will have three modes of operation: passive, active-assisted, and active-constrained. A linear actuator provides the necessary movement of the exoskeleton and a pair of force sensors tracks the response of the patient to the therapeutic session. The passive mode is for patients that have complete muscle atrophy. In this mode the actuator does all the work to emotionally stimulate the patient. In the active-assisted mode in the case of bicep rehabilitation, the patient exerts a slight force, which increases the load on the force sensor and triggers the actuator to contract or extend the exoskeleton. Last but not least, in the active-constrained mode, the user must apply a load on the load sensor the passes a certain threshold. When the robot detects this, it moves the actuator at a speed that creates resistance for the user. In both of the active modes, if the load applied by the user falls below the threshold, the actuator stops.

Considering the portable nature of this design, in the active-assisted mode the exoskeleton can also be used to multiply force. This will allow patients with weak muscles to perform everyday tasks such as lifting, pushing, pulling, etc. For these tasks a hook was added to the part of the frame that supports the forearm.

Conceptual Design

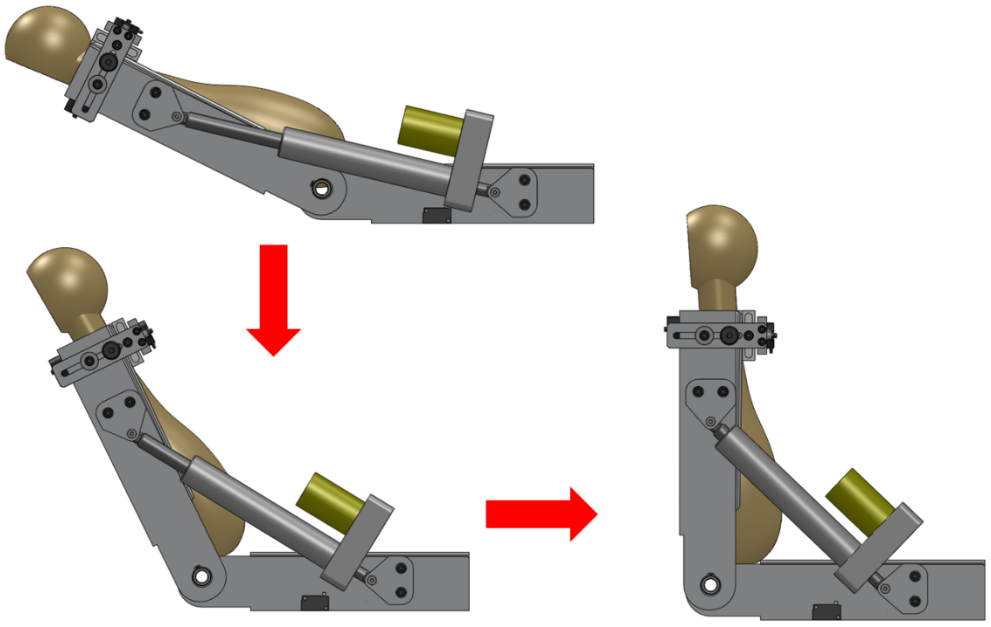


Figure 1 - Motion Schematic

The only conceptual design considered as part of this system is shown in figure 1. This basic 1 Degree-of-Freedom (DOF) system consists of two links connected by a revolute joint. A linear actuator adjusts the angular separation between the two links as shown by the sequence in the figure.

Theoretical Work

As shown in figure 2, based on an external applied force of 50 lb, the resulting external torque of this 1-DOF system is 22.9 lb-ft.

Static Equilibrium

$$a1 := 0.916 = 0.916 \phi 1 := \frac{\pi}{6} == \frac{1}{6}\pi F := 50$$

$$Fhx := F \cdot \sin(\phi 1) = 25$$

$$Fhy := F \cdot \cos(\phi 1) = 25\sqrt{3}$$

$$Xh = a1 \cdot \cos(\phi 1) = Xh = 0.4580000000\sqrt{3}$$

$$Yh = a1 \cdot \sin(\phi 1) = Yh = 0.4580000000$$

$$Gh1x := -a1 \cdot \sin(\phi 1) = -0.4580000000$$

$$Gh1y := a1 \cdot \cos(\phi 1) = 0.4580000000\sqrt{3}$$

$$Fh := \langle \mathrm{Fhx}, \mathrm{Fhy} \rangle = \begin{bmatrix} 25 \\ 25\sqrt{3} \end{bmatrix}$$

$$Jt := \begin{bmatrix} Gh1x & Gh1y \end{bmatrix} = \begin{bmatrix} -0.4580000000 & 0.4580000000\sqrt{3} \end{bmatrix}$$

$$\tau^{e} = -Jt.\ Fh = \tau^{e} = -22.90000000$$

Figure 2 - Static Equilibrium Equations

The force vs. speed curve (Appendix B) of the linear actuator were then used to estimate the torque applied by the actuator, as shown in figure 3. The torque (22.86 lb-ft) available from the actuator at the desired speed is very close to the required torque (22.9 lb-ft) of the external 50 lb load shown in figure 2. The system is therefore theoretically rated up to fifty pounds.

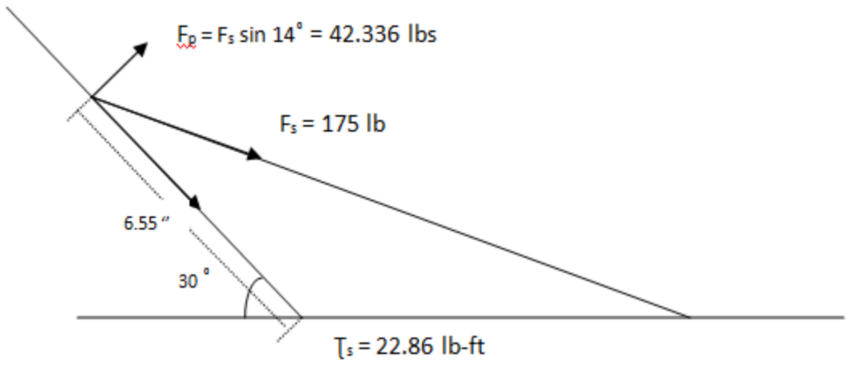
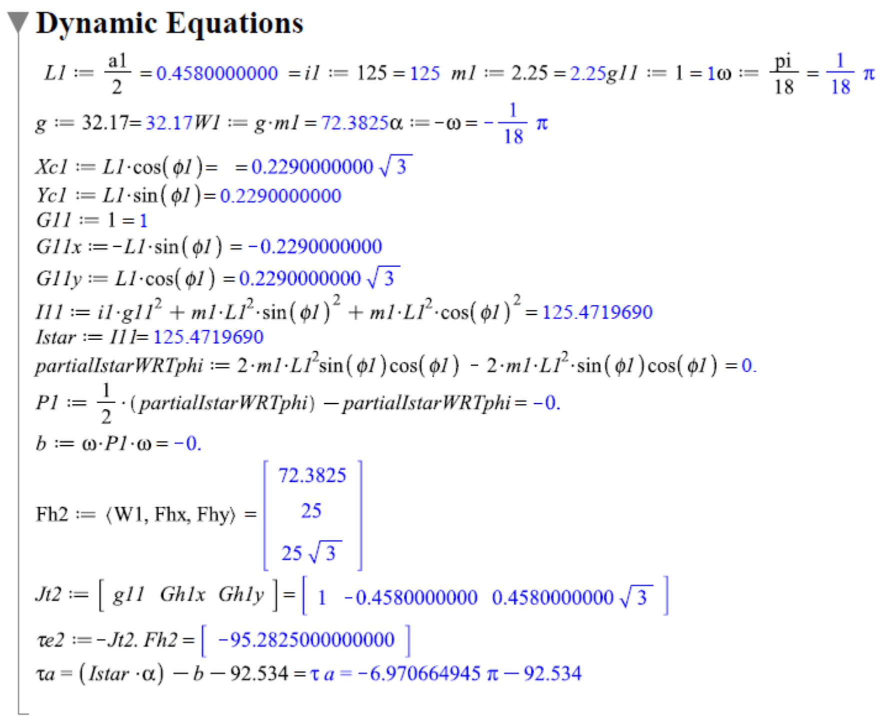


Figure 3 - Actuator Torque



Dynamic Equations

$$L1 := \frac{\mathrm{a1}}{2} = 0.4580000000 = i1 := 125 = 125 \quad m1 := 2.25 = 2.25 g11 := 1 = 1 \omega := \frac{\mathrm{pi}}{18} = \frac{1}{18}\pi$$

$$g := 32.17 = 32.17 W1 := g \cdot m1 = 72.3825 \alpha := -\omega = -\frac{1}{18}\pi$$

$$Xc1 := L1 \cdot \cos(\phi 1) = \; = 0.2290000000\sqrt{3}$$

$$Yc1 := L1 \cdot \sin(\phi 1) = 0.2290000000$$

$$G11 := 1 = 1$$

$$G11x := -L1 \cdot \sin(\phi 1) = -0.2290000000$$

$$G11y := L1 \cdot \cos(\phi 1) = 0.2290000000\sqrt{3}$$

$$I11 := i1 \cdot g11^2 + m1 \cdot L1^2 \cdot \sin(\phi 1)^2 + m1 \cdot L1^2 \cdot \cos(\phi 1)^2 = 125.4719690$$

$$Istar := I11 = 125.4719690$$

$$partialIstarWRTphi := 2 \cdot m1 \cdot L1^2 \sin(\phi 1)\cos(\phi 1) - 2 \cdot m1 \cdot L1^2 \cdot \sin(\phi 1)\cos(\phi 1) = 0.$$

$$P1 := \frac{1}{2} \cdot (partialIstarWRTphi) - partialIstarWRTphi = -0.$$

$$b := \omega \cdot P1 \cdot \omega = -0.$$

$$\mathrm{Fh2} := \langle \mathrm{W1}, \mathrm{Fhx}, \mathrm{Fhy} \rangle = \begin{bmatrix} 72.3825 \\ 25 \\ 25\sqrt{3} \end{bmatrix}$$

$$Jt2 := \begin{bmatrix} g11 & Gh1x & Gh1y \end{bmatrix} = \begin{bmatrix} 1 & -0.4580000000 & 0.4580000000\sqrt{3} \end{bmatrix}$$

$$\tau e2 := -Jt2.\,Fh2 = \begin{bmatrix} -95.2825000000000 \end{bmatrix}$$

$$\tau a = (Istar \cdot \alpha) - b - 92.534 = \tau a = -6.970664945\,\pi - 92.534$$

Figure 4 - Dynamic Equations

Additionally, as shown in figure 3, the dynamic equations of this system were also obtained. The weight of the link had a significant effect on the external torque and thus the necessary actuator torque. To properly compensate for these dynamic requirements, a smaller external load must be applied to the robot. The velocity related inertias cancel out for a one degree of freedom system and therefore the only additional component affecting the dynamic equation is the acceleration.

Major components

Qty	Component	Vendor	Part Number
1	6061 – T6 5” Aluminum U-Channel	McMaster	1630T352
1	6061 – T6 6” Aluminum U-Channel	McMaster	1630T372
2	6A1 – 4V Titanium Pivot Pin	McMaster	89145K219
2	Steel Shoulder Bolt ¼ in x 1 in	McMaster	97345A542
1	24V Linear Actuator 90lbf	Ultramotion	5-A.083-DC-24-4-RC4/3
1	DC Servo	Pittman	9434K181
2	10LBF Load Cell	Digi-Key	MSP6948-ND
2	Limit Switch	RadioShack	275-016
1	Arduino Uno Microcontroller Board	RadioShack	276-128
1	Pololu Motor Shield	Pololu	2502

Table 1 - Bill of Materials

Design Drawings

See Appendix A for machined component engineering drawings.

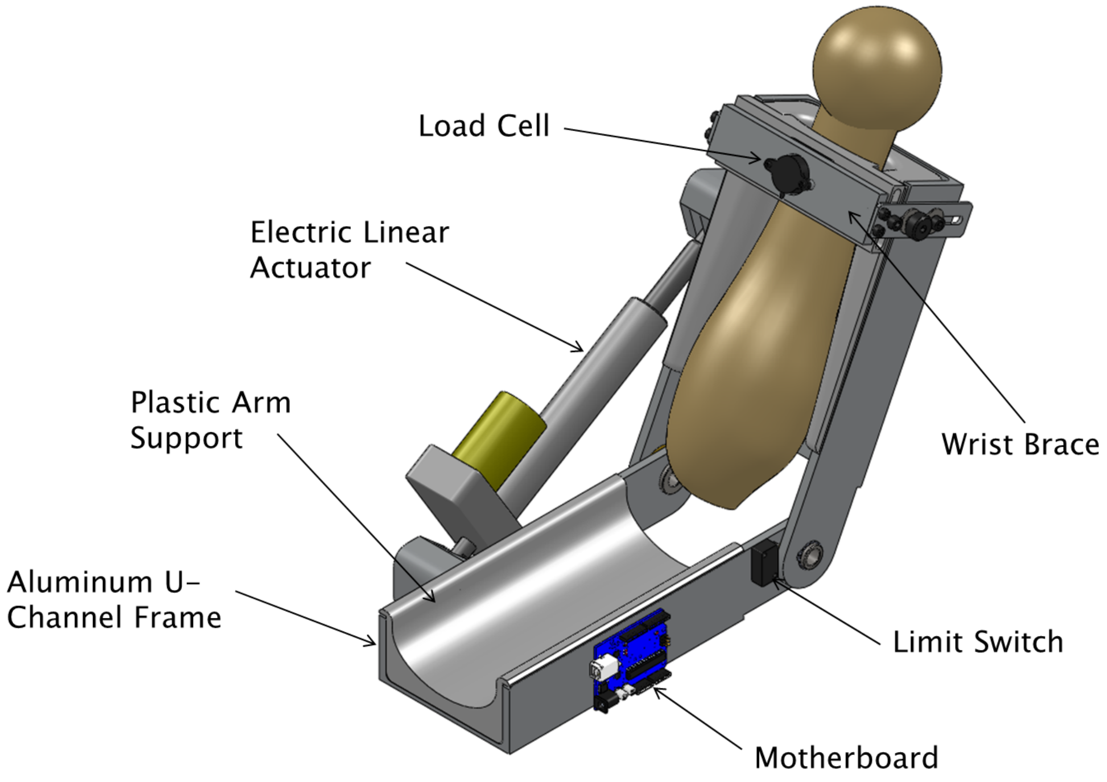


Figure 5 - Major Components

Programming

The initial phase of programming began with understanding the syntax of the wiring language that the Arduino uses. Wiring is a custom language that is derived from C++ with a few modifications for the simplification of syntax. Because the motor shield had already been decided upon as the main device to control the servos, additional libraries that were written for the Arduino compiler had to be implemented and thus new syntax had to be added and understood.

The first iteration of the program began as a simple test of all the components to have a basic understanding of how the Arduino interacted with the device, as well as the response time and calibration of the sensors that were being used to detect force applied. Several tests were conducted with the first iteration which included: sensor detection, switch detection, servo initiation, button mode switching.

After the initial trial, the code was rewritten from nearly scratch to incorporate the intended design and flow chart that was created to execute the desired function of the exoskeleton robotic arm. The program begins with the initiation of all the global variables, along with the setup of the custom library, and pin modes. Several subroutines were developed that could be called upon on every loop of the program to read the sensors and limit switches. Once the program checks the sensor and switch values, it responds to the button mode by entering different conditional loops that control which desired function the arm executes. Divided into three conditions, the first program runs the cyclic flex and relaxation motion of the arm, the second program implements a linear function that acts as an assistive ramp up mechanism when sensor input is detected, and the third condition implements a linear function that slowly activates the servo to simulate a resistive force in response to sensor input. Please refer to Appendix C for the program flow chart as well as the final code.

Circuitry

The circuit setup was relatively straight forward. With the motor shield occupying a majority of the digital pins because of the PWM capabilities, the remaining digital pins were used for the switch inputs which all had drop down resistors for functionality. The sensors were connected directly to the power source and ground, with the signal output connected directly to the analog input pins through the motor shield. Because the motor shield handled the inputs and outputs when interfaced with the Arduino, the rest was just connecting the power and motor to the shield directly. Please refer to Appendix D for the circuit chart.

Prototype Construction

The prototype consists of a two piece extruded Aluminum U-channel frame that was machined on a vertical mill and sanded down to eliminate sharp edges. Two hollow Titanium shafts were machined on a lathe and secure the two segments of the Aluminum channels with external retaining rings.

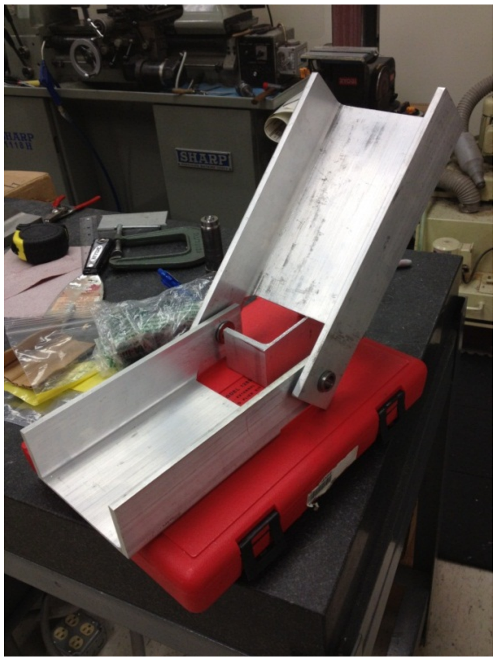


Figure 6 - Frame Assembly

Initial testing of the linear actuator resulted in the motor burning out from stalling when the piston accidentally reached its limit. This caused us to get a replacement motor of similar size and torque but with a much higher max RPM. The replacement motor operated at too high a speed, so a significant gear reduction was needed in order for the piston to move at a manageable speed. New timing belt pulleys and a longer timing belt were ordered from McMaster to replace the existing ones, changing the gear ratio from 3:2 to 6:1 before the change from rotation to linear motion. Active idler pulleys were machined out of Delrin in order to ensure that the belt wrapped around approximately 50% of the smaller driving pinion and prevent belt slippage and damage.

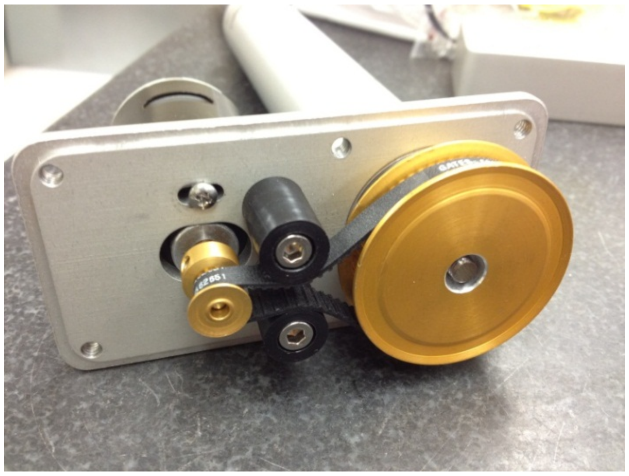


Figure 7 - Linear Actuator Drive

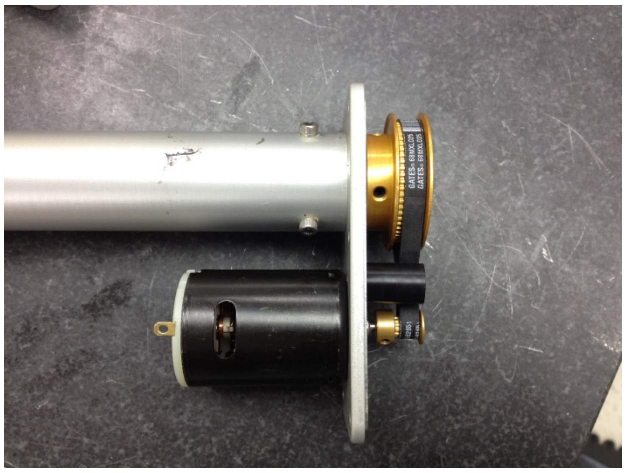


Figure 8 - Linear Actuator Drive Side

Once the linear actuator was up and running again, two solid blocks of Aluminum were machined on a CNC machine to create the joints between the linear actuator and the frame assembly. Soon after the adjusting sensor mount/wrist support was machined on a vertical mill and assembled onto the frame. At this point all the machined components were then sandblasted to provide a clean finish.

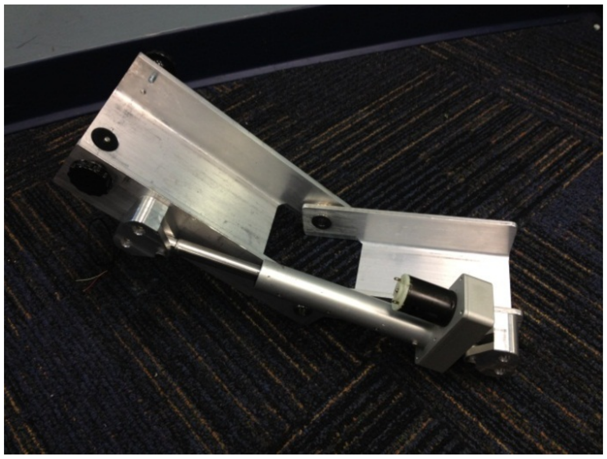


Figure 9 - Linear Actuator Mounted

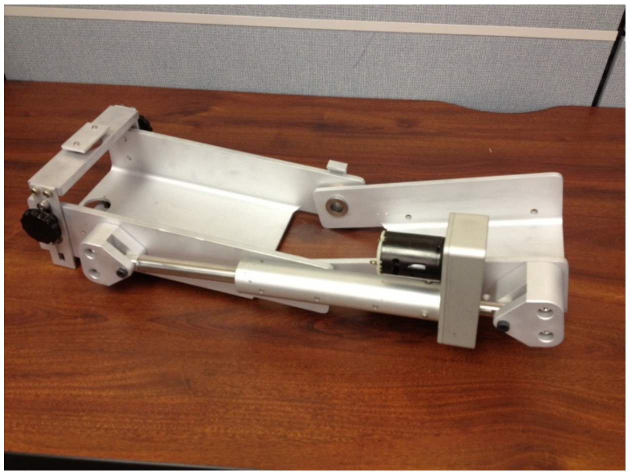


Figure 10 - Frame Assembly Sandblasted

Once the machined components were assembled, the Arduino motherboard, along with the Pololu motor driver, and hand-built circuit on a breadboard were connected to the mechanism. This allowed us to test the software and ensure proper functionality of the mechanism.

