Actuators and You,   
 An In Depth Guide.

Class:    ECE 579  
Author: David Gaskin  
Professor: Marek Perkowski  
Contributors: Omar Mohsin, Ali Alnasser

Table of Contents

Introduction/Project Description..………………………………...….3  
Setup………………………………………………………………….5

Parts……………………………………………………………7

Procedure …………………………………………………………...10

Actuators……………………………………………………..10  
Motor Controllers…………………………………………….12

Complete Wiring Setup……………………………………...15

Terminal Block for Power…………………………………...17

Arduino ……………………………………………………...19

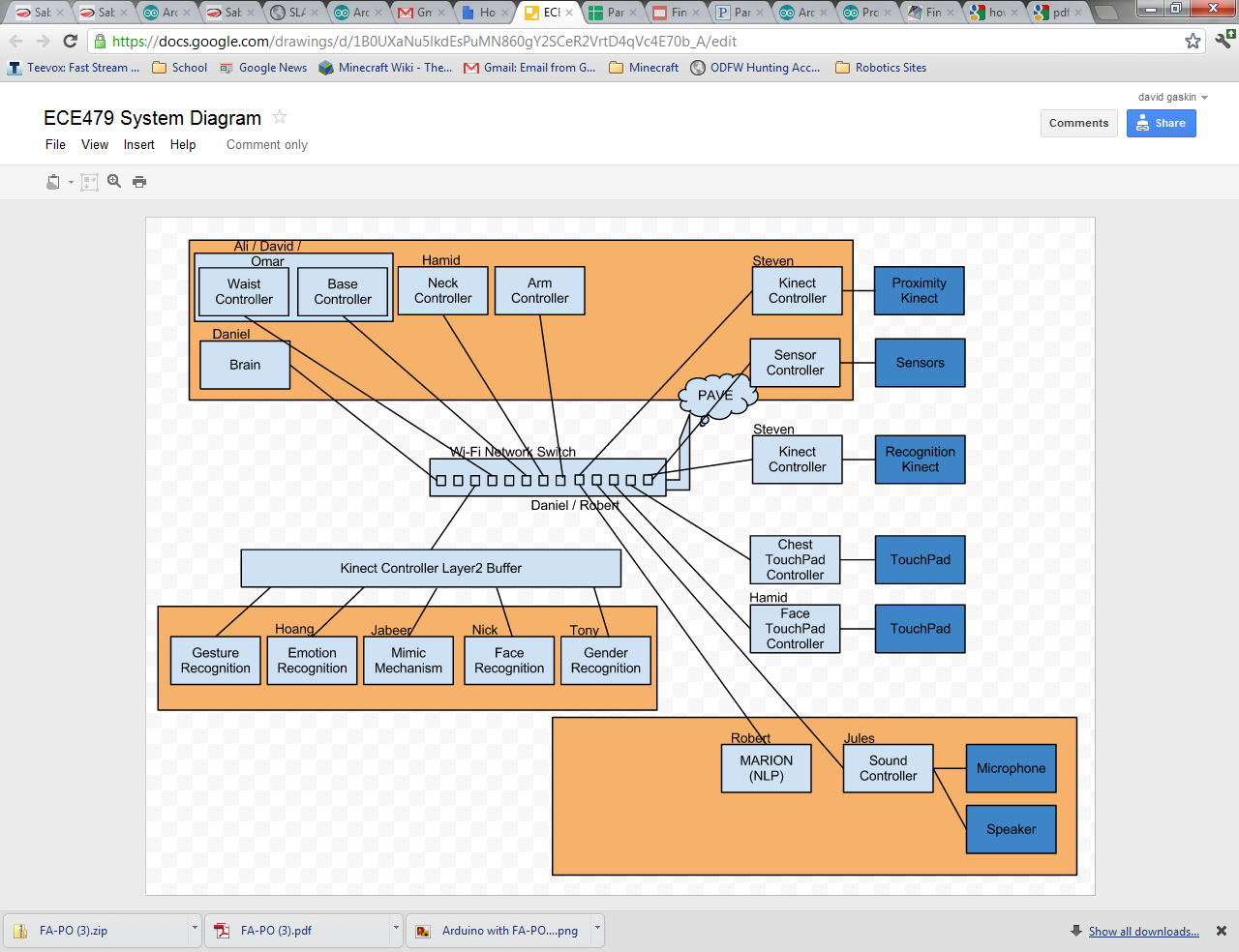
Future Development………………………………………………...28  
Appendix …………………………………………………………..29   
 Arduino Board……………………………………………….29  
 Actuators……………………………………………………..29  
 Motor Controllers…………………………………………….34  
 Code………………………………………………………….56