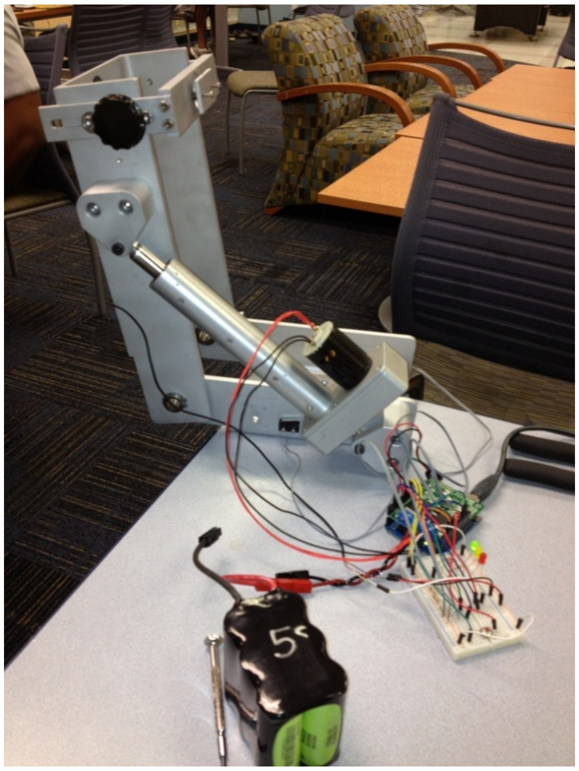


Figure 11 - Electronics Connected

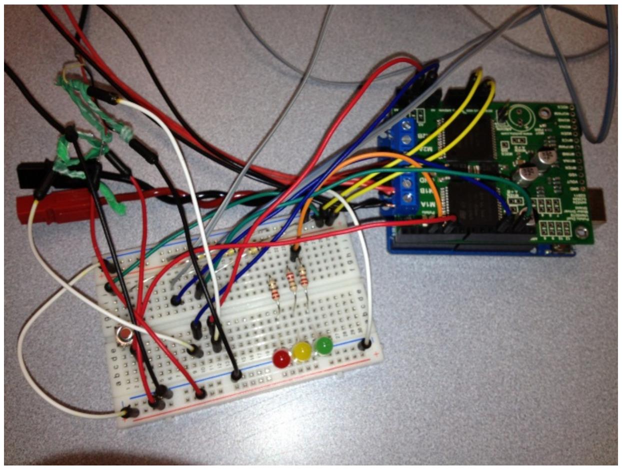


Figure 12 - Electronic Connections

Once the software was running properly, the electronic boards were mounted and all the electrical connections were soldered on and routed neatly. Enough slack was left to ensure proper routing throughout the mechanism's range of motion. Conforming arm rests that were printed with an SLS machine were then assembled to the mechanism, and Velcro straps were cut to length and placed on.

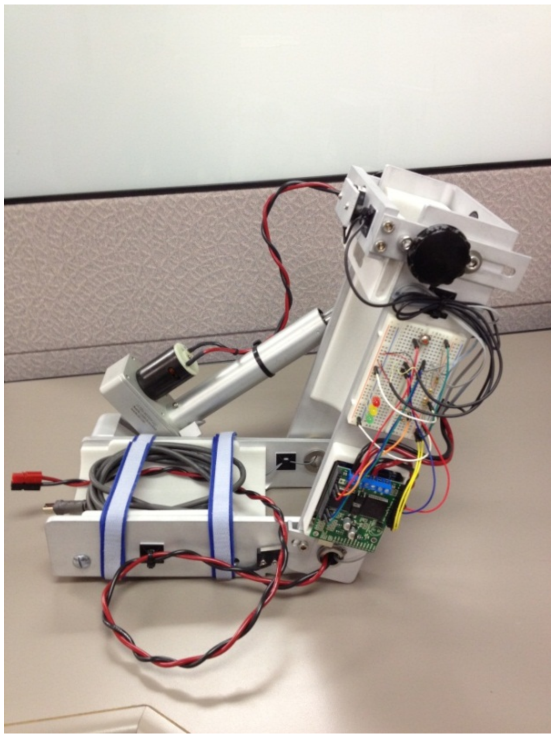


Figure 13 - Final Assembly Right Side

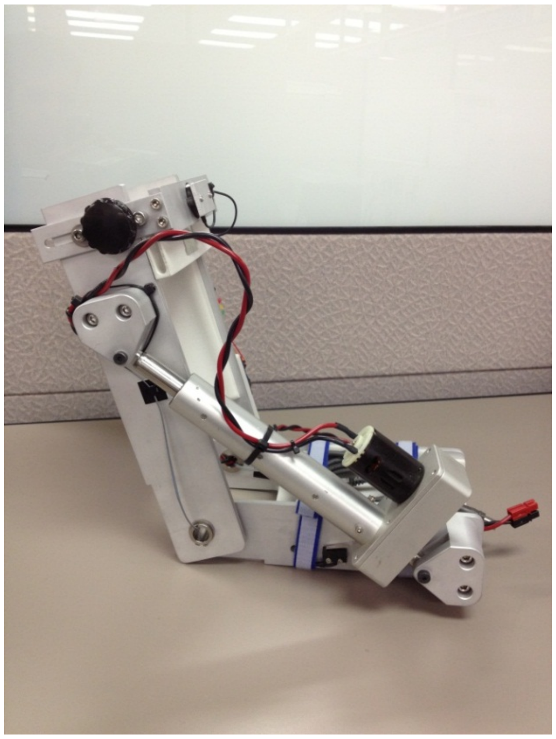


Figure 14 - Final Assembly Left Side

Testing and Test Results

Testing of the mechanism consisted of confirming that each sensor and actuator worked individually, before the assembly as a whole was tested. Both touch sensors were measured with a multimeter to ensure that the circuit was normally open when untouched, and closed when the lever depressed the buttons. The load cells were measured with a multimeter as well to ensure that there was a consistent change in voltage between the signal and ground wires when 5V was applied. The 24V motor was run independent of the linear actuator to confirm proper functionality, and then it was actuated when assembled to the linear actuator to ensure proper function of the pulley system and screw/nut mechanism.

Once all individual components were verified, the mechanism was assembled, with the exception of the linear actuator, and the early revisions of the software was run to ensure the logic was sound and proper movement of the actuator per load cell and touch sensor inputs. The final stage of testing prior to final assembly was verification of the linear actuator reaction to load cell inputs from the user – the higher the load on the cell, the faster the linear actuator moved.

Once the mechanism was fully assembled and all wires were routed, final testing was performed. A user would strap their arm into the mechanism and operate the mechanism at various speeds and intensities to ensure proper functionality, reaction time, and calibrated for a comfortable input threshold for motor reaction (start-up force and variable speed control). This testing was performed for all three states of the mechanism: passive, active-assisted, and active-constrained.

The system performed well, with a low force threshold and good sensitivity in the active-assisted mode. A balance between the linear actuator force and speed was needed in the passive state in order to allow the mechanism to begin its motion up from a fully extended position (flexing), as it is the orientation that requires the largest amount of force from the actuator to begin movement after hitting the touch sensor. The active-constrained state resists motion well in order to make the user work for small slow movements of the mechanism, and may be altered via the software to allow for users of varying levels of strength.

Overall the mechanism performed quite well, but the ergonomics of the system are not the best. Velcro straps are used to attach the mechanism to the user's arm, which provides a loose fit at best and is most noticeably lacking in the active-constrained mode. The mechanism as a whole has opportunities for weight reduction as having it strapped to your arm may cause shoulder soreness after extended periods of time. Finally, the size of the overall frame and packaging of the electronics also have the opportunity to be shrunken down for a more comfortable fit.

Conclusions and Future Work

This first generation robotic system contains the necessary features needed for the device to be therapeutically sound. Further design improvements are needed however before clinical trials can be performed to ensure patient safety and reliability of the device. The fastening system for example needs to be improved, to ensure that the device is securely attached to a patient's arm during the therapy sessions. Additionally, a soft padding material should be added to the parts that are in contact with the patient's arm.

References

[1] Vertechy, Rocco. Development of a New Endoskeleton for Upper Limb Rehabilitation. Kyoto International Conference Center. June 2009.

[2] Lum, Peter. A ROBOTIC SYSTEM FOR UPPER-LIMB EXERCISES TO PROMOTE RECOVERY OF MOTOR FUNCTION FOLLOWING STROKE. International Conference on Rehabilitation Robotics. 1999.

Appendix A

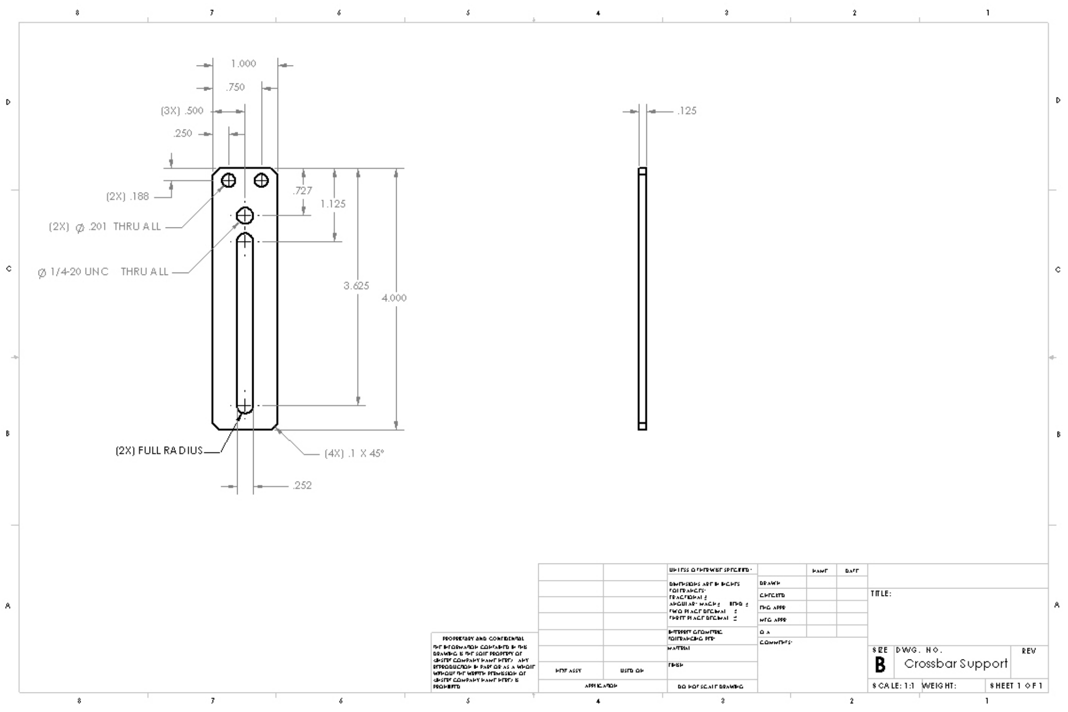


Figure 1: Crossbar Support Drawing

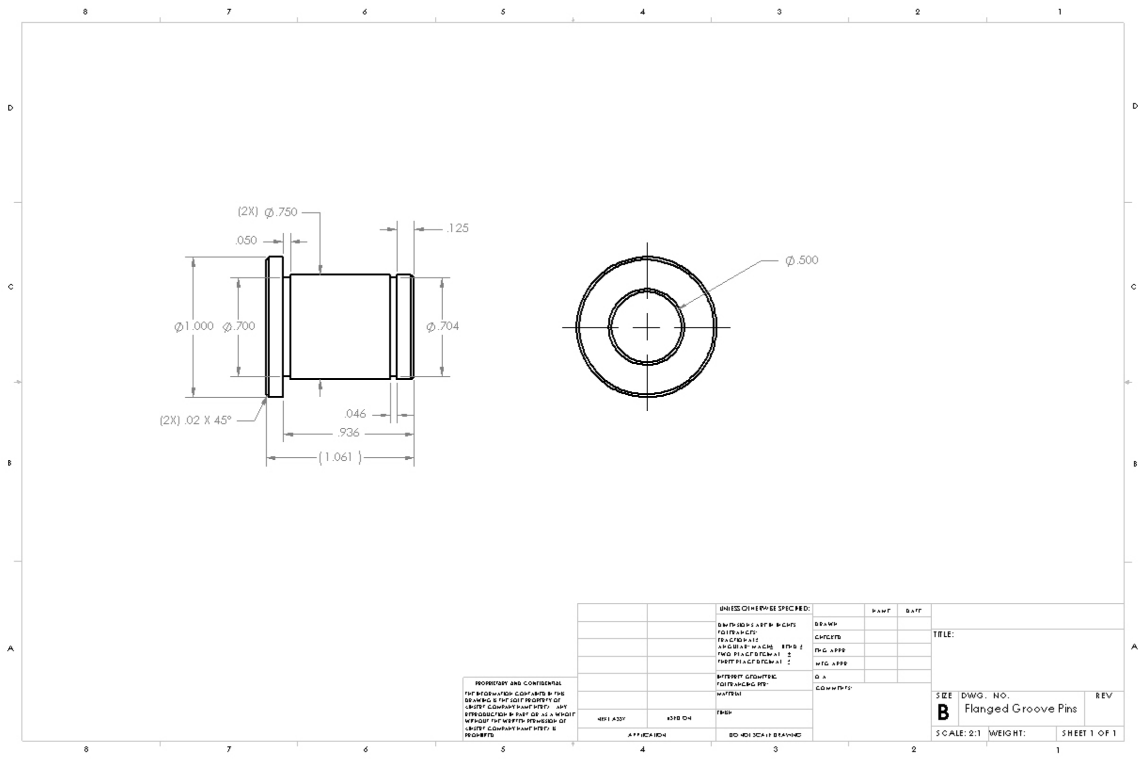


Figure 2: Flanged Grooved Pin Drawing

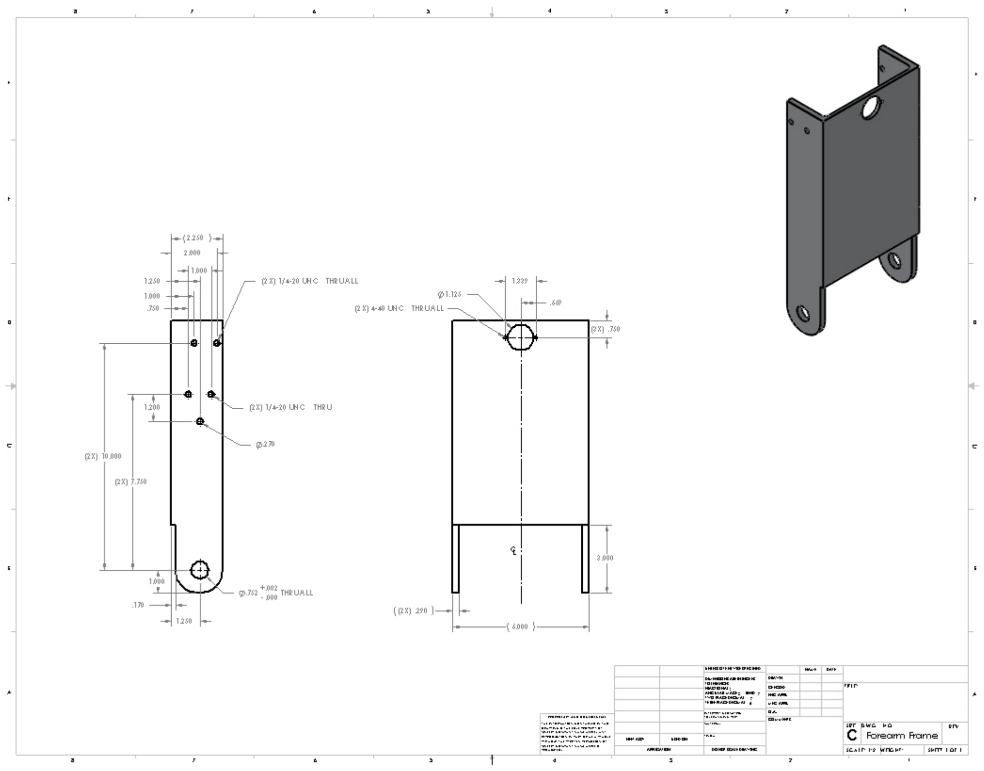


Figure 3: Forearm Frame Drawing

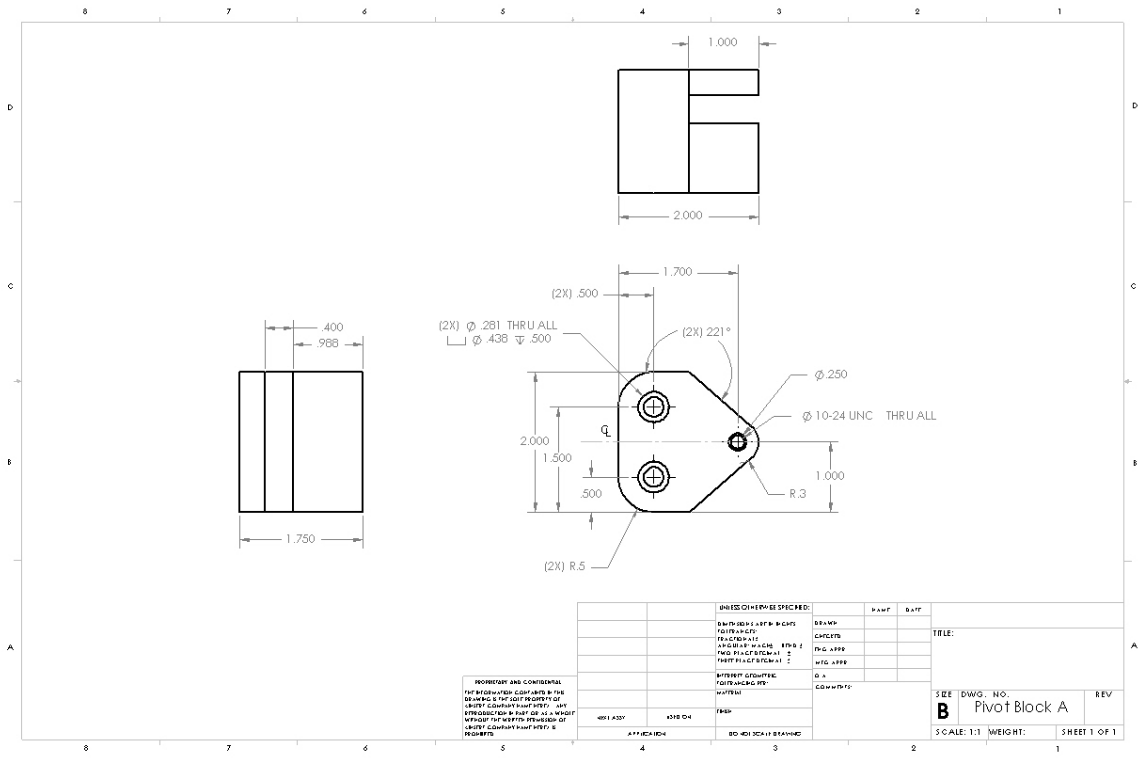


Figure 4: Pivot Block A Drawing

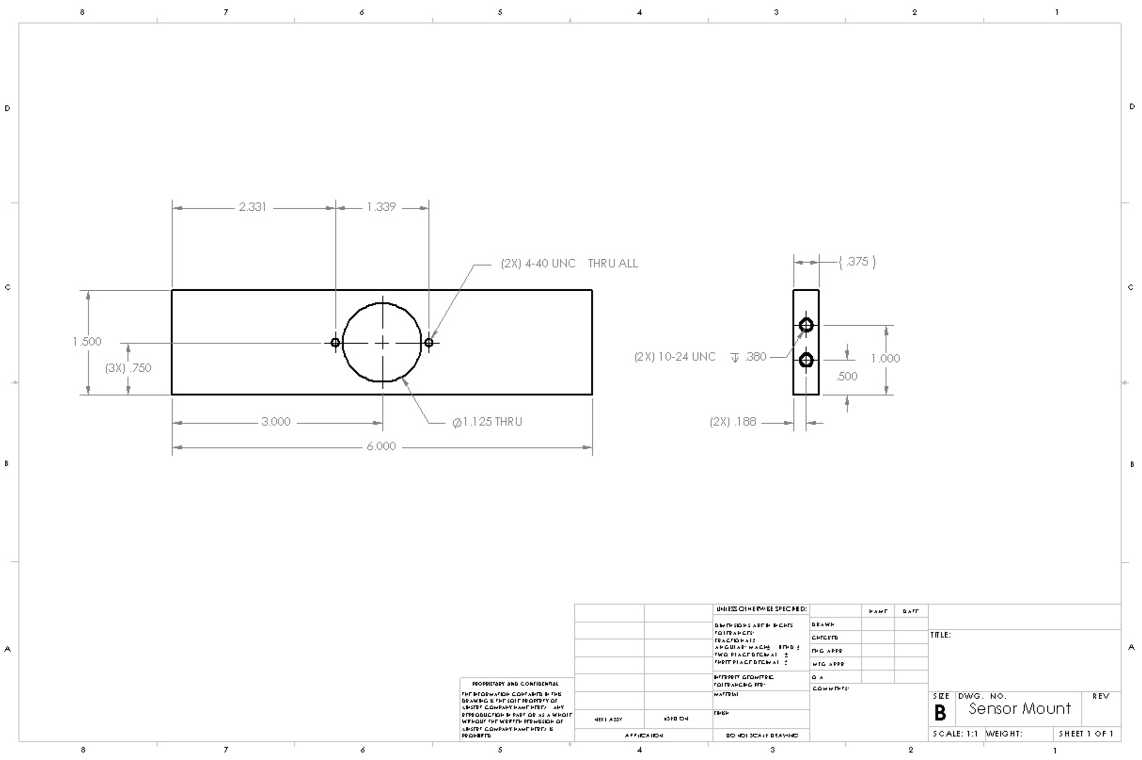


Figure 5: Sensor Mount Drawing

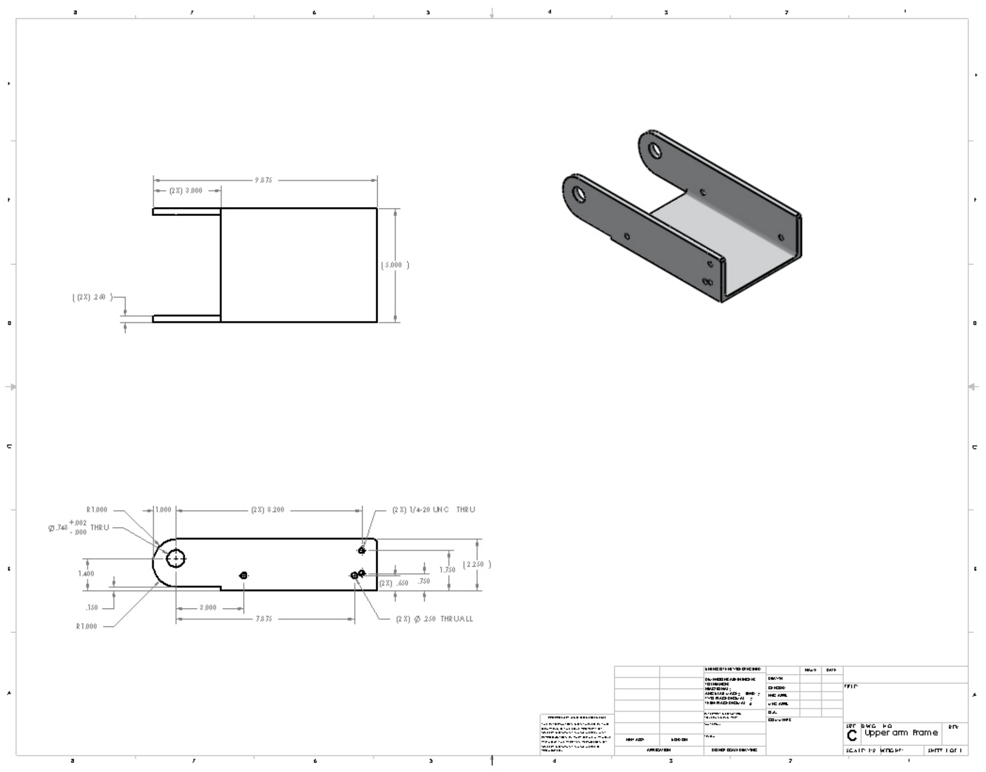


Figure 6: Upper Arm Frame Drawing

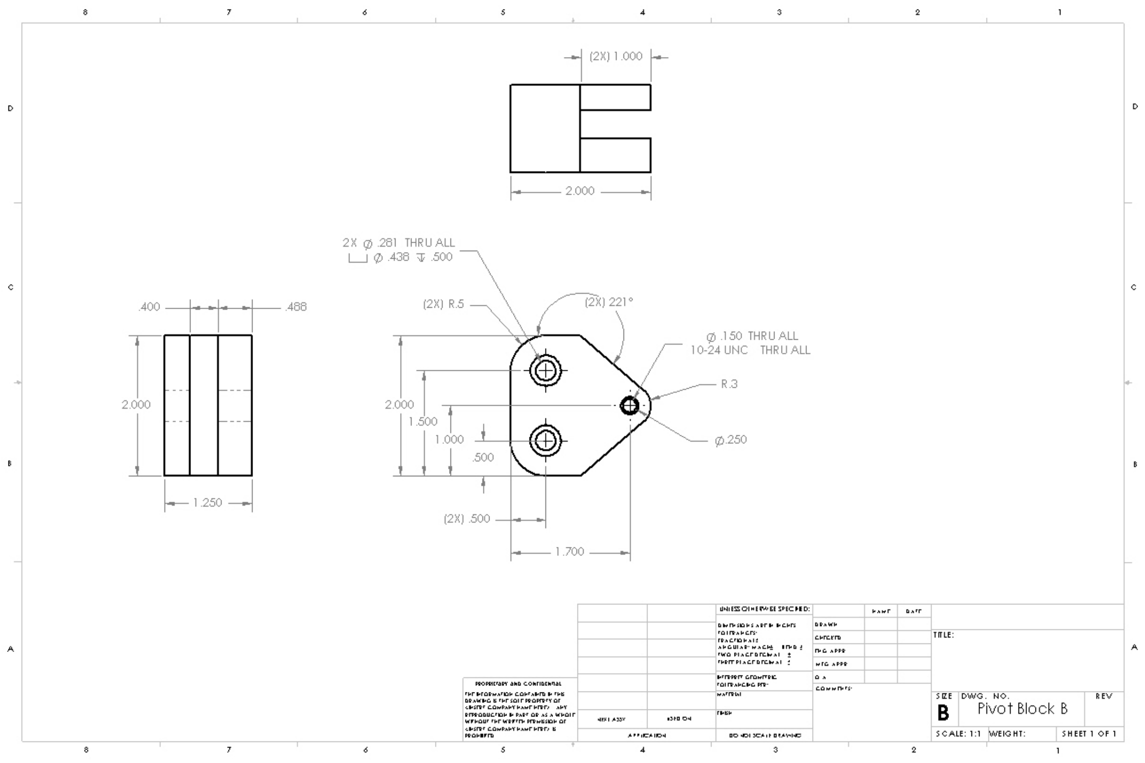
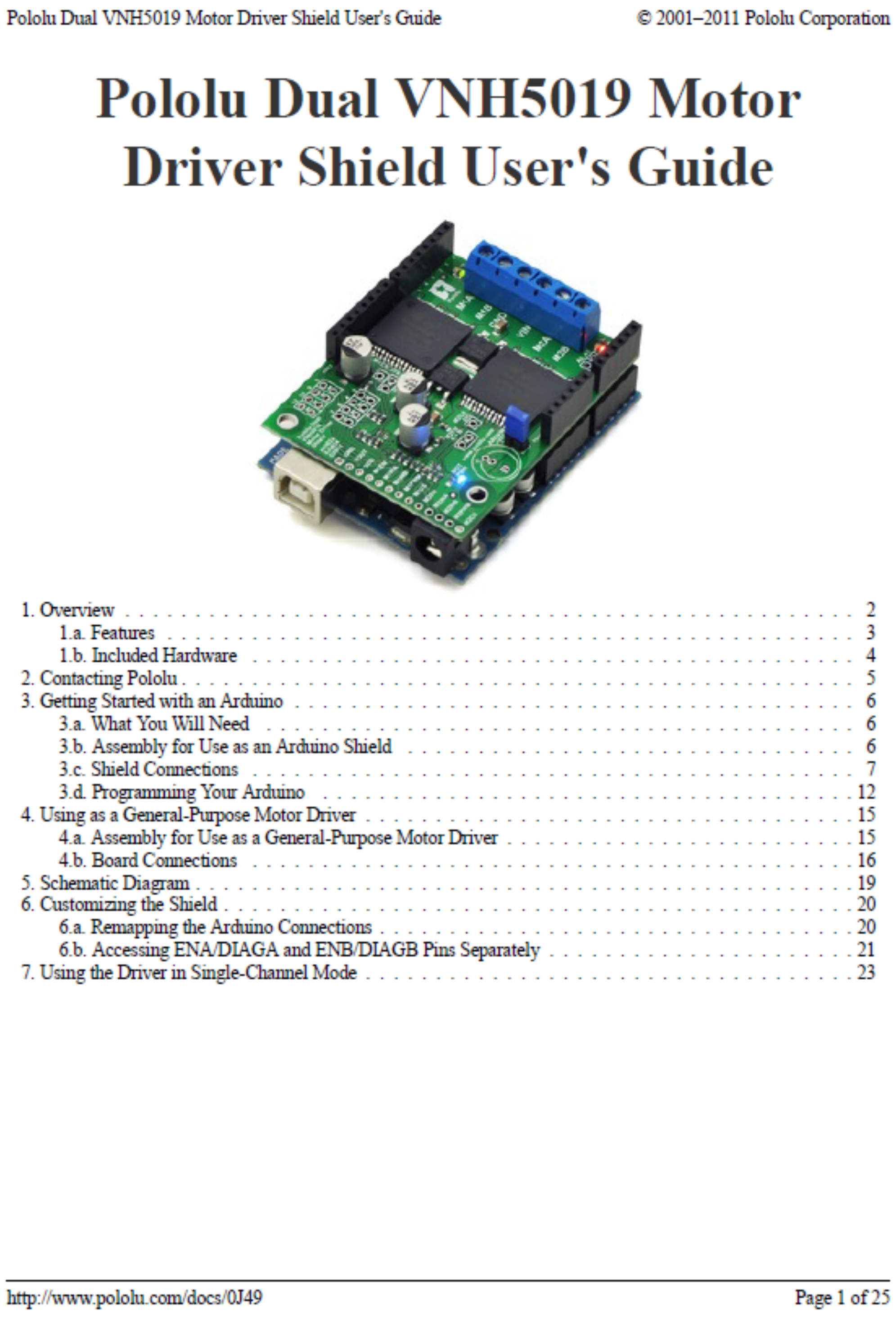


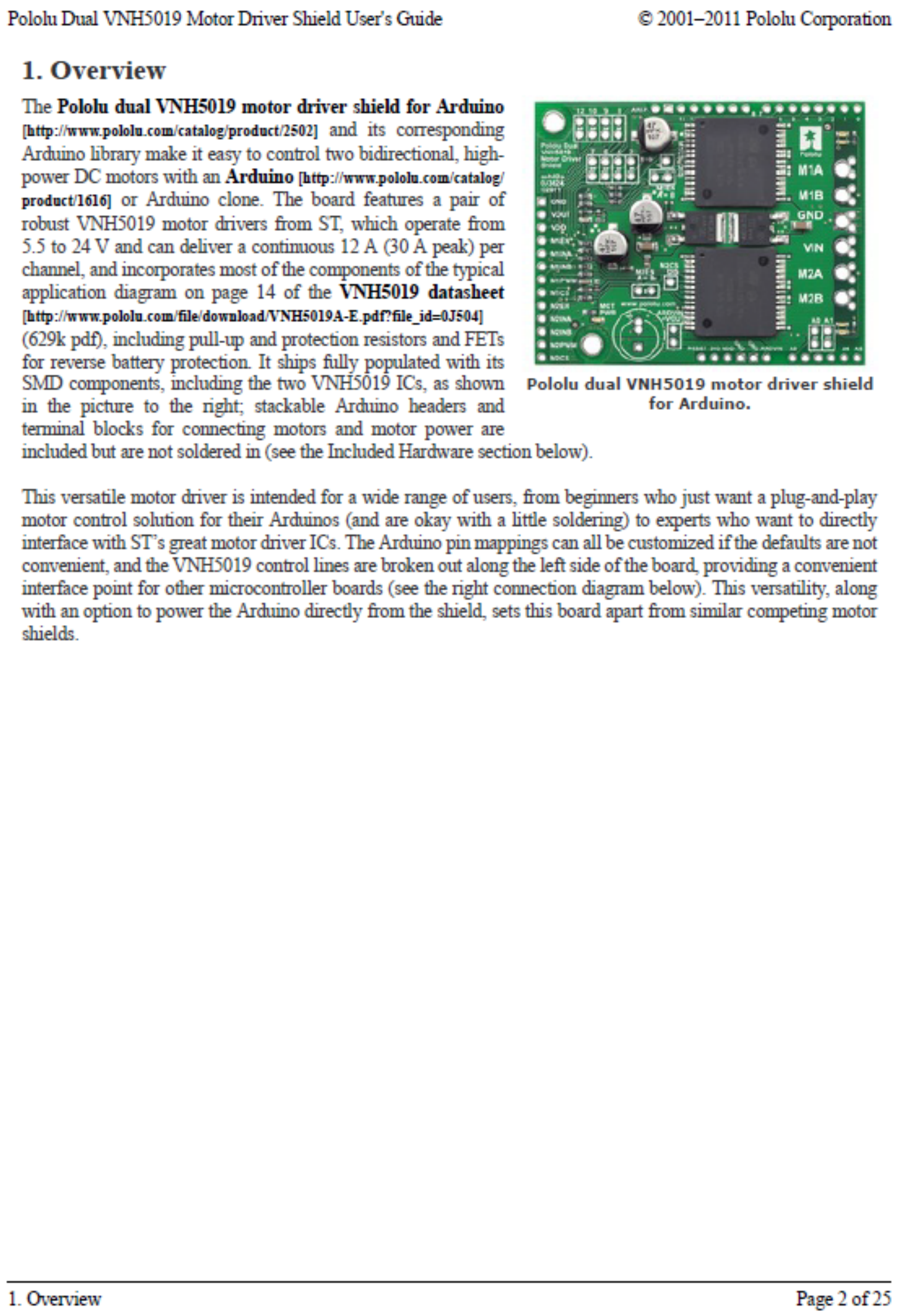
Figure 7: Pivot Block B Drawing

Appendix B

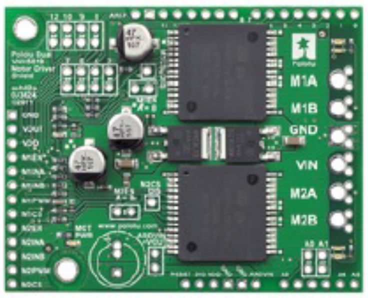


Pololu Dual VNH5019 Motor Driver Shield User's Guide

1. Overview 2
   1.a. Features 3
   1.b. Included Hardware 4
2. Contacting Pololu 5
3. Getting Started with an Arduino 6
   3.a. What You Will Need 6
   3.b. Assembly for Use as an Arduino Shield 6
   3.c. Shield Connections 7
   3.d. Programming Your Arduino 12
4. Using as a General-Purpose Motor Driver 15
   4.a. Assembly for Use as a General-Purpose Motor Driver 15
   4.b. Board Connections 16
5. Schematic Diagram 19
6. Customizing the Shield 20
   6.a. Remapping the Arduino Connections 20
   6.b. Accessing ENA/DIAGA and ENB/DIAGB Pins Separately 21
7. Using the Driver in Single-Channel Mode 23



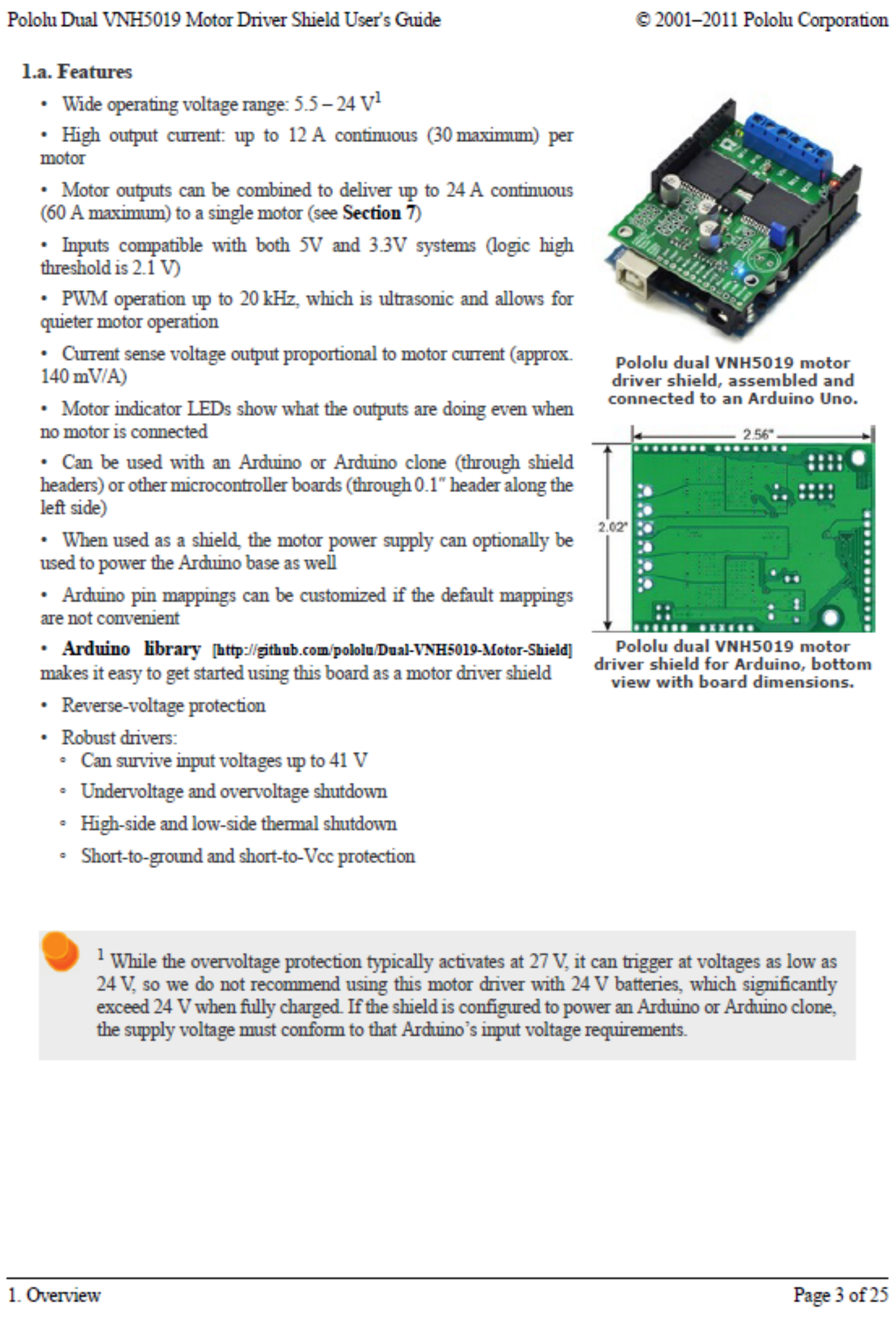
1. Overview



Pololu dual VNH5019 motor driver shield for Arduino.

The **Pololu dual VNH5019 motor driver shield for Arduino** **[http://www.pololu.com/catalog/product/2502]** and its corresponding Arduino library make it easy to control two bidirectional, high-power DC motors with an **Arduino** **[http://www.pololu.com/catalog/product/1616]** or Arduino clone. The board features a pair of robust VNH5019 motor drivers from ST, which operate from 5.5 to 24 V and can deliver a continuous 12 A (30 A peak) per channel, and incorporates most of the components of the typical application diagram on page 14 of the **VNH5019 datasheet** **[http://www.pololu.com/file/download/VNH5019A-E.pdf?file_id=0J504]** (629k pdf), including pull-up and protection resistors and FETs for reverse battery protection. It ships fully populated with its SMD components, including the two VNH5019 ICs, as shown in the picture to the right; stackable Arduino headers and terminal blocks for connecting motors and motor power are included but are not soldered in (see the Included Hardware section below).

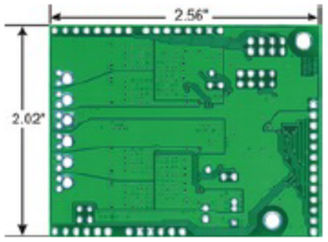
This versatile motor driver is intended for a wide range of users, from beginners who just want a plug-and-play motor control solution for their Arduinos (and are okay with a little soldering) to experts who want to directly interface with ST's great motor driver ICs. The Arduino pin mappings can all be customized if the defaults are not convenient, and the VNH5019 control lines are broken out along the left side of the board, providing a convenient interface point for other microcontroller boards (see the right connection diagram below). This versatility, along with an option to power the Arduino directly from the shield, sets this board apart from similar competing motor shields.



1.a. Features

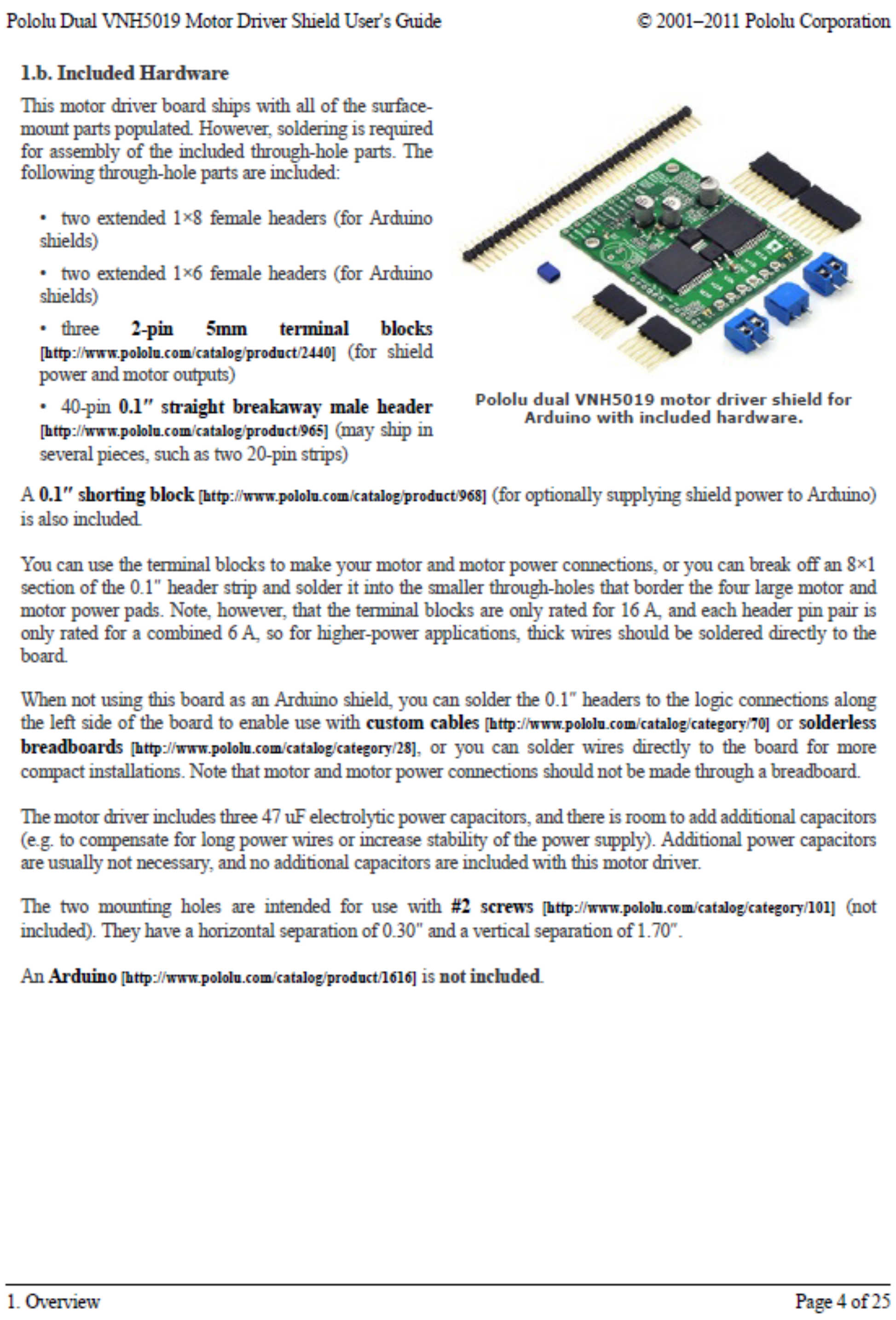
- Wide operating voltage range: 5.5 – 24 V[1]
- High output current: up to 12 A continuous (30 maximum) per motor
- Motor outputs can be combined to deliver up to 24 A continuous (60 A maximum) to a single motor (see **Section 7**)
- Inputs compatible with both 5V and 3.3V systems (logic high threshold is 2.1 V)
- PWM operation up to 20 kHz, which is ultrasonic and allows for quieter motor operation
- Current sense voltage output proportional to motor current (approx. 140 mV/A)
- Motor indicator LEDs show what the outputs are doing even when no motor is connected
- Can be used with an Arduino or Arduino clone (through shield headers) or other microcontroller boards (through 0.1″ header along the left side)
- When used as a shield, the motor power supply can optionally be used to power the Arduino base as well
- Arduino pin mappings can be customized if the default mappings are not convenient
- **Arduino library** **[http://github.com/pololu/Dual-VNH5019-Motor-Shield]** makes it easy to get started using this board as a motor driver shield
- Reverse-voltage protection
- Robust drivers:
  - Can survive input voltages up to 41 V
  - Undervoltage and overvoltage shutdown
  - High-side and low-side thermal shutdown
  - Short-to-ground and short-to-Vcc protection

Pololu dual VNH5019 motor driver shield, assembled and connected to an Arduino Uno.



Pololu dual VNH5019 motor driver shield for Arduino, bottom view with board dimensions.

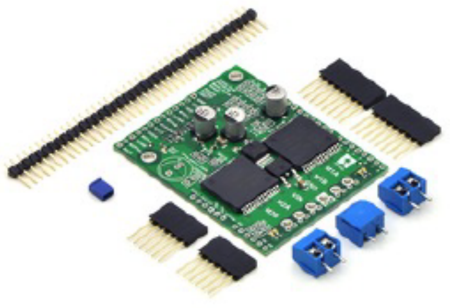
[1] While the overvoltage protection typically activates at 27 V, it can trigger at voltages as low as 24 V, so we do not recommend using this motor driver with 24 V batteries, which significantly exceed 24 V when fully charged. If the shield is configured to power an Arduino or Arduino clone, the supply voltage must conform to that Arduino's input voltage requirements.



1.b. Included Hardware

This motor driver board ships with all of the surface-mount parts populated. However, soldering is required for assembly of the included through-hole parts. The following through-hole parts are included:

- two extended 1×8 female headers (for Arduino shields)
- two extended 1×6 female headers (for Arduino shields)
- three **2-pin 5mm terminal blocks** **[http://www.pololu.com/catalog/product/2440]** (for shield power and motor outputs)
- 40-pin **0.1″ straight breakaway male header** **[http://www.pololu.com/catalog/product/965]** (may ship in several pieces, such as two 20-pin strips)



Pololu dual VNH5019 motor driver shield for Arduino with included hardware.

A **0.1″ shorting block** **[http://www.pololu.com/catalog/product/968]** (for optionally supplying shield power to Arduino) is also included.

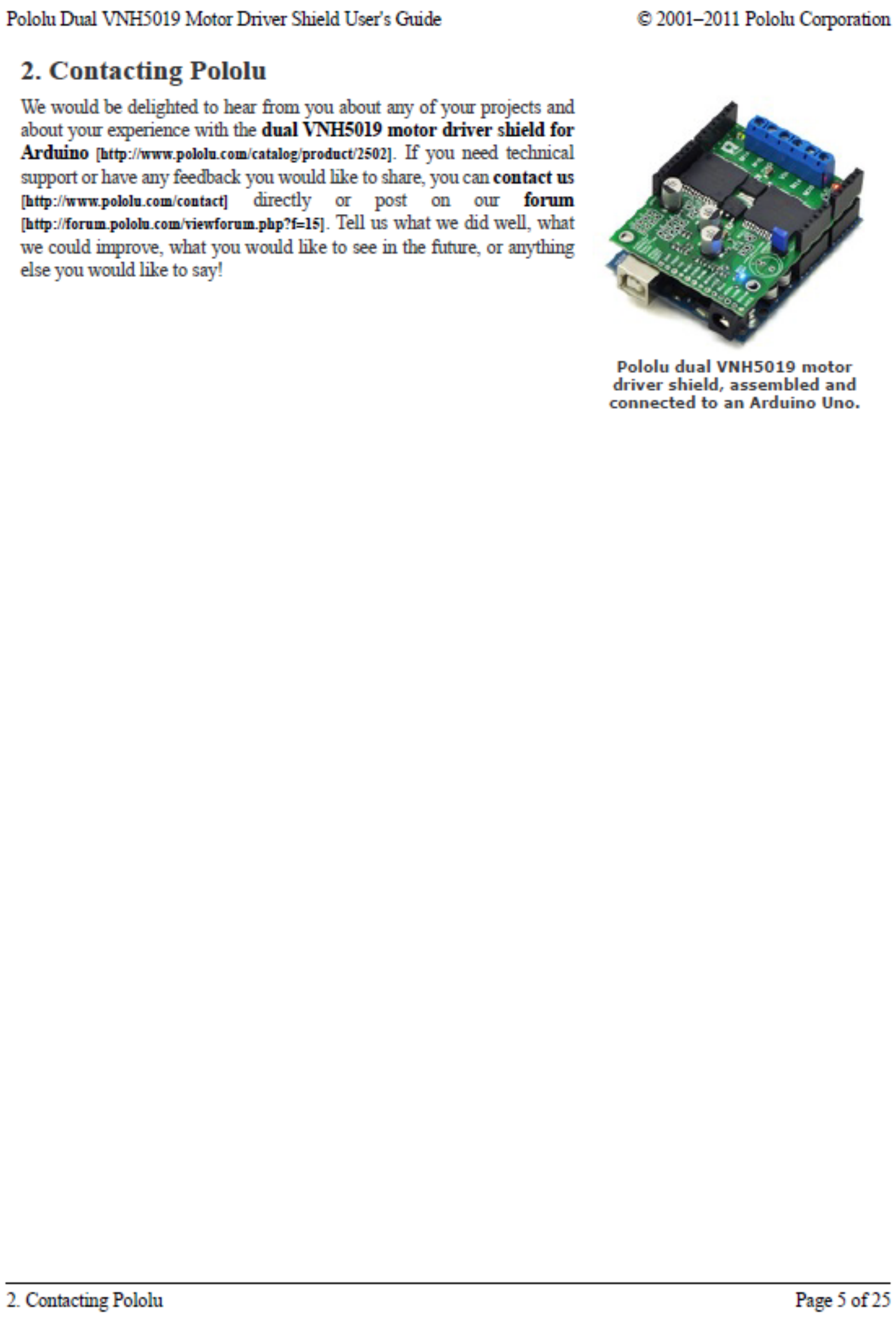
You can use the terminal blocks to make your motor and motor power connections, or you can break off an 8×1 section of the 0.1" header strip and solder it into the smaller through-holes that border the four large motor and motor power pads. Note, however, that the terminal blocks are only rated for 16 A, and each header pin pair is only rated for a combined 6 A, so for higher-power applications, thick wires should be soldered directly to the board.

When not using this board as an Arduino shield, you can solder the 0.1" headers to the logic connections along the left side of the board to enable use with **custom cables** **[http://www.pololu.com/catalog/category/70]** or **solderless breadboards** **[http://www.pololu.com/catalog/category/28]**, or you can solder wires directly to the board for more compact installations. Note that motor and motor power connections should not be made through a breadboard.

The motor driver includes three 47 uF electrolytic power capacitors, and there is room to add additional capacitors (e.g. to compensate for long power wires or increase stability of the power supply). Additional power capacitors are usually not necessary, and no additional capacitors are included with this motor driver.

The two mounting holes are intended for use with **#2 screws** **[http://www.pololu.com/catalog/category/101]** (not included). They have a horizontal separation of 0.30" and a vertical separation of 1.70".

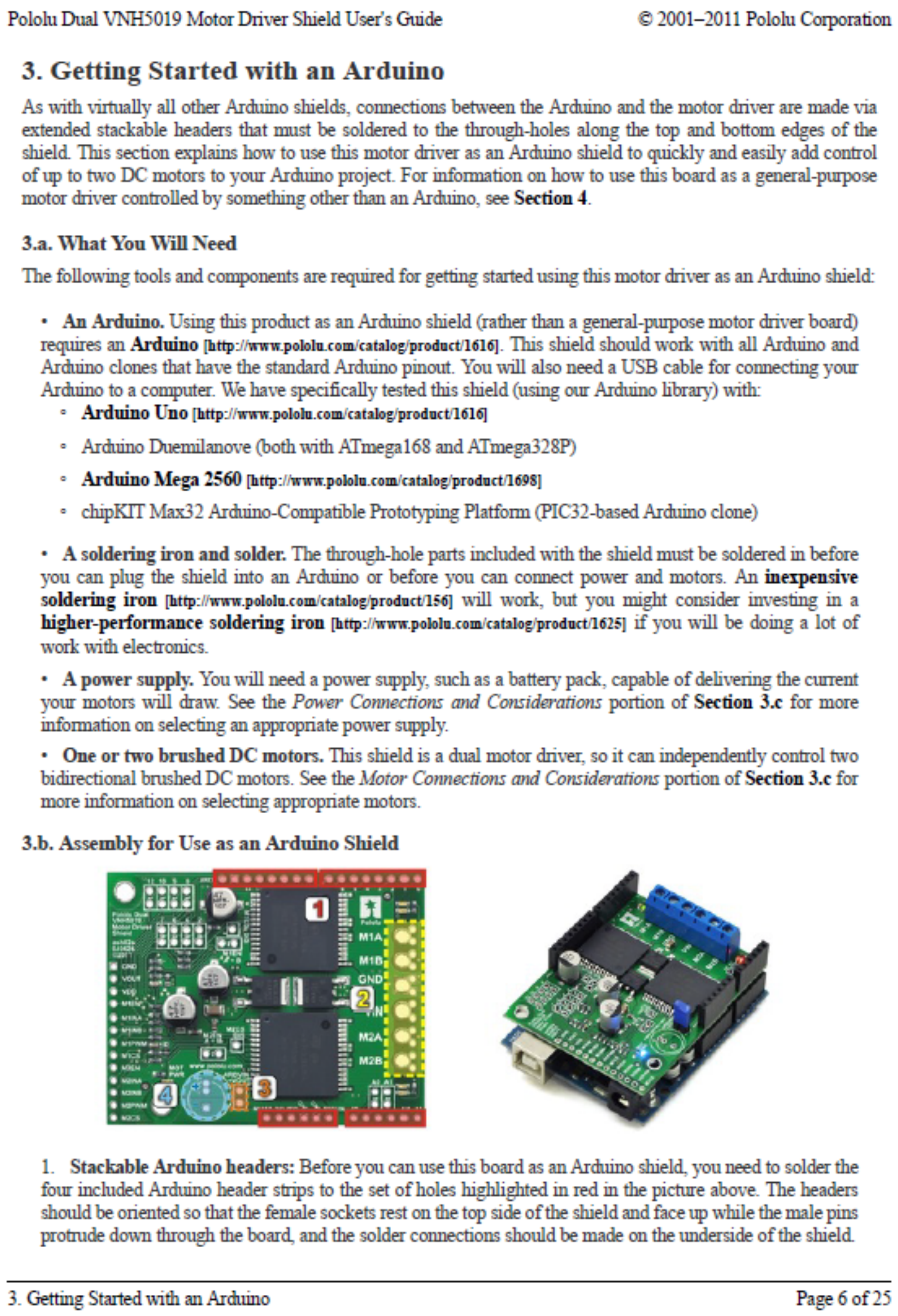
An **Arduino** **[http://www.pololu.com/catalog/product/1616]** is **not included**.



2. Contacting Pololu

We would be delighted to hear from you about any of your projects and about your experience with the **dual VNH5019 motor driver shield for Arduino** **[http://www.pololu.com/catalog/product/2502]**. If you need technical support or have any feedback you would like to share, you can **contact us** **[http://www.pololu.com/contact]** directly or post on our **forum** **[http://forum.pololu.com/viewforum.php?f=15]**. Tell us what we did well, what we could improve, what you would like to see in the future, or anything else you would like to say!

Pololu dual VNH5019 motor driver shield, assembled and connected to an Arduino Uno.



3. Getting Started with an Arduino

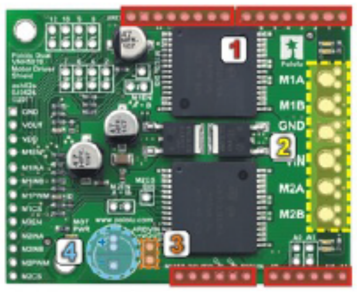
As with virtually all other Arduino shields, connections between the Arduino and the motor driver are made via extended stackable headers that must be soldered to the through-holes along the top and bottom edges of the shield. This section explains how to use this motor driver as an Arduino shield to quickly and easily add control of up to two DC motors to your Arduino project. For information on how to use this board as a general-purpose motor driver controlled by something other than an Arduino, see **Section 4**.

3.a. What You Will Need

The following tools and components are required for getting started using this motor driver as an Arduino shield:

- **An Arduino.** Using this product as an Arduino shield (rather than a general-purpose motor driver board) requires an **Arduino [http://www.pololu.com/catalog/product/1616]**. This shield should work with all Arduino and Arduino clones that have the standard Arduino pinout. You will also need a USB cable for connecting your Arduino to a computer. We have specifically tested this shield (using our Arduino library) with:
  - **Arduino Uno [http://www.pololu.com/catalog/product/1616]**
  - Arduino Duemilanove (both with ATmega168 and ATmega328P)
  - **Arduino Mega 2560 [http://www.pololu.com/catalog/product/1698]**
  - chipKIT Max32 Arduino-Compatible Prototyping Platform (PIC32-based Arduino clone)
- **A soldering iron and solder.** The through-hole parts included with the shield must be soldered in before you can plug the shield into an Arduino or before you can connect power and motors. An **inexpensive soldering iron [http://www.pololu.com/catalog/product/156]** will work, but you might consider investing in a **higher-performance soldering iron [http://www.pololu.com/catalog/product/1625]** if you will be doing a lot of work with electronics.
- **A power supply.** You will need a power supply, such as a battery pack, capable of delivering the current your motors will draw. See the *Power Connections and Considerations* portion of **Section 3.c** for more information on selecting an appropriate power supply.
- **One or two brushed DC motors.** This shield is a dual motor driver, so it can independently control two bidirectional brushed DC motors. See the *Motor Connections and Considerations* portion of **Section 3.c** for more information on selecting appropriate motors.

3.b. Assembly for Use as an Arduino Shield



1. **Stackable Arduino headers:** Before you can use this board as an Arduino shield, you need to solder the four included Arduino header strips to the set of holes highlighted in red in the picture above. The headers should be oriented so that the female sockets rest on the top side of the shield and face up while the male pins protrude down through the board, and the solder connections should be made on the underside of the shield.

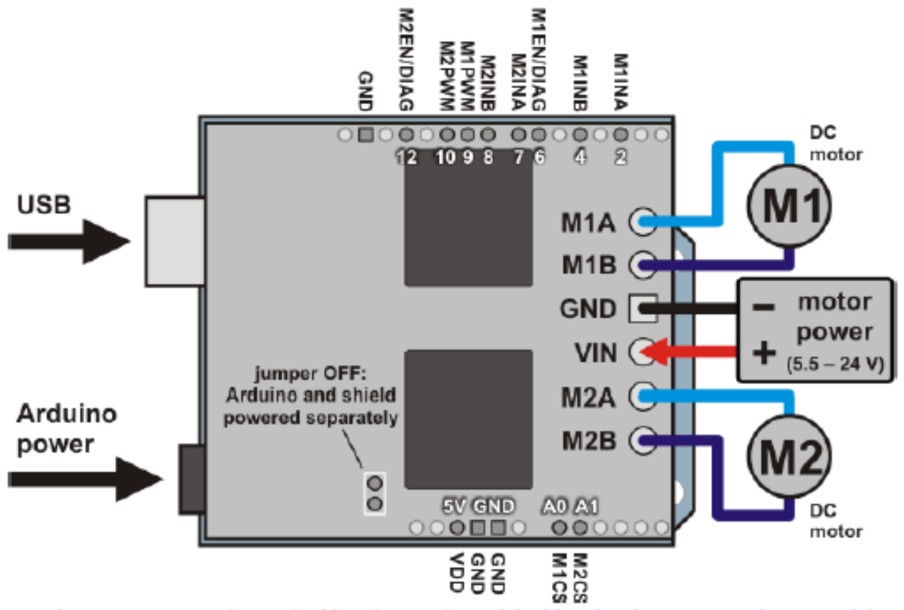
2. **Motor and power connections:** The six large holes/twelve small holes on the right side of the board, highlighted in yellow in the above diagram, are the motor outputs and power inputs. You can optionally solder the included 5mm-pitch terminal blocks to the board to enable temporary motor and motor power connections, or you can break off an 12×1 section of the included 0.1" header strip and solder it into the smaller through-holes that border the six large motor and motor power pads. Note, however, that the terminal blocks are only rated for 16 A, and each header pin pair is only rated for a combined 6 A, so for higher-current applications, thick wires with **high-current connectors** [http://www.pololu.com/catalog/product/925] should be soldered directly to the board.

3. **Arduino power jumper:** If you want the option of powering your Arduino and motor shield from the same source, you can solder a 2×1 piece of the included 0.1" male header strip to the pins highlighted in orange in the above picture. Shorting across these pins with the included shorting block will connect the shield power to the Arduino's VIN pin. You should **not** use this to power the shield from the Arduino as this connection is not designed to handle high currents, and you should never supply power to the Arduino's VIN pin or power jack while this shorting block is in place, because it will create as short between the shield power supply and the Arduino power supply.

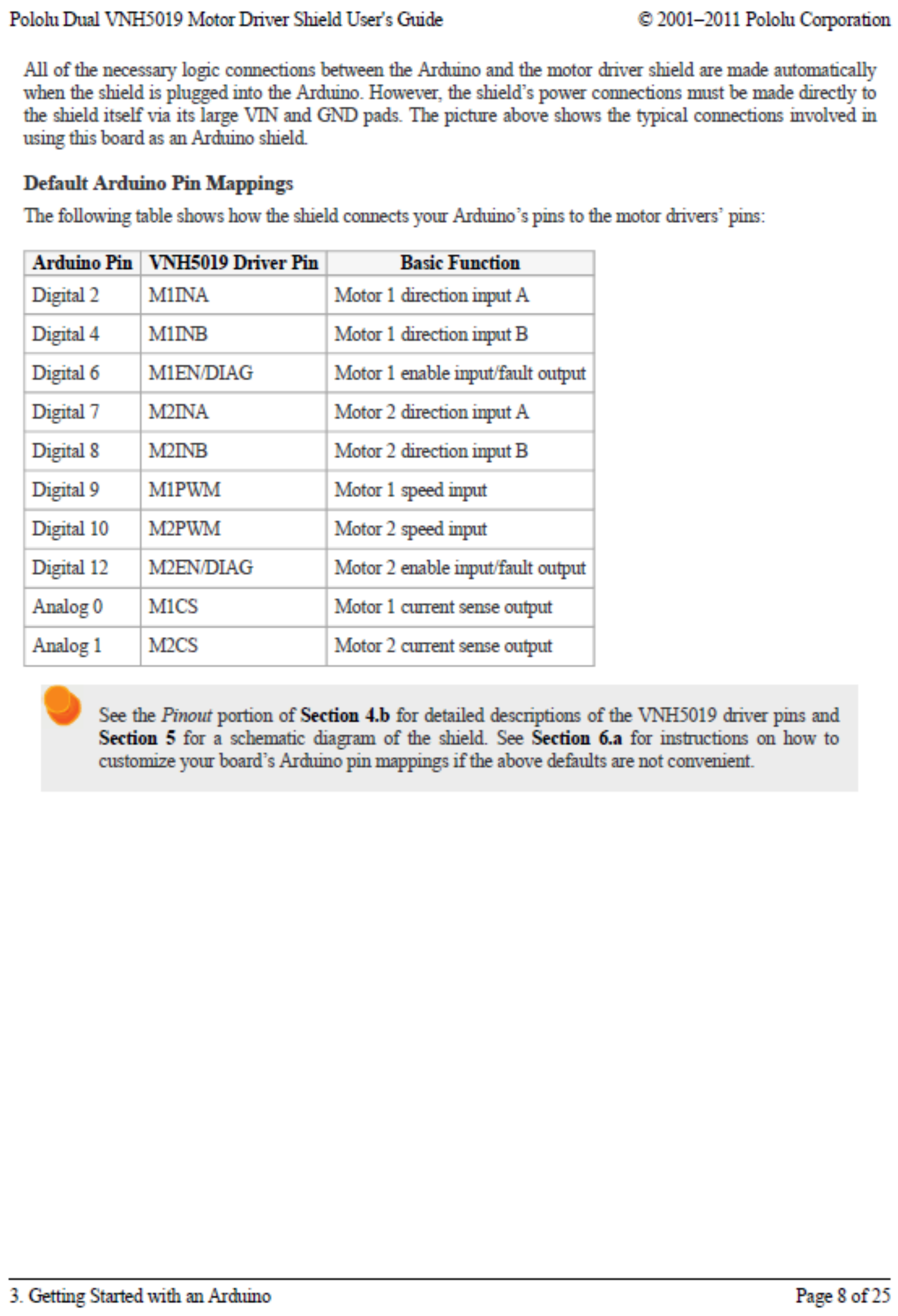
4. **Additional power capacitor:** The motor driver shield includes three pre-installed 47 uF electrolytic power capacitors, and there is space—highlighted in blue in the above picture—to add an additional capacitor (e.g. to compensate for long power wires or increase stability of the power supply). An additional power capacitor is usually not necessary, and no additional capacitors are included with this shield.

The other through-holes on the shield are used for more advanced things like customizing the Arduino pin mappings and are not necessary for getting started using this shield with an Arduino. They are discussed in more detail later in this guide.

3.c. Shield Connections



Dual VNH5019 motor driver shield with an Arduino (shield and Arduino powered separately).



Pololu Dual VNH5019 Motor Driver Shield User's Guide © 2001–2011 Pololu Corporation

All of the necessary logic connections between the Arduino and the motor driver shield are made automatically when the shield is plugged into the Arduino. However, the shield's power connections must be made directly to the shield itself via its large VIN and GND pads. The picture above shows the typical connections involved in using this board as an Arduino shield.

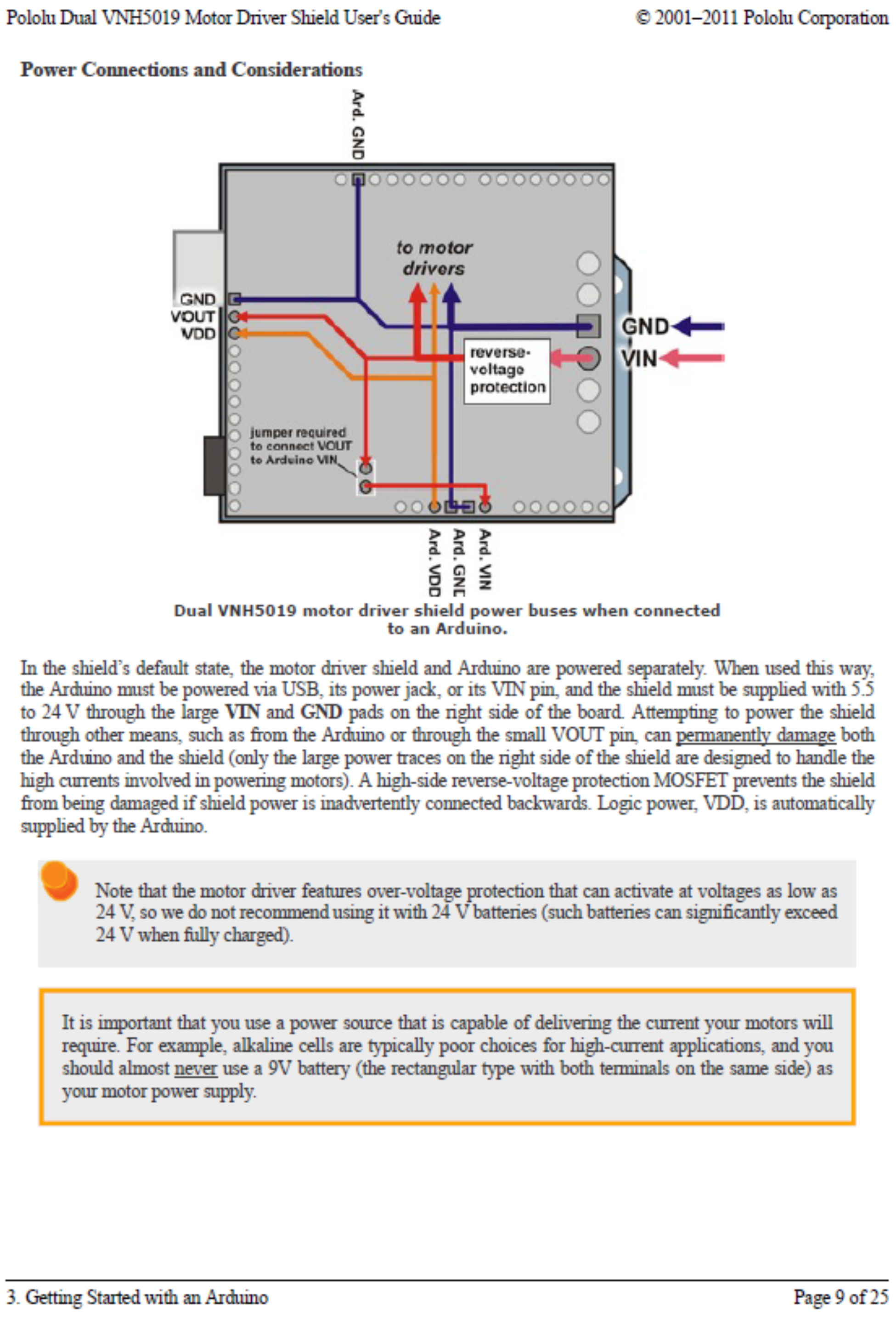
Default Arduino Pin Mappings

The following table shows how the shield connects your Arduino's pins to the motor drivers' pins:

Arduino Pin	VNH5019 Driver Pin	Basic Function
Digital 2	M1INA	Motor 1 direction input A
Digital 4	M1INB	Motor 1 direction input B
Digital 6	M1EN/DIAG	Motor 1 enable input/fault output
Digital 7	M2INA	Motor 2 direction input A
Digital 8	M2INB	Motor 2 direction input B
Digital 9	M1PWM	Motor 1 speed input
Digital 10	M2PWM	Motor 2 speed input
Digital 12	M2EN/DIAG	Motor 2 enable input/fault output
Analog 0	M1CS	Motor 1 current sense output
Analog 1	M2CS	Motor 2 current sense output

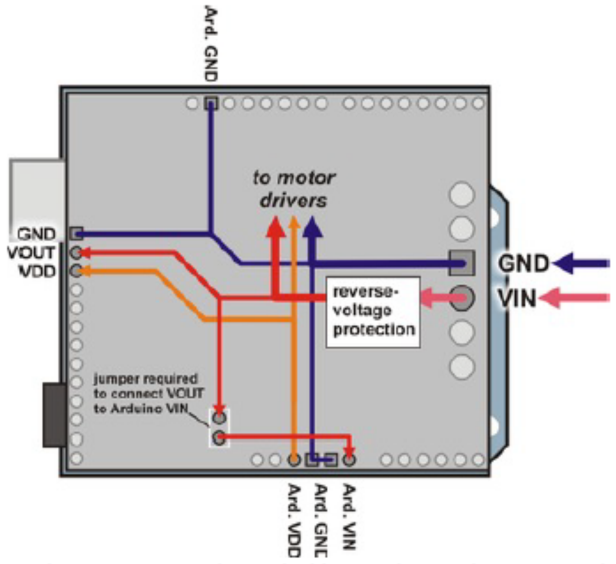
See the *Pinout* portion of **Section 4.b** for detailed descriptions of the VNH5019 driver pins and **Section 5** for a schematic diagram of the shield. See **Section 6.a** for instructions on how to customize your board's Arduino pin mappings if the above defaults are not convenient.

3. Getting Started with an Arduino Page 8 of 25



Pololu Dual VNH5019 Motor Driver Shield User's Guide © 2001–2011 Pololu Corporation

Power Connections and Considerations



Ard. GND
to motor drivers
GND
VOUT
VDD
reverse-voltage protection
jumper required to connect VOUT to Arduino VIN
GND
VIN
Ard. VDD
Ard. GND
Ard. VIN

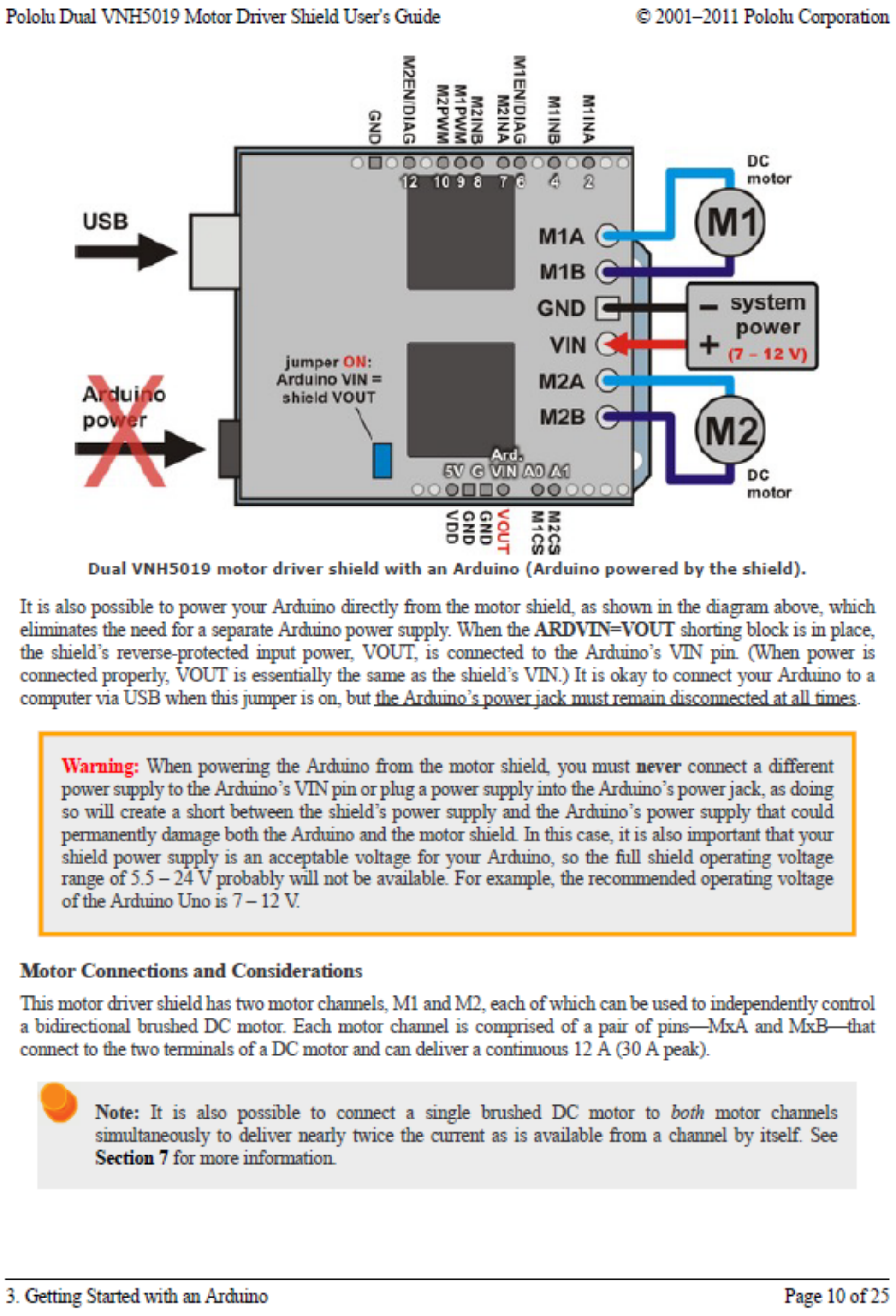
Dual VNH5019 motor driver shield power buses when connected to an Arduino.

In the shield's default state, the motor driver shield and Arduino are powered separately. When used this way, the Arduino must be powered via USB, its power jack, or its VIN pin, and the shield must be supplied with 5.5 to 24 V through the large **VIN** and **GND** pads on the right side of the board. Attempting to power the shield through other means, such as from the Arduino or through the small VOUT pin, can permanently damage both the Arduino and the shield (only the large power traces on the right side of the shield are designed to handle the high currents involved in powering motors). A high-side reverse-voltage protection MOSFET prevents the shield from being damaged if shield power is inadvertently connected backwards. Logic power, VDD, is automatically supplied by the Arduino.

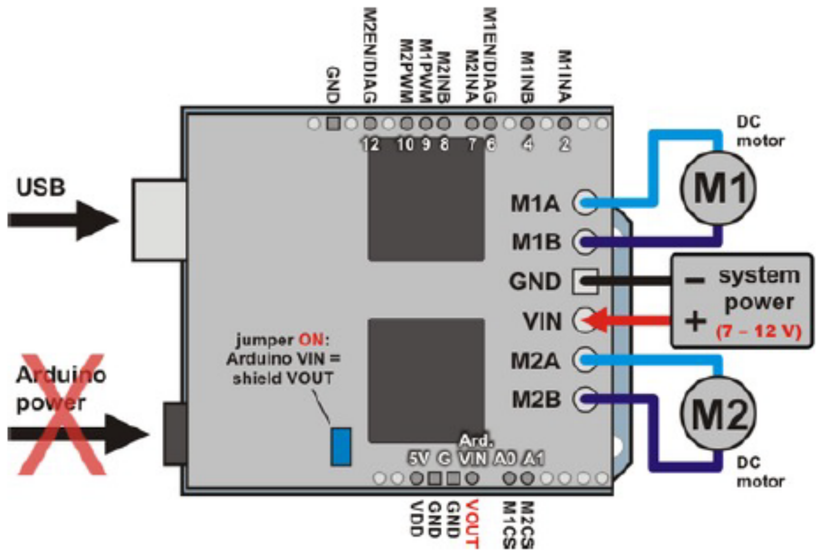
Note that the motor driver features over-voltage protection that can activate at voltages as low as 24 V, so we do not recommend using it with 24 V batteries (such batteries can significantly exceed 24 V when fully charged).

It is important that you use a power source that is capable of delivering the current your motors will require. For example, alkaline cells are typically poor choices for high-current applications, and you should almost never use a 9V battery (the rectangular type with both terminals on the same side) as your motor power supply.

3. Getting Started with an Arduino Page 9 of 25



Pololu Dual VNH5019 Motor Driver Shield User's Guide © 2001–2011 Pololu Corporation



Dual VNH5019 motor driver shield with an Arduino (Arduino powered by the shield).

It is also possible to power your Arduino directly from the motor shield, as shown in the diagram above, which eliminates the need for a separate Arduino power supply. When the **ARDVIN=VOUT** shorting block is in place, the shield's reverse-protected input power, VOUT, is connected to the Arduino's VIN pin. (When power is connected properly, VOUT is essentially the same as the shield's VIN.) It is okay to connect your Arduino to a computer via USB when this jumper is on, but the Arduino's power jack must remain disconnected at all times.

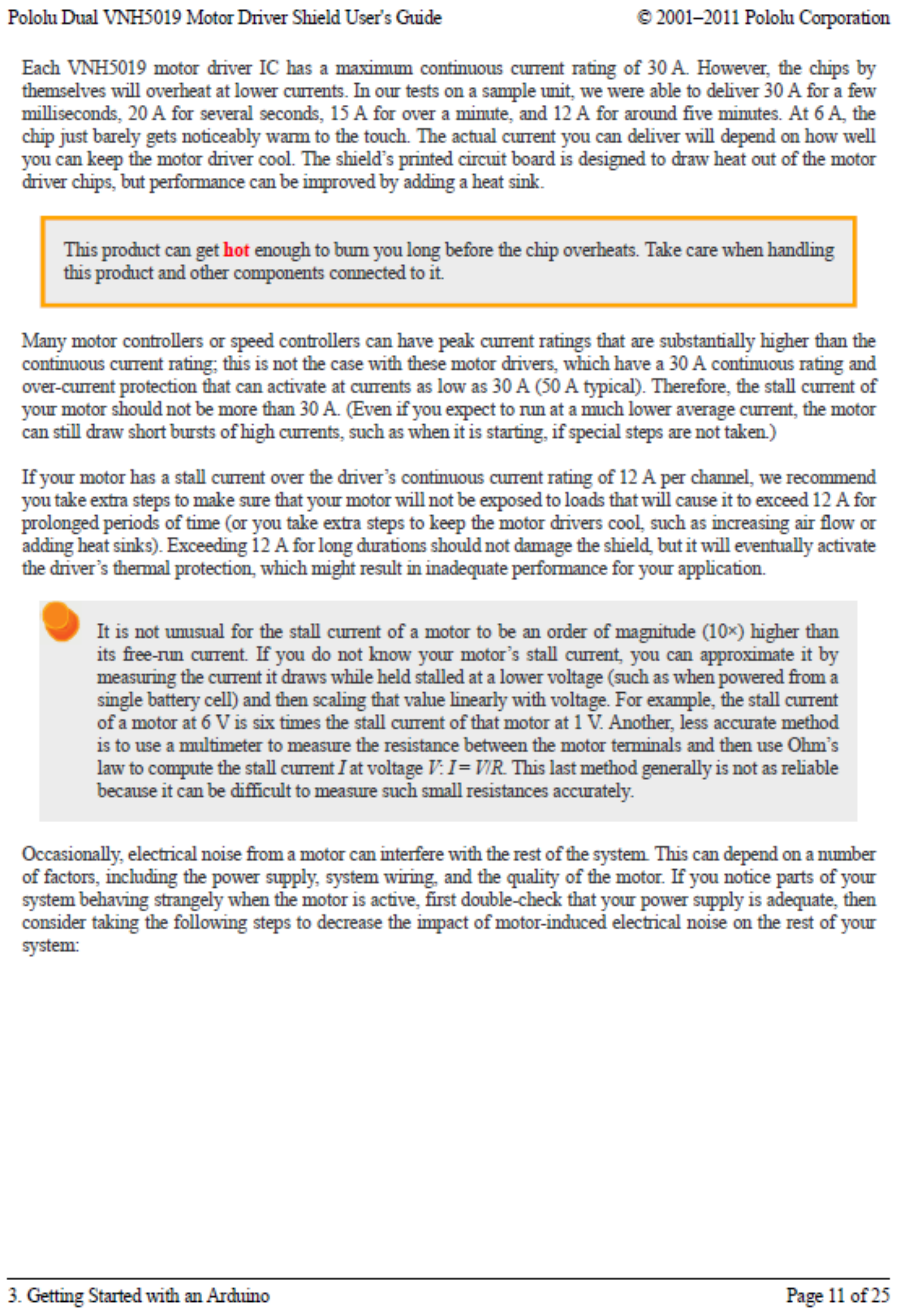
Warning: When powering the Arduino from the motor shield, you must **never** connect a different power supply to the Arduino's VIN pin or plug a power supply into the Arduino's power jack, as doing so will create a short between the shield's power supply and the Arduino's power supply that could permanently damage both the Arduino and the motor shield. In this case, it is also important that your shield power supply is an acceptable voltage for your Arduino, so the full shield operating voltage range of 5.5 – 24 V probably will not be available. For example, the recommended operating voltage of the Arduino Uno is 7 – 12 V.

Motor Connections and Considerations

This motor driver shield has two motor channels, M1 and M2, each of which can be used to independently control a bidirectional brushed DC motor. Each motor channel is comprised of a pair of pins—MxA and MxB—that connect to the two terminals of a DC motor and can deliver a continuous 12 A (30 A peak).

Note: It is also possible to connect a single brushed DC motor to *both* motor channels simultaneously to deliver nearly twice the current as is available from a channel by itself. See **Section 7** for more information.

3. Getting Started with an Arduino Page 10 of 25



Pololu Dual VNH5019 Motor Driver Shield User's Guide © 2001–2011 Pololu Corporation

Each VNH5019 motor driver IC has a maximum continuous current rating of 30 A. However, the chips by themselves will overheat at lower currents. In our tests on a sample unit, we were able to deliver 30 A for a few milliseconds, 20 A for several seconds, 15 A for over a minute, and 12 A for around five minutes. At 6 A, the chip just barely gets noticeably warm to the touch. The actual current you can deliver will depend on how well you can keep the motor driver cool. The shield's printed circuit board is designed to draw heat out of the motor driver chips, but performance can be improved by adding a heat sink.

This product can get **hot** enough to burn you long before the chip overheats. Take care when handling this product and other components connected to it.

Many motor controllers or speed controllers can have peak current ratings that are substantially higher than the continuous current rating; this is not the case with these motor drivers, which have a 30 A continuous rating and over-current protection that can activate at currents as low as 30 A (50 A typical). Therefore, the stall current of your motor should not be more than 30 A. (Even if you expect to run at a much lower average current, the motor can still draw short bursts of high currents, such as when it is starting, if special steps are not taken.)

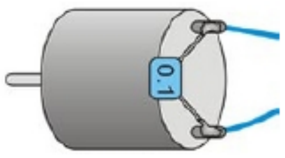
If your motor has a stall current over the driver's continuous current rating of 12 A per channel, we recommend you take extra steps to make sure that your motor will not be exposed to loads that will cause it to exceed 12 A for prolonged periods of time (or you take extra steps to keep the motor drivers cool, such as increasing air flow or adding heat sinks). Exceeding 12 A for long durations should not damage the shield, but it will eventually activate the driver's thermal protection, which might result in inadequate performance for your application.

It is not unusual for the stall current of a motor to be an order of magnitude (10×) higher than its free-run current. If you do not know your motor's stall current, you can approximate it by measuring the current it draws while held stalled at a lower voltage (such as when powered from a single battery cell) and then scaling that value linearly with voltage. For example, the stall current of a motor at 6 V is six times the stall current of that motor at 1 V. Another, less accurate method is to use a multimeter to measure the resistance between the motor terminals and then use Ohm's law to compute the stall current I at voltage V: $I = V/R$. This last method generally is not as reliable because it can be difficult to measure such small resistances accurately.

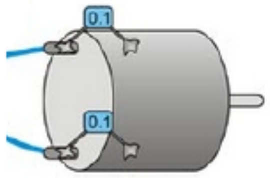
Occasionally, electrical noise from a motor can interfere with the rest of the system. This can depend on a number of factors, including the power supply, system wiring, and the quality of the motor. If you notice parts of your system behaving strangely when the motor is active, first double-check that your power supply is adequate, then consider taking the following steps to decrease the impact of motor-induced electrical noise on the rest of your system:

3. Getting Started with an Arduino Page 11 of 25

1. Solder a **0.1 µF ceramic capacitor** **[http://www.pololu.com/catalog/product/1166]** across the terminals of your motors, or solder one capacitor from each terminal to the motor case (see the pictures to the right). For the greatest noise suppression, you can use three capacitors per motor (one across the terminals and one from each terminal to the case).
2. Make your motor leads as thick and as short as possible, and twist them around each other. It is also beneficial to do this with your power supply leads.
3. Route your motor and power leads away from your logic connections if possible.
4. Place decoupling capacitors (also known as "bypass capacitors") across power and ground near any electronics you want to isolate from noise. These can typically range from 10 uF to a few hundred uF.



Motor with one 0.1 uF capacitor soldered across its terminals.



Motor with two 0.1 uF capacitors soldered from its terminals to its case.

3.d. Programming Your Arduino

Our Arduino library for the dual VNH5019 motor driver shield makes it easy to get started writing your Arduino sketches. A link to download the library, installation instructions, and the library command reference can be found on the **library's github page** **[http://github.com/pololu/Dual-VNH5019-Motor-Shield]**. Once installed, we recommend you try out the example sketch by selecting

```
File > Examples > DualVNH5019MotorShield > Demo
```

from the Arduino IDE, or by copying the following code into a new sketch:

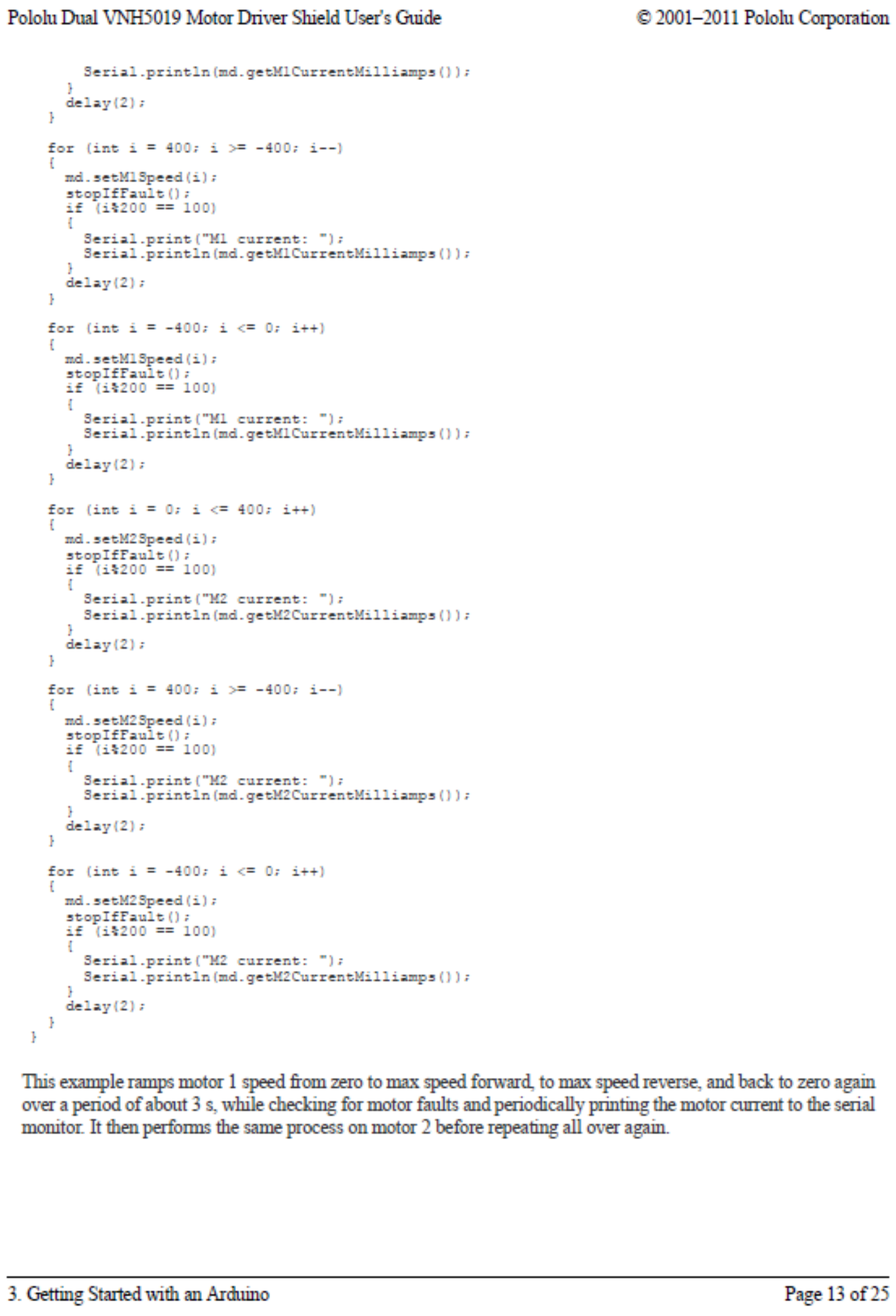
```
#include "DualVNH5019MotorShield.h"

DualVNH5019MotorShield md;

void stopIfFault()
{
  if (md.getM1Fault())
  {
    Serial.println("M1 fault");
    while(1);
  }
  if (md.getM2Fault())
  {
    Serial.println("M2 fault");
    while(1);
  }
}

void setup()
{
  Serial.begin(115200);
  Serial.println("Dual VNH5019 Motor Shield");
  md.init();
}

void loop()
{
  for (int i = 0; i <= 400; i++)
  {
    md.setM1Speed(i);
    stopIfFault();
    if (i%200 == 100)
    {
      Serial.print("M1 current: ");
```



```
      Serial.println(md.getM1CurrentMilliamps());
    }
    delay(2);
  }

  for (int i = 400; i >= -400; i--)
  {
    md.setM1Speed(i);
    stopIfFault();
    if (i%200 == 100)
    {
      Serial.print("M1 current: ");
      Serial.println(md.getM1CurrentMilliamps());
    }
    delay(2);
  }

  for (int i = -400; i <= 0; i++)
  {
    md.setM1Speed(i);
    stopIfFault();
    if (i%200 == 100)
    {
      Serial.print("M1 current: ");
      Serial.println(md.getM1CurrentMilliamps());
    }
    delay(2);
  }

  for (int i = 0; i <= 400; i++)
  {
    md.setM2Speed(i);
    stopIfFault();
    if (i%200 == 100)
    {
      Serial.print("M2 current: ");
      Serial.println(md.getM2CurrentMilliamps());
    }
    delay(2);
  }

  for (int i = 400; i >= -400; i--)
  {
    md.setM2Speed(i);
    stopIfFault();
    if (i%200 == 100)
    {
      Serial.print("M2 current: ");
      Serial.println(md.getM2CurrentMilliamps());
    }
    delay(2);
  }

  for (int i = -400; i <= 0; i++)
  {
    md.setM2Speed(i);
    stopIfFault();
    if (i%200 == 100)
    {
      Serial.print("M2 current: ");
      Serial.println(md.getM2CurrentMilliamps());
    }
    delay(2);
  }
}
```

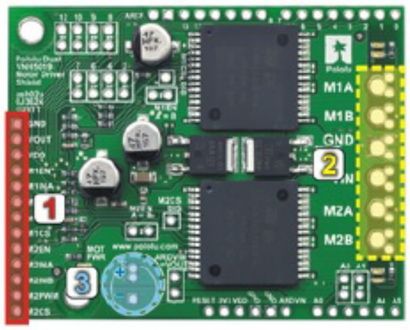
This example ramps motor 1 speed from zero to max speed forward, to max speed reverse, and back to zero again over a period of about 3 s, while checking for motor faults and periodically printing the motor current to the serial monitor. It then performs the same process on motor 2 before repeating all over again.

Note: Even if you don't have any motors yet, you can still try out this sketch and use the motor indicator LEDs for feedback that it's working properly.

4. Using as a General-Purpose Motor Driver

The set of pins along the left side of the shield provides direct access to the VNH5019 motor drivers, which means this board can be used as a general-purpose motor driver controlled by devices other than Arduinos. This section explains how to use the dual VNH5019 motor driver shield this way and provides some basic information about the motor driver pins to help get you started. However, we strongly encourage you to consult the **VNH5019 datasheet** [http://www.pololu.com/file/download/VNH5019A-E.pdf?file_id=0J504] (629k pdf) for detailed pin descriptions, truth tables, and electrical characteristics. This shield is essentially a breakout board for two VNH5019 motor driver ICs, so the datasheet is your best resource for answering questions not covered by this user's guide.

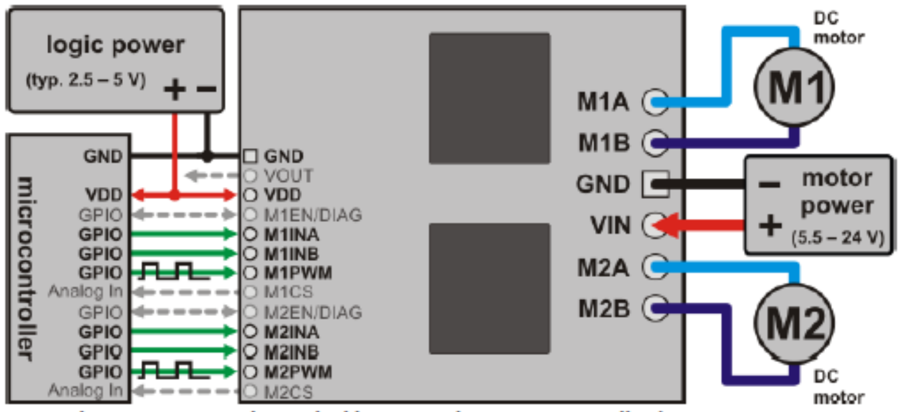
4.a. Assembly for Use as a General-Purpose Motor Driver



1. **Logic connections:** The 13 small holes along the left side of the board, highlighted in red in the above diagram, are used to interface with the motor drivers. You can optionally solder a 13×1 piece of the included 0.1" male header strip to these pins. Soldering the pins so they protrude down allows the logic side of the motor driver to be plugged into a standard **solderless breadboard** [http://www.pololu.com/catalog/category/28] or perfboard. You can also solder **0.1" female headers** [http://www.pololu.com/catalog/category/50] or custom connectors to these pins.
2. **Motor and power connections:** The six large holes/twelve small holes on the right side of the board, highlighted in yellow in the above diagram, are the motor outputs and power inputs. You can optionally solder the included 5mm-pitch terminal blocks to the board to enable temporary motor and motor power connections, or you can break off an 12×1 section of the included 0.1" header strip and solder it into the smaller through-holes that border the six large motor and motor power pads. Note, however, that the terminal blocks are only rated for 16 A, and each header pin pair is only rated for a combined 6 A, so for higher-current applications, thick wires with **high-current connectors** [http://www.pololu.com/catalog/product/925] should be soldered directly to the board.
3. **Additional power capacitor:** The motor driver shield includes three pre-installed 47 uF electrolytic power capacitors, and there is space—highlighted in blue in the above picture—to add an additional capacitor (e.g. to compensate for long power wires or increase stability of the power supply). An additional power capacitor is usually not necessary, and no additional capacitors are included with this shield.

With the exception of the pins labeled "MxEN A=B" and "MxCS_DIS", all of the through-holes not highlighted in the above diagram are only relevant when using this driver as an Arduino shield. The "MxEN A=B" and "MxCS_DIS" pins are explained in the "Pinout" portion of **Section 4.b**, but they will not be needed in typical applications and can generally be ignored.

4.b. Board Connections

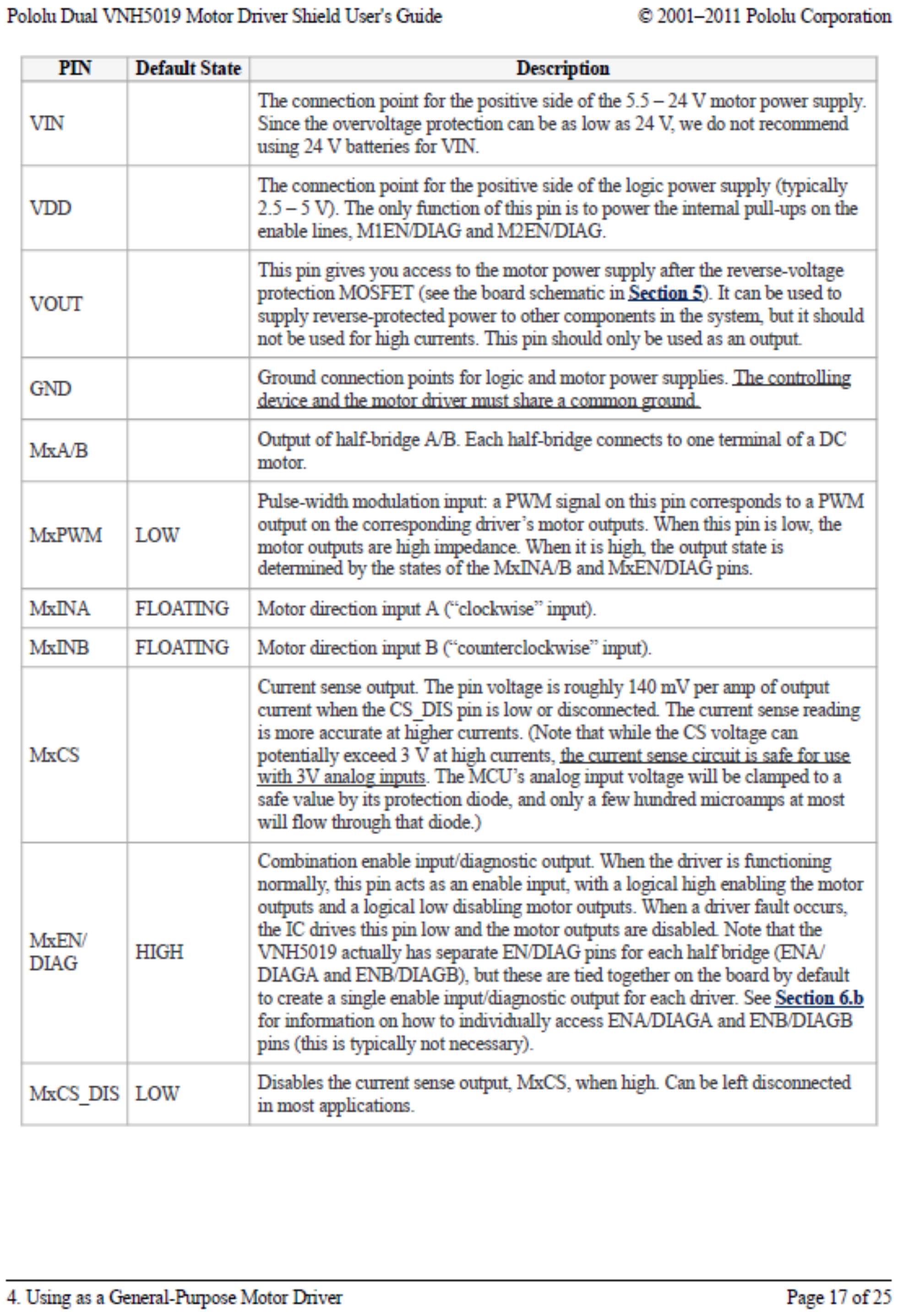


Dual VNH5019 motor driver shield connected to a microcontroller (gray connections are optional).

The above diagram shows the minimum connections typically required to interface this motor driver with a microcontroller. Note that it is possible to get by with even fewer connections if you use pulse-width modulation (PWM) on the MxINA and MxINB pins directly while holding the MxPWM pin high (e.g. by connecting it directly to VDD). This approach also results in a different kind of motor-driving: supplying the PWM signal to the MxINA/B pins directly results in drive-brake operation (outputs drive during the high portion of the PWM and are shorted together during the low portion), while supplying the PWM signal to the MxPWM pin results in drive-coast operation (outputs alternate between driving and high-impedance).

Pinout

The following table explains the board pins in detail. See the VNH5019 datasheet":/file/download/VNH5019A-E.pdf?file_id=0J504 (629k pdf) for even more detailed information about these pins, including the truth table that explains how the MxPWM and MxINA/B pins affect the MxA/B motor outputs.

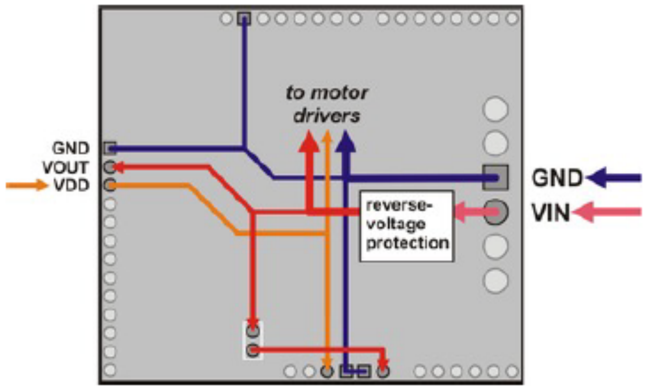


Pololu Dual VNH5019 Motor Driver Shield User's Guide © 2001–2011 Pololu Corporation

PIN	Default State	Description
VIN		The connection point for the positive side of the 5.5 – 24 V motor power supply. Since the overvoltage protection can be as low as 24 V, we do not recommend using 24 V batteries for VIN.
VDD		The connection point for the positive side of the logic power supply (typically 2.5 – 5 V). The only function of this pin is to power the internal pull-ups on the enable lines, M1EN/DIAG and M2EN/DIAG.
VOUT		This pin gives you access to the motor power supply after the reverse-voltage protection MOSFET (see the board schematic in **Section 5**). It can be used to supply reverse-protected power to other components in the system, but it should not be used for high currents. This pin should only be used as an output.
GND		Ground connection points for logic and motor power supplies. The controlling device and the motor driver must share a common ground.
MxA/B		Output of half-bridge A/B. Each half-bridge connects to one terminal of a DC motor.
MxPWM	LOW	Pulse-width modulation input: a PWM signal on this pin corresponds to a PWM output on the corresponding driver's motor outputs. When this pin is low, the motor outputs are high impedance. When it is high, the output state is determined by the states of the MxINA/B and MxEN/DIAG pins.
MxINA	FLOATING	Motor direction input A ("clockwise" input).
MxINB	FLOATING	Motor direction input B ("counterclockwise" input).
MxCS		Current sense output. The pin voltage is roughly 140 mV per amp of output current when the CS_DIS pin is low or disconnected. The current sense reading is more accurate at higher currents. (Note that while the CS voltage can potentially exceed 3 V at high currents, the current sense circuit is safe for use with 3V analog inputs. The MCU's analog input voltage will be clamped to a safe value by its protection diode, and only a few hundred microamps at most will flow through that diode.)
MxEN/ DIAG	HIGH	Combination enable input/diagnostic output. When the driver is functioning normally, this pin acts as an enable input, with a logical high enabling the motor outputs and a logical low disabling motor outputs. When a driver fault occurs, the IC drives this pin low and the motor outputs are disabled. Note that the VNH5019 actually has separate EN/DIAG pins for each half bridge (ENA/ DIAGA and ENB/DIAGB), but these are tied together on the board by default to create a single enable input/diagnostic output for each driver. See **Section 6.b** for information on how to individually access ENA/DIAGA and ENB/DIAGB pins (this is typically not necessary).
MxCS_DIS	LOW	Disables the current sense output, MxCS, when high. Can be left disconnected in most applications.

4. Using as a General-Purpose Motor Driver Page 17 of 25

Power Considerations



Dual VNH5019 motor driver shield power buses when not used with an Arduino shield.

The shield must be supplied with 5.5 to 24 V through the large VIN and GND pads on the right side of the board. A high-side reverse-voltage protection MOSFET prevents the shield from being damaged if shield power is inadvertently connected backwards.

Note that the motor driver features over-voltage protection that can kick it at voltages as low as 24 V, so we do not recommend using it with 24 V batteries (such batteries can significantly exceed 24 V when fully charged).

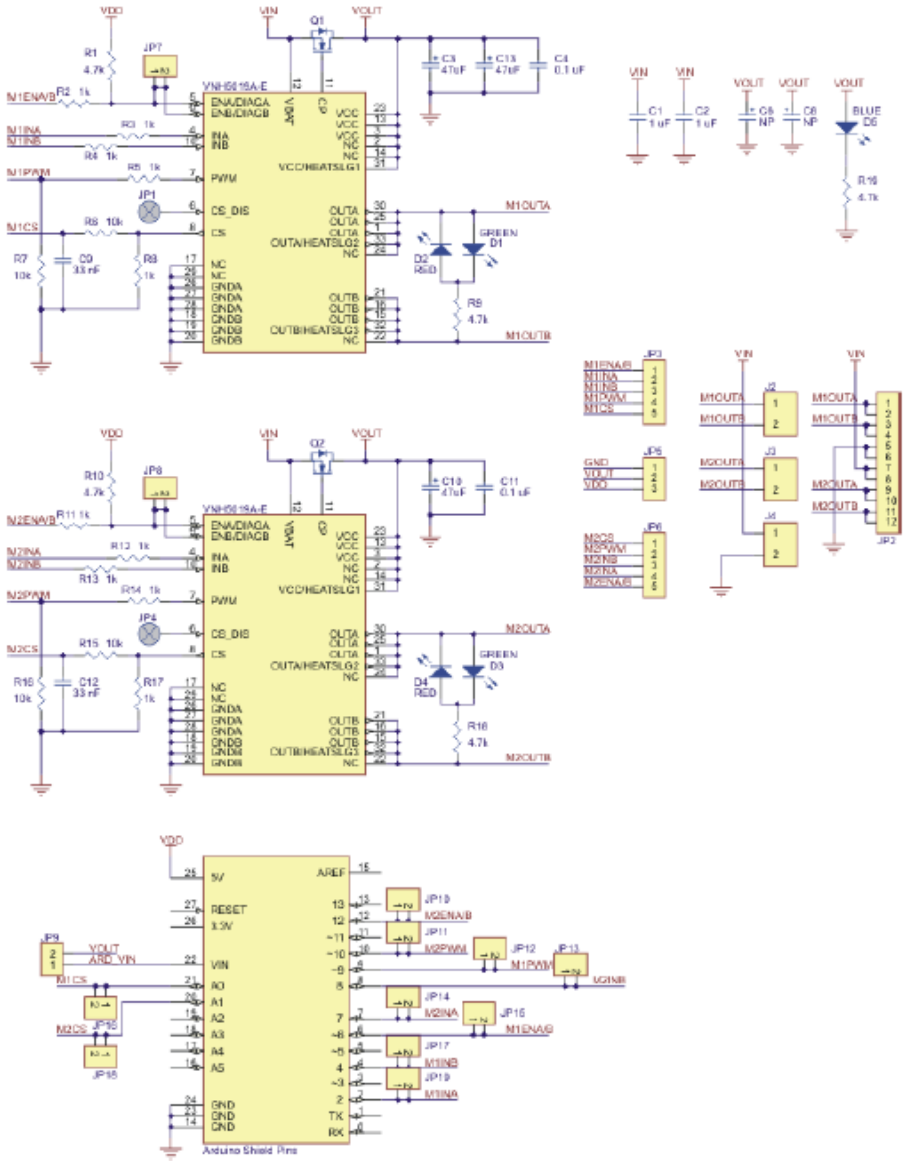
It is important that you use a power source that is capable of delivering the current your motors will require. For example, alkaline cells are typically poor choices for high-current applications, and you should almost never use a 9V battery (the rectangular type with both terminals on the same side) as your motor power supply.

Logic power at the same level as your controlling device should be supplied to the VDD pin. This will typically be between 2.5 and 5 V, but the VNH5019 motor drivers are guaranteed to treat any logic input voltage over 2.1 V as high. The only purpose of the VDD pin is to power the pull-up resistors on the EN/DIAG lines.

Motor Considerations

The motor considerations are the same as those detailed in **Section 3.c**.

5. Schematic Diagram



Schematic diagram of the Pololu dual VNH5019 motor driver shield for Arduino.

This schematic is also available as a downloadable pdf: **dual VNH5019 motor driver shield schematic [http://www.pololu.com/file/download/dual_VNH5019_shield_schematic.pdf?file_id=0J513]** (87k pdf)

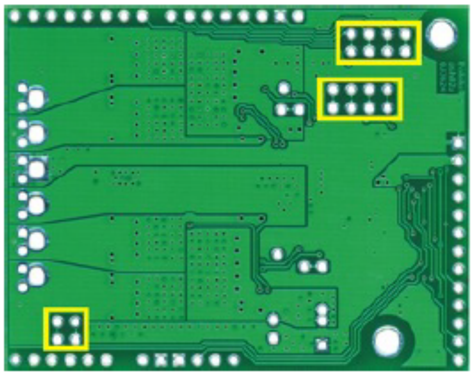
6. Customizing the Shield

This motor driver shield has several features that will not be useful in a typical application but that might benefit an advanced user. This section explains how to modify the shield from its default state to access these features.

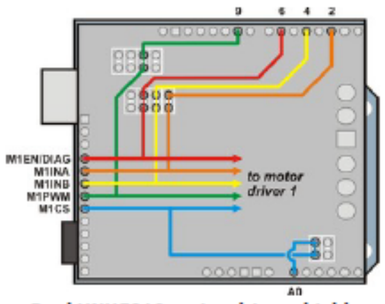
6.a. Remapping the Arduino Connections

For some applications, this shield's default Arduino pin mappings might not be convenient. For example, maybe you want to use the 16-bit Timer 1 for making music on a buzzer and would rather use PWMs from Timer 0 to control your motor speed. Or maybe you don't care about monitoring the motor current and would rather use all of your analog inputs for reading sensors. With this in mind, we designed the shield to have break points in the connection between the Arduino pins and the motor drivers. It is easy to cut the connections at these points and establish new connections to replace the broken ones if desired.

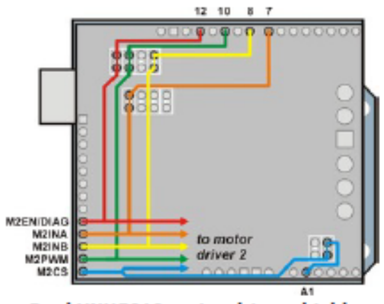
The connections between the Arduino pins and the VNH5019 motor driver pins are each made through a pair of 0.1"-spaced holes that are connected on the underside of the shield by a thin trace:


Cuttable traces on the dual VNH5019 motor driver shield for changing default Arduino pin mappings.

The following two diagrams show the default pin mapping for motor drivers 1 and 2:



Dual VNH5019 motor driver shield: Arduino pin mappings for motor driver 1.



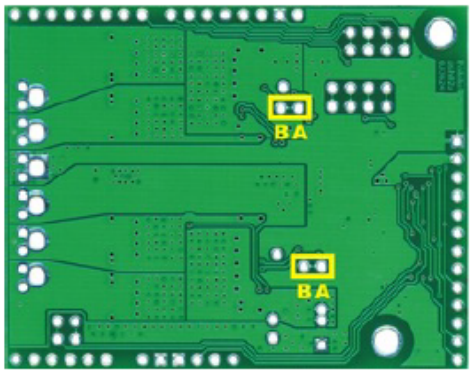
Dual VNH5019 motor driver shield: Arduino pin mappings for motor driver 2.

In all cases, the top through-hole of the pair connects to the Arduino pin and the bottom through-hole connects to the motor driver pin. To change one of the default mappings, simply use a knife to cut the trace between the appropriate pair of holes on the underside of the PCB and run a wire from a different Arduino pin to the bottom hole of the pair to create a new connection. You can later use **shorting blocks** [http://www.pololu.com/catalog/product/968] to restore the default pin mapping if you populate the severed hole pairs with 2×1 pieces of the included 0.1″ male header strip.

6.b. Accessing ENA/DIAGA and ENB/DIAGB Pins Separately

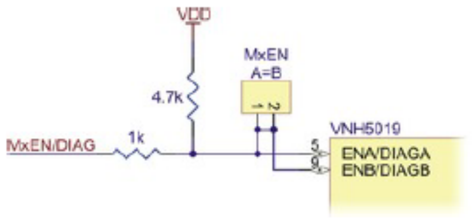
The VNH5019 motor drivers have separate enable/diagnostic pins for the A and B half bridges, but the shield combines these lines into a single enable/diagnostic pin for each motor driver in order to decrease the number of I/O lines required to monitor motor faults. This combined line is sufficient for most applications, but you can modify the board to get independent access to MxENA/DIAGA and MxENB/DIAGB if you want the additional information.

There are two pairs of 0.1″-spaced holes on the shield labeled "M1EN A=B" and "M2EN A=B". These pairs are connected on the underside of the PCB by a thin trace, with the hole labeled "A" connecting to the ENA/DIAGA pin of the corresponding motor driver and the hole labeled "B" connecting to the ENB/DIAGB pin of the motor driver:

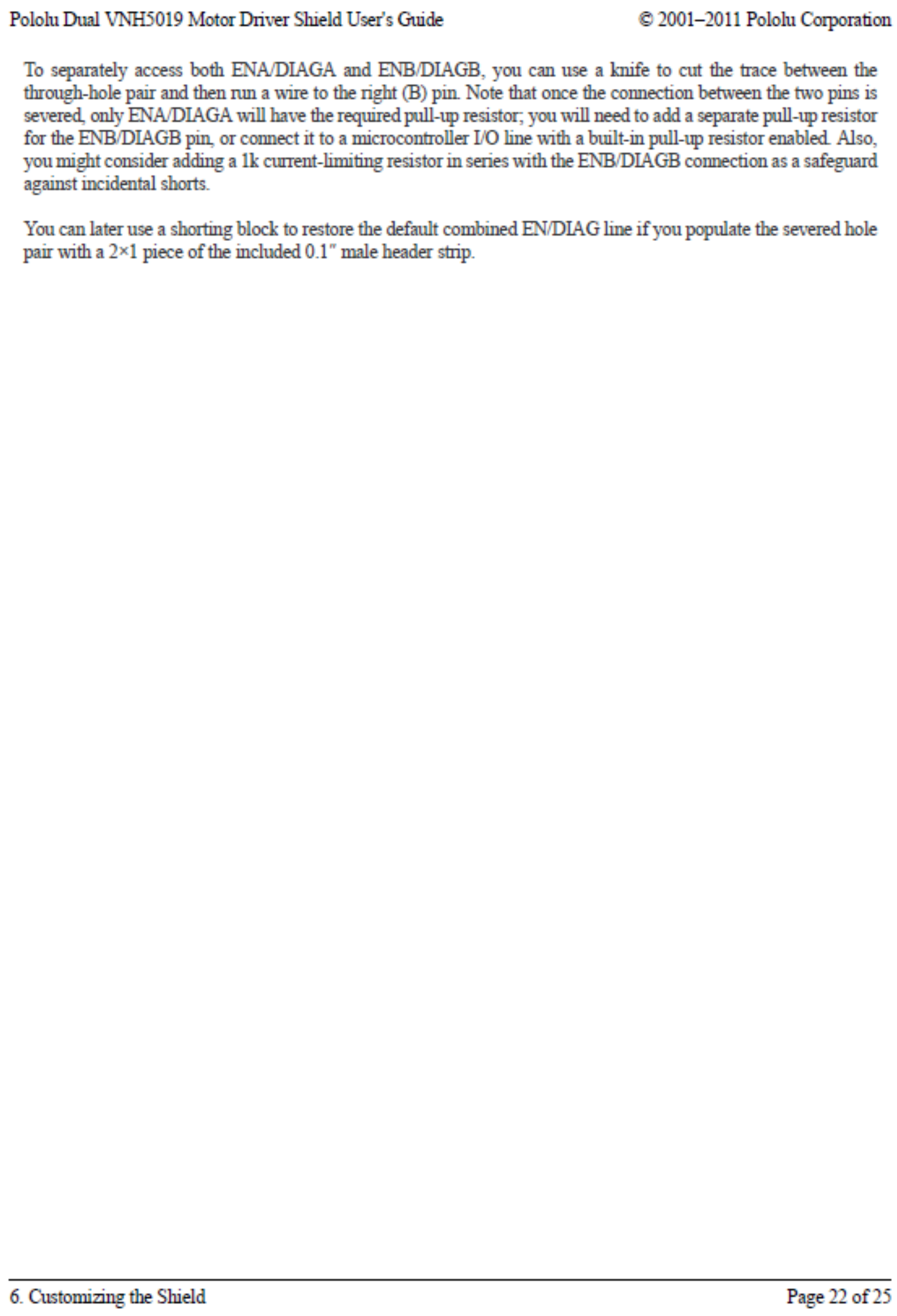


Cuttable traces on the dual VNH5019 motor driver shield for separately accessing ENA/DIAGA and ENB/DIAGB.

The following diagram shows the relevant section of the board schematic:



Schematic diagram of the enable/diagnostic circuit on the Pololu dual VNH5019 motor driver shield.



Pololu Dual VNH5019 Motor Driver Shield User's Guide © 2001–2011 Pololu Corporation

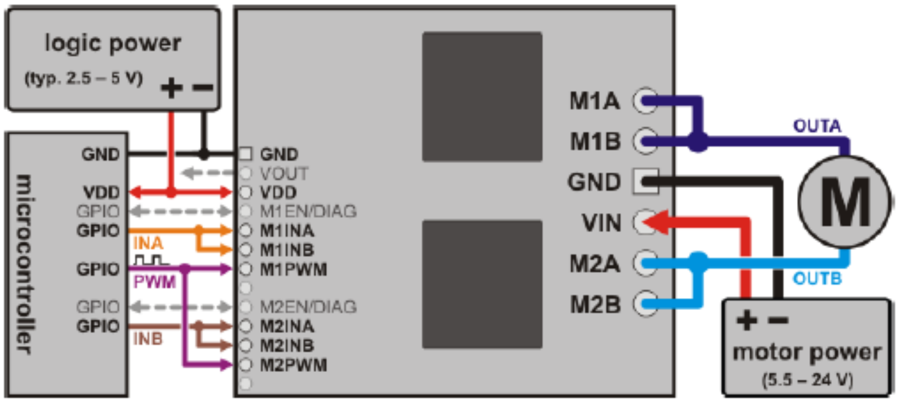
To separately access both ENA/DIAGA and ENB/DIAGB, you can use a knife to cut the trace between the through-hole pair and then run a wire to the right (B) pin. Note that once the connection between the two pins is severed, only ENA/DIAGA will have the required pull-up resistor; you will need to add a separate pull-up resistor for the ENB/DIAGB pin, or connect it to a microcontroller I/O line with a built-in pull-up resistor enabled. Also, you might consider adding a 1k current-limiting resistor in series with the ENB/DIAGB connection as a safeguard against incidental shorts.

You can later use a shorting block to restore the default combined EN/DIAG line if you populate the severed hole pair with a 2×1 piece of the included 0.1″ male header strip.

6. Customizing the Shield Page 22 of 25

7. Using the Driver in Single-Channel Mode

The dual VNH5019 motor driver shield uses two VNH5019 motor driver ICs to enable independent control of two bidirectional brushed DC motors, and each motor channel by itself is capable of delivering up to 12 A of continuous current while tolerating brief current spikes up to 30 A. If you need more power than this, however, you can combine the two motor channels into a single, doubly-powerful channel that can deliver up to a continuous 24 A (60 A peak) to one brushed DC motor. Page 23 of the **VNH5019 datasheet** **[http://www.pololu.com/file/download/VNH5019A-E.pdf?file_id=0J504]** (629k pdf) recommends accomplishing this by using each of the two motor drivers as a half-bridge. The connections shown in the following diagram turn motor driver M1 into a new "half-bridge A" and motor driver M2 into a new "half-bridge B":



Using the dual VNH5019 motor driver shield to drive a single, more powerful motor.

- Effectively create a new **INA** pin by connecting M1INA and M1B to the same digital output.
- Effectively create a new **INB** pin by connecting M2INA and M2INB to another digital output.
- Effectively create a new **PWM** pin by connecting M1PWM and M2PWM to the same PWM output.
- Effectively create a new **OUTA** pin by connecting one motor terminal to both M1A and M1B motor outputs.
- Effectively create a new **OUTB** pin by connecting the other motor terminal to both M2A and M2B motor outputs.

The motor driver truth table for this new setup then becomes:

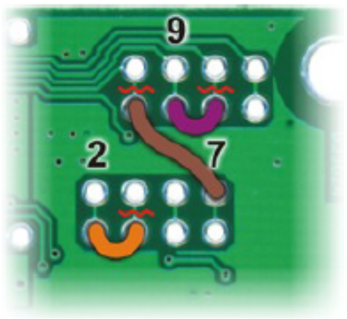
PWM	INA	INB	OUTA	OUTB	operating mode
0	X	X	OPEN	OPEN	coast
1	0	0	L	L	brake low (to GND)
1	1	0	H	L	drive "clockwise"
1	0	1	L	H	drive "counterclockwise"
1	1	1	H	H	brake high (to VOUT)

The above truth table assumes that both motor drivers are enabled and are not experiencing a fault condition (i.e. M1EN/DIAG and M2EN/DIAG pins are low). Otherwise, the disabled output will be "open" (high-impedance). Note that in this new configuration, the M1EN/DIAG pin effectively becomes a new **ENA/DIAGA** pin and the M2EN/DIAG pin effectively becomes a new **ENB/DIAGB** pin.

Note: In single-channel mode, the current sense feedback is disabled and the motor indicator LEDs do not illuminate.

Arduino Shield Modifications

If you are using this motor driver as an Arduino shield and want to reconfigure it for single-channel mode, we recommend you physically modify the Arduino pin mappings (general information on this is available in **Section 6.a**). While the reconfiguration could be done completely in software, doing so makes it very easy to accidentally command the motor drivers to create a short circuit that could damage something, so we advise against this approach. Instead, we recommend you do the following (in order):



Arduino pin remapping of dual VNH5019 motor driver shield for single-channel mode (view of underside of PCB).

- Disconnect Arduino pins **4**, **8**, and **10** from driver pins **M1INB**, **M2INB**, and **M2PWM**, respectively, by cutting the appropriate traces on the underside of the PCB with a knife. The thin, red, wavy lines in the picture to the right show where to cut. (You can restore these connections with shorting blocks should you want to revert back to the default pin mappings later.)
- Next, connect Arduino pin **2** to **M1INB**, pin **7** to **M2INB**, and pin **9** to **M2PWM**. The orange, brown, and purple "wires" in the picture to the right are one example of where these connections can be made. You might find it convenient to first solder 2×1 pieces of the included 0.1″ male header strip into these holes and then use **jumper wires** **[http://www.pololu.com/catalog/product/1706]** or, in the case of neighboring pins, **shorting blocks** **[http://www.pololu.com/catalog/product/968]** to make these connections.

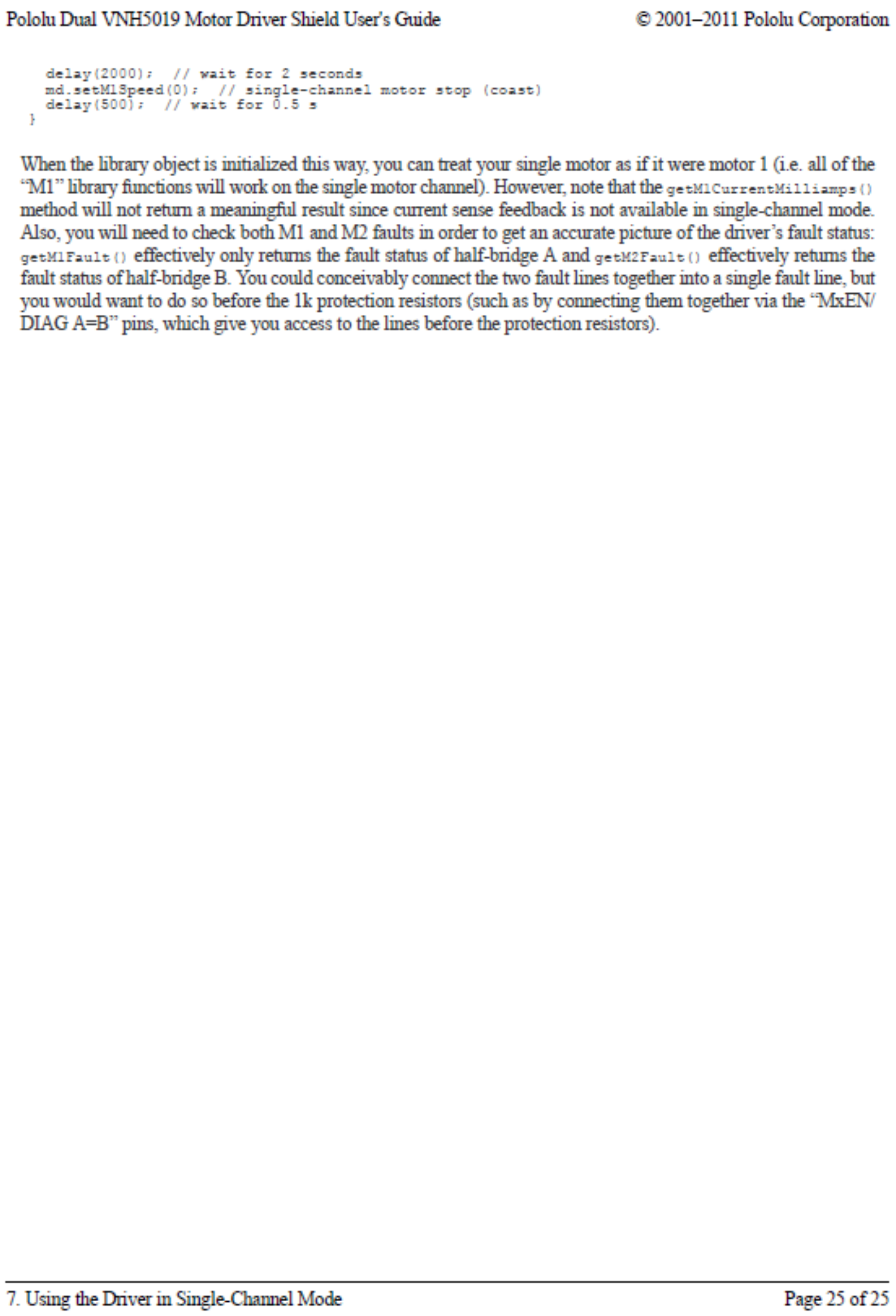
To use our **Arduino library** **[http://github.com/pololu/Dual-VNH5019-Motor-Shield]** with your newly reconfigured board, you should initialize your `DualVNH5019MotorShield` object as follows:

```
#include "DualVNH5019MotorShield.h"

// configure library with pins as remapped for single-channel operation
// this lets the single motor be controlled as if it were "motor 1"
DualVNH5019MotorShield md(2, 7, 6, A0, 2, 7, 12, A1);

void setup()
{
  md.init();
  // remaining setup code goes here
  ...
}

void loop()
{
  // loops endlessly; main loop goes here
  // the following code is a simple example:
  md.setM1Speed(400);  // single-channel motor full-speed "forward"
  delay(2000);  // wait for 2 seconds
  md.setM1Speed(0);  // single-channel motor stop (coast)
  delay(500);  // wait for 0.5 s
  md.setM1Speed(-400);  // single-channel motor full-speed "reverse"
```



```
  delay(2000);  // wait for 2 seconds
  md.setM1Speed(0);  // single-channel motor stop (coast)
  delay(500);  // wait for 0.5 s
}
```

When the library object is initialized this way, you can treat your single motor as if it were motor 1 (i.e. all of the "M1" library functions will work on the single motor channel). However, note that the `getM1CurrentMilliamps()` method will not return a meaningful result since current sense feedback is not available in single-channel mode. Also, you will need to check both M1 and M2 faults in order to get an accurate picture of the driver's fault status: `getM1Fault()` effectively only returns the fault status of half-bridge A and `getM2Fault()` effectively returns the fault status of half-bridge B. You could conceivably connect the two fault lines together into a single fault line, but you would want to do so before the 1k protection resistors (such as by connecting them together via the "MxEN/DIAG A=B" pins, which give you access to the lines before the protection resistors).

measurement SPECIALTIES™

FC22 Compression Load Cell

✓RoHS CE

- **10 – 100 lbf Ranges**
- **High Level or mV Outputs**
- **Interchangeable**
- **Compact Easy to Fixture Design**
- **CE Compliance**

DESCRIPTION

The FC22 is a medium compression force sensor that creates new markets previously unrealizable due to cost and performance constraints. The FC22 offers normalized zero and span for interchangeability and is thermally compensated for changes in zero and span with respect to temperature.

The FC22 incorporates MEAS' proprietary Microfused™ technology which employs micromachined silicon piezoresistive strain gages fused with high temperature glass to a high performance stainless steel substrate. Microfused™ technology eliminates age-sensitive organic epoxies used in traditional load cell designs providing excellent long term span and zero stability. The FC22 measures direct force and is therefore not subject to lead-die fatigue failure common with competitive designs which use a pressure capsule embedded within a silicone gel-filled cavity. Operating at very low strains, Microfused™ technology provides an essentially unlimited cycle life expectancy, superior resolution, and high over-range capabilities.

Cost-optimization of the FC22 brings your OEM product to life whether you need thousands or millions of load cells annually. Although the standard model is ideal for a wide range of applications, our dedicated design team at our Load Cell Engineering Center is ready to provide you with custom designs for your OEM applications.

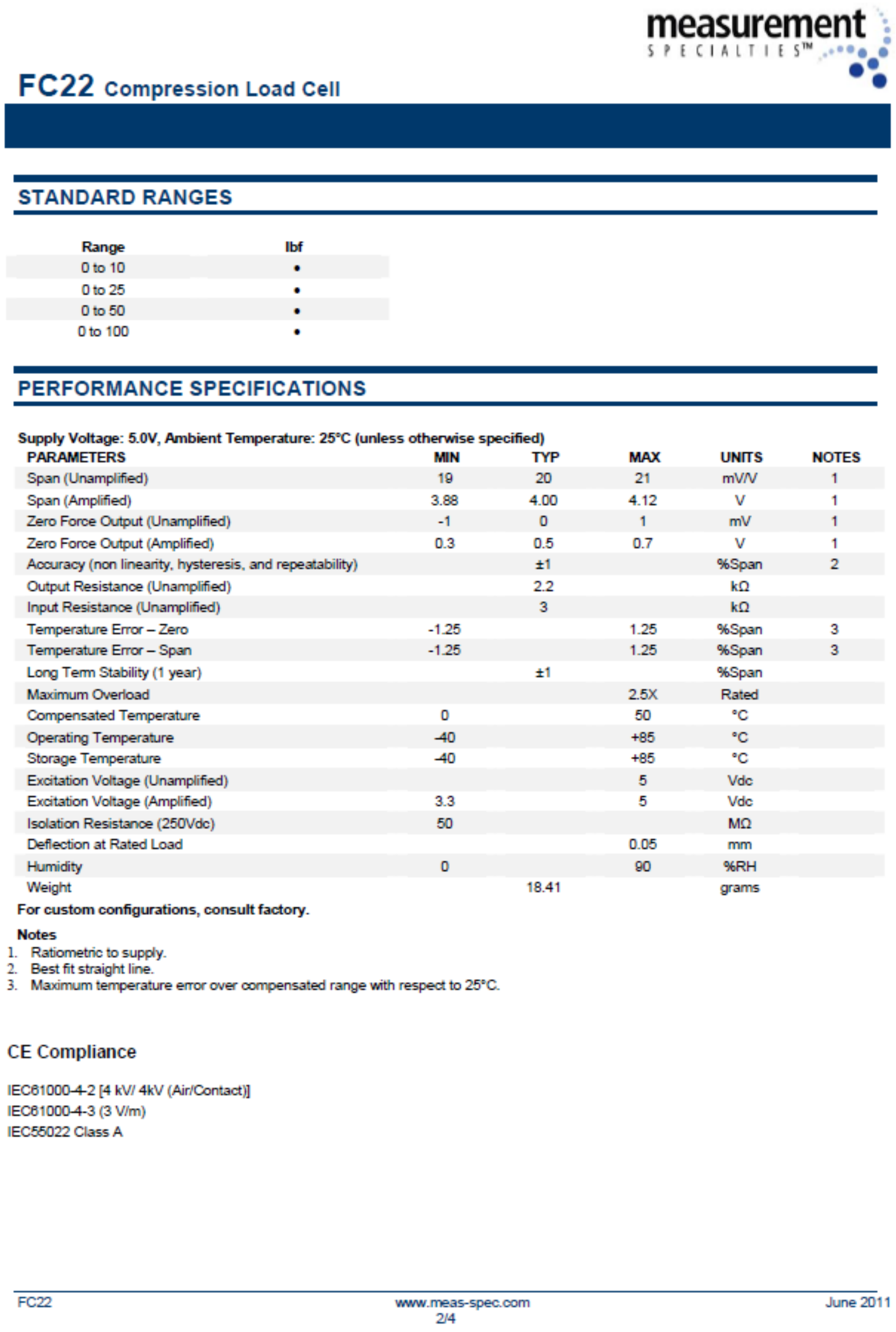
Please refer to the FS20 for lower force applications or the FC23 for higher force applications.

FEATURES

- Small Size
- Low Noise
- Robust: High Over-Range Capability
- High Reliability
- Low Deflection
- Essentially Unlimited Cycle Life Expectancy
- Low Off Center Errors
- Fast Response Time
- 10 to 100 lbf Ranges
- Reverse Polarity Protected

APPLICATIONS

- Medical Infusion Pumps
- Robotics End-Effectors
- Variable Force Control
- Load and Compression Sensing
- Exercise Machines
- Pumps
- Contact Sensing
- Weighing
- Household Appliances



measurement SPECIALTIES™

FC22 Compression Load Cell

STANDARD RANGES

Range	lbf
0 to 10	•
0 to 25	•
0 to 50	•
0 to 100	•

PERFORMANCE SPECIFICATIONS

Supply Voltage: 5.0V, Ambient Temperature: 25°C (unless otherwise specified)

PARAMETERS	MIN	TYP	MAX	UNITS	NOTES
Span (Unamplified)	19	20	21	mV/V	1
Span (Amplified)	3.88	4.00	4.12	V	1
Zero Force Output (Unamplified)	-1	0	1	mV	1
Zero Force Output (Amplified)	0.3	0.5	0.7	V	1
Accuracy (non linearity, hysteresis, and repeatability)		±1		%Span	2
Output Resistance (Unamplified)		2.2		kΩ	
Input Resistance (Unamplified)		3		kΩ	
Temperature Error – Zero	-1.25		1.25	%Span	3
Temperature Error – Span	-1.25		1.25	%Span	3
Long Term Stability (1 year)		±1		%Span	
Maximum Overload			2.5X	Rated	
Compensated Temperature	0		50	°C	
Operating Temperature	-40		+85	°C	
Storage Temperature	-40		+85	°C	
Excitation Voltage (Unamplified)			5	Vdc	
Excitation Voltage (Amplified)	3.3		5	Vdc	
Isolation Resistance (250Vdc)	50			MΩ	
Deflection at Rated Load			0.05	mm	
Humidity	0		90	%RH	
Weight		18.41		grams	

For custom configurations, consult factory.

Notes
1. Ratiometric to supply.
2. Best fit straight line.
3. Maximum temperature error over compensated range with respect to 25°C.

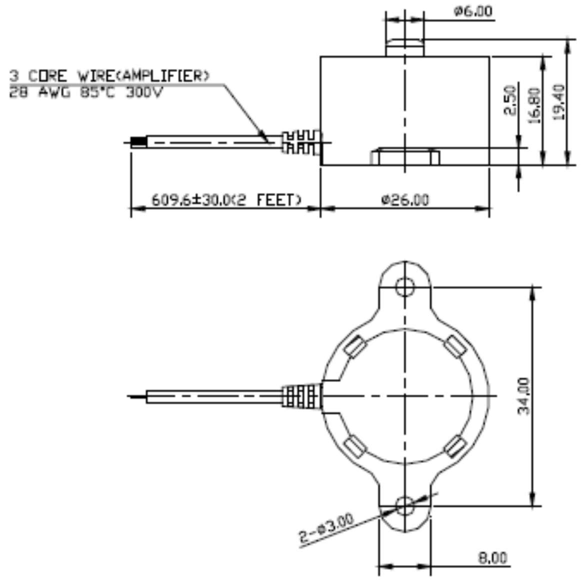
CE Compliance

IEC61000-4-2 [4 kV/ 4kV (Air/Contact)]
IEC61000-4-3 (3 V/m)
IEC55022 Class A

FC22 www.meas-spec.com 2/4 June 2011

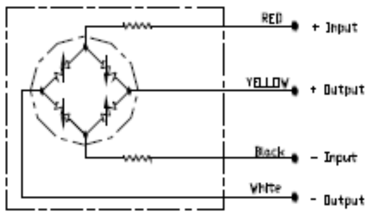
FC22 Compression Load Cell

DIMENSIONS

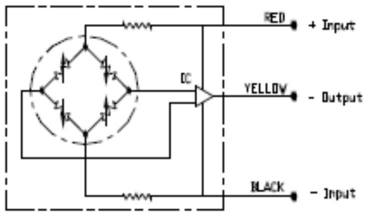


CONNECTIONS

Millivolt Bridge Version



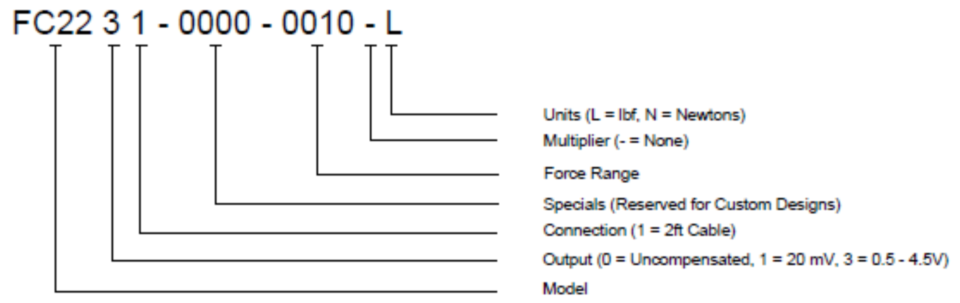
High Level Amplified Output



measurement SPECIALTIES™

FC22 Compression Load Cell

ORDERING INFORMATION


FC22 3 1 - 0000 - 0010 - L
Units (L = lbf, N = Newtons)
Multiplier (- = None)
Force Range
Specials (Reserved for Custom Designs)
Connection (1 = 2ft Cable)
Output (0 = Uncompensated, 1 = 20 mV, 3 = 0.5 - 4.5V)
Model

NORTH AMERICA

Measurement Specialties
45738 Northport Loop West
Fremont, CA 94538
Tel: 1-800-767-1888
Fax: 1-510-498-1578
Sales: pfg.cs.amer@meas-spec.com

EUROPE

Measurement Specialties
(Europe), Ltd.
26 Rue des Dames
78340 Les Clayes-sous-Bois, France
Tel: +33 (0) 130 79 33 00
Fax: +33 (0) 134 81 03 59
Sales: pfg.cs.emea@meas-spec.com

ASIA

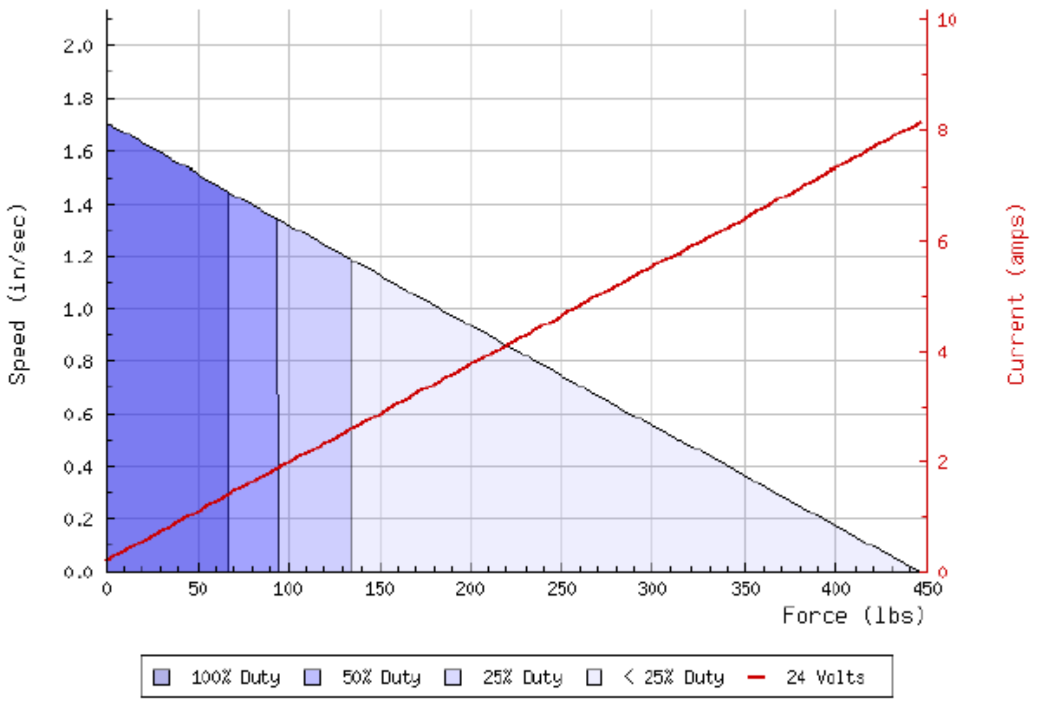
Measurement Specialties
(China), Ltd.
No. 26 Langshan Road
Shenzhen High-Tech Park (North)
Nanshan District, Shenzhen 518057
China
Tel: +86 755 3330 5088
Fax: +86 755 3330 5099
Sales: pfg.cs.asia@meas-spec.com

Bug Page | Ultra Motion **Bug** linear actuator

part number: **5-A.083-DC92_24-4-/4**

max force:	446 lbs
max speed:	1.7 in/sec
no-load current:	0.18 amps
stall current:	8.1 amps
max current:	8.1 amps
backlash:	0.001 in

actuator speed/force graph: **5 - A.083 - DC92_24**



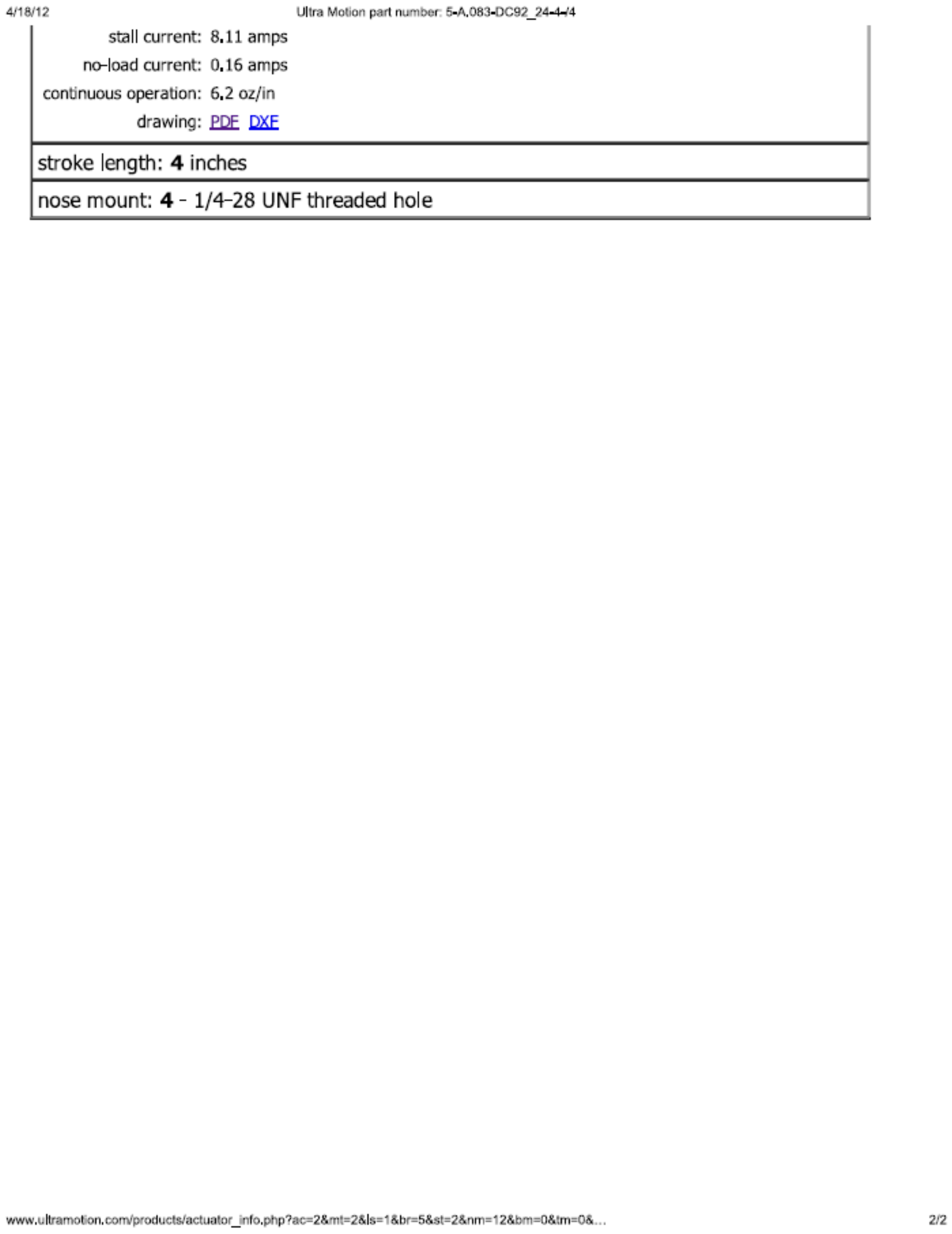
belt ratio: **5** - 5:1 ratio

lead screw: **A.083** - Acme nut, 0.0833 in/rev lead screw

pitch:	0.0833 in/rev
efficiency:	48 % (self locking)
dynamic load:	100 lb*in/sec (100% duty cycle)

motor: **DC92_24** - Low-power 24VDC servo motor

stall torque:	41.3 oz/in
no-load speed:	102.5 rev/sec



4/18/12 Ultra Motion part number: 5-A.083-DC92_24-4-/4

stall current:	8.11 amps
no-load current:	0.16 amps
continuous operation:	6.2 oz/in
drawing:	PDF DXF

stroke length: **4** inches

nose mount: **4** - 1/4-28 UNF threaded hole

www.ultramotion.com/products/actuator_info.php?ac=2&mt=2&ls=1&br=5&st=2&nm=12&bm=0&tm=0&… 2/2

Appendix C – Program

Flow Chart

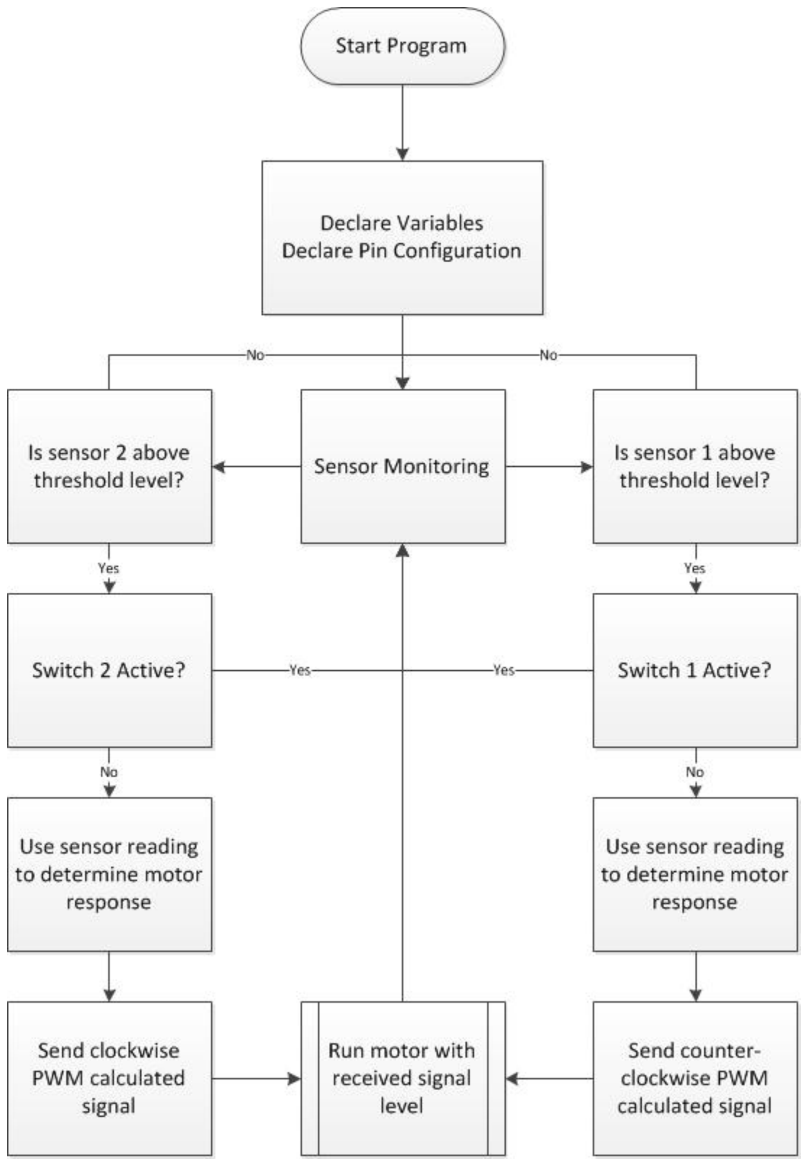

Start Program
Declare Variables
Declare Pin Configuration
No
No
Is sensor 2 above threshold level?
Sensor Monitoring
Is sensor 1 above threshold level?
Yes
Yes
Switch 2 Active?
Yes
Yes
Switch 1 Active?
No
No
Use sensor reading to determine motor response
Use sensor reading to determine motor response
Send clockwise PWM calculated signal
Run motor with received signal level
Send counter-clockwise PWM calculated signal

Figure 15 - Program Logic

Code

```
//Exoskeleton Rehabilitative Arm
//ERA_R0.1
#include "DualVNH5019MotorShield.h"

DualVNH5019MotorShield md;

//Setup fixed variables for pin connections
int UpperSwitch = 11;
int LowerSwitch = 13;
int PushButton = 5;
int LED1 = 0;
int LED2 = 1;
int LED3 = 3;
int FrontSensor = A2;
int BackSensor = A3;

//Setup global variables
byte statusone;
byte laststatusone;
byte statustwo;
byte laststatustwo;
byte motordirection;
int switchmode = 0;
int buttonstate;
int lastbuttonstate;
float backvalue;
float frontvalue;
int motorspeed;
int motorchange = 2000;
long previousMillis = 0;


void setup(){
        md.init();                                                   //Start motor shield custom library
        Serial.begin (115200);                          //Begin monitor transmission
        pinMode(UpperSwitch, INPUT);      //Pin mode for 1st switch
        pinMode(LowerSwitch, INPUT);      //Pin mode for 2nd switch
        pinMode(PushButton, INPUT);                     //Pin mode for Slide Switch 1st Position
        pinMode(LED1, OUTPUT);                          //Pin mode for LED 3
        pinMode(LED2, OUTPUT);                          //Pin mode for LED 2
        pinMode(LED3, OUTPUT);
}

void loop(){
        //Serial.print(switchmode);
        //Gather readings from sensors
        switchone();
        switchtwo();
        frontsensor();
```

```
		backsensor();

		//Set current Circuit Run Time variable
		unsigned long currentMillis = millis();


		//Monitor program state mode
		buttonstate = digitalRead(PushButton);
		if (buttonstate != lastbuttonstate){
				if (buttonstate == 1){
						switchmode++;
						if (switchmode > 2){
								switchmode = 0;
						}
				}
		}
		lastbuttonstate = buttonstate;

		//Excute programs
		//If pushbutton mode is set to two
		if (switchmode == 1){										//Program 2
				digitalWrite(LED2, HIGH);
				digitalWrite(LED1, LOW);
				digitalWrite(LED3, LOW);
				if(frontvalue>0.25 && frontvalue>backvalue){				//Low tolerance for ease of
acivation
	Serial.print("...");
						Serial.print(frontvalue);
						Serial.println("....Driving motor up");
						motorspeed = 40*(frontvalue-0.25);					//Motor Ramp
Up function
						if(statusone == 0){
								md.setM1Speed(motorspeed);
								stopiffault();
		//Serial.println(md.getM1CurrentMilliamps());
						}
						else{
								md.setM1Speed(0);
								stopiffault();
						}
						delay(2);
				}
				if(backvalue>1.25 && backvalue>frontvalue){					//Low tolerance for ease
of activation
	Serial.print("...");
						Serial.print(backvalue);
						Serial.println("....Driving motor down");
						motorspeed = -40*(backvalue-1.25);					//Motor Ramp
Up function
						if(statustwo == 0){
								md.setM1Speed(motorspeed);
								stopiffault();
		//Serial.println(md.getM1CurrentMilliamps());
```

```
				}
				else{
					md.setM1Speed(0);
					stopiffault();
				}
			delay(2);
			}
			else{
				md.setM1Speed(0);
				stopiffault();
			}
		}
		//If pushbutton mode is set to three
		else if (switchmode == 2){								//Program 3
			digitalWrite(LED3, HIGH);
			digitalWrite(LED1, LOW);
			digitalWrite(LED2, LOW);
			if(frontvalue>1 && frontvalue>backvalue){		//High tolerance for resistive motion
	    Serial.print("...");
				Serial.print(frontvalue);
				Serial.println("....Driving motor up");
				motorspeed = 10*frontvalue;						//Motor Ramp
Up function
				if(statusone == 0){
					md.setM1Speed(motorspeed);
					stopiffault();
		//Serial.println(md.getM1CurrentMilliamps());
				}
				else{
					md.setM1Speed(0);
					stopiffault();
				}
				delay(2);
			}
			if(backvalue>1 && backvalue>frontvalue){		//High tolerance for resistive motion
	    Serial.print("...");
				Serial.print(backvalue);
				Serial.println("....Driving motor down");
				motorspeed = -5*backvalue;						//Motor Ramp
Downb function
				if(statustwo == 0){
					md.setM1Speed(motorspeed);
					stopiffault();
		//Serial.println(md.getM1CurrentMilliamps());
				}
				else{
					md.setM1Speed(0);
					stopiffault();
				}
			delay(2);
			}
			else{
				md.setM1Speed(0);
```

```
                        stopiffault();
                }
        }
        //If pushbutton mode is set to one
        else{
        //Program 1
                digitalWrite(LED1, HIGH);
                digitalWrite(LED2, LOW);
                digitalWrite(LED3, LOW);
                if(motordirection == 0){
                        motorspeed = 50;
            Serial.println("Up");
                        md.setM1Speed(motorspeed);
                        stopiffault();
                        //Serial.println(md.getM1CurrentMilliamps());
                        delay(2);
                }
                else{
                        motorspeed = -50;
            Serial.println("Down");
                        md.setM1Speed(motorspeed);
                        stopiffault();
                        //Serial.println(md.getM1CurrentMilliamps());
                        delay(2);
                }
        }
}

void stopiffault(){                     //Check for motor error
        if (md.getM1Fault()){
                while(1);
                }
}

void switchone(){                       //Check status of Upper Switch
        statusone = digitalRead(UpperSwitch);
        if (statusone != laststatusone){
                if (statusone == 1){
                motordirection = 1;                     //Set motor direction
                }
        }
        laststatusone = statusone;
}

void switchtwo(){                       //Check status of Lower Switch
        statustwo = digitalRead(LowerSwitch);
        if (statustwo != laststatustwo){
                if (statustwo == 1){
                motordirection = 0;                     //Set motor direction
                }
        }
        laststatustwo = statustwo;
}
```

```
void frontsensor(){                                   //Read sensor in front of wrist
            frontvalue = analogRead(FrontSensor);
            frontvalue = (0.0566*frontvalue) - 5.716;                     //Calibrated transduction function
   //Serial.print("Force Value = ");
   //Serial.println(frontvalue);
}

void backsensor(){                                    //Read sensor in back of wrist
            backvalue = analogRead(BackSensor);
            backvalue = (0.0121*backvalue) - 1.1433;                   //Calibrated transduction function
   //Serial.print("Force Value = ");
   //Serial.println(backvalue);
}
```

Appendix D - Circuit Chart

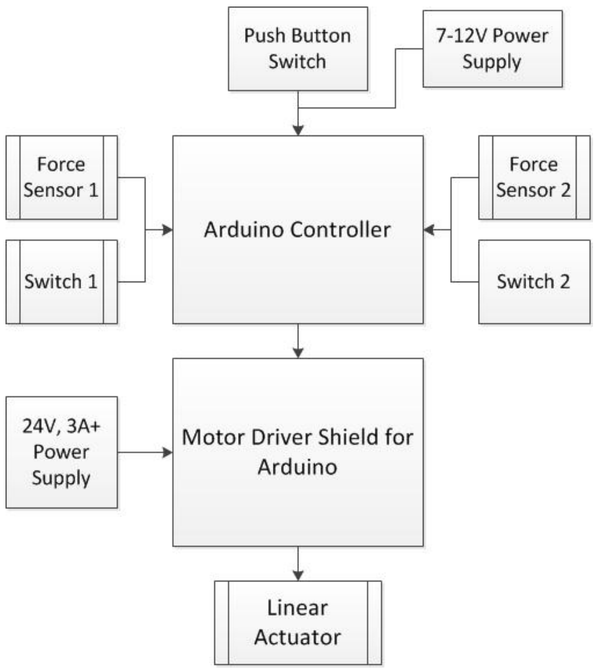


Figure 16 - Circuit Diagram