Time Log……………………………………………………..61

          Parts List, Cost and Source…………………………………..61

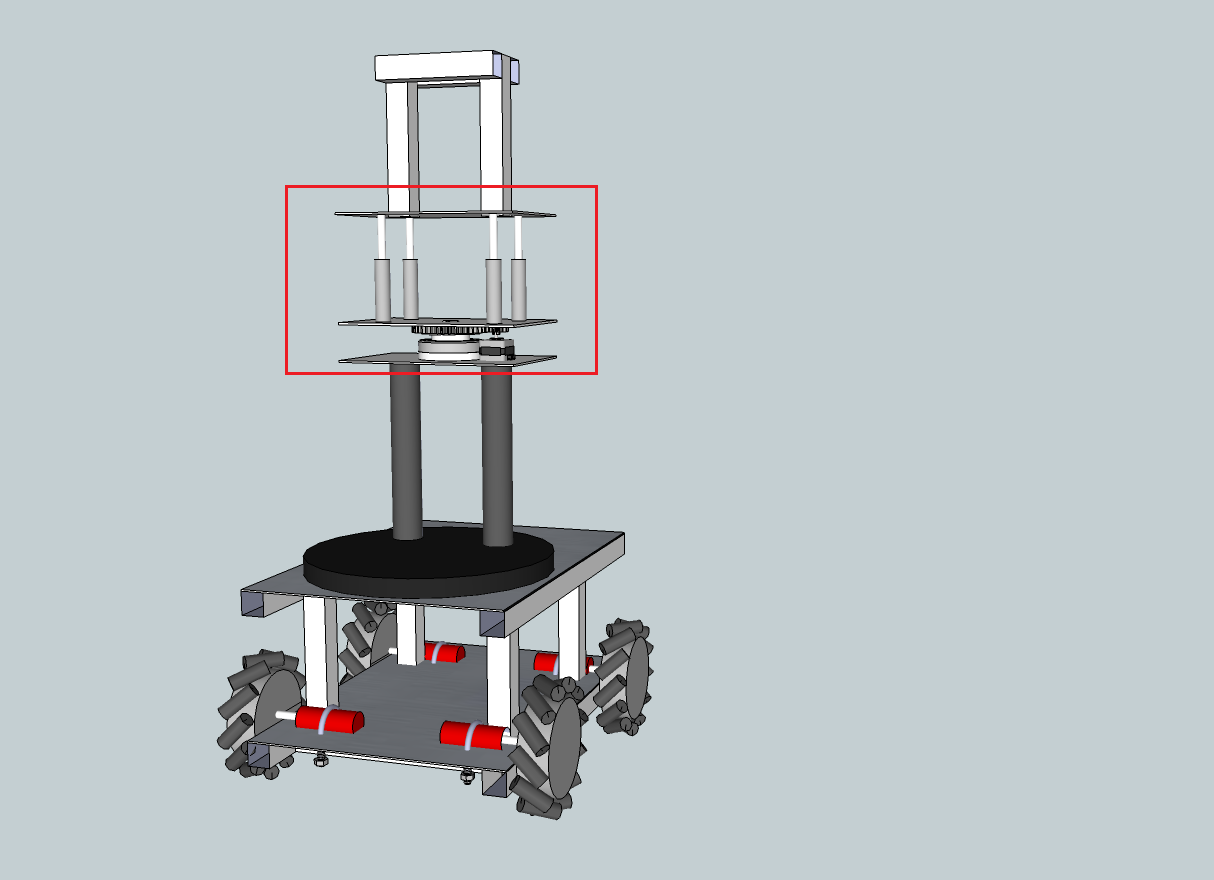
**Introduction/Project Description**

The purpose of this project was to create a mobile base platform that could:

* Move and support multiple components on a fully mobile robot
* Have many degrees of freedom, both on the mobile base portion as well as the robotic joint
* Be able to rotate on the wheels, and be able to bend at the waist in any direction.
* Be able to support the weight and integration of other modules, shown below:

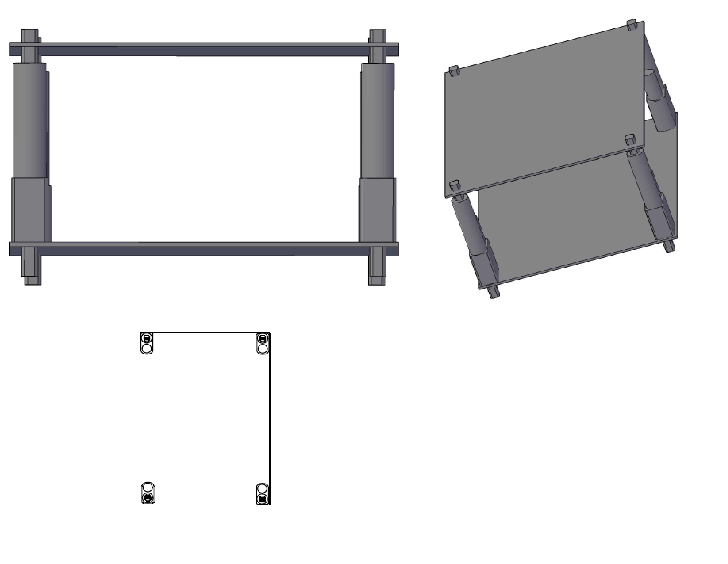
My specific portion of the project required me to:

* Integrate 4 actuators with an arduino board and 2 motor controllers
* Allow for position specific control
* Machine mountings for the actuators that could support the weight of the upper half of the robot, and withstand various motions.
* Program the arduino board, working with serial communications and the motor controllers, getting feedback from the actuators in order to move and maintain certain positions.
* See whole robot below, section in red is the section the report discusses.
* Lean side to side at most 21.9 degrees, and lean forward/backwards at most 40.51 degrees. This was determined using the measurements of the actuators (extends a max of four inches, roughly 10 inches apart along the long side, roughly 6.5 inches apart on the widthe. SIN(4/10.2)=21.9 degrees, TAN (4/6.5) = 40.51 degrees
* Support up to 75 lbs



Materials for the structure:

* 2x 8.5x12x3/16” aluminum plates
* 4 x1/16” thick washers with ¼” holes
* 4x 1/8” cotter pins
* CAD program



The above was drawn using Autodesk, which gives out free versions to students (<http://students.autodesk.com/?nd=download_center>). It is recommended that the student replicating this gains access to an ME lab, as it is difficult to use what is in the robotics lab to create very nice looking square holes. The dimensions of this are roughly half as long and wide as the base. It will eventually become connected to the base and support the top using aluminum bars, and be able to twist on the bottom.

**Setup**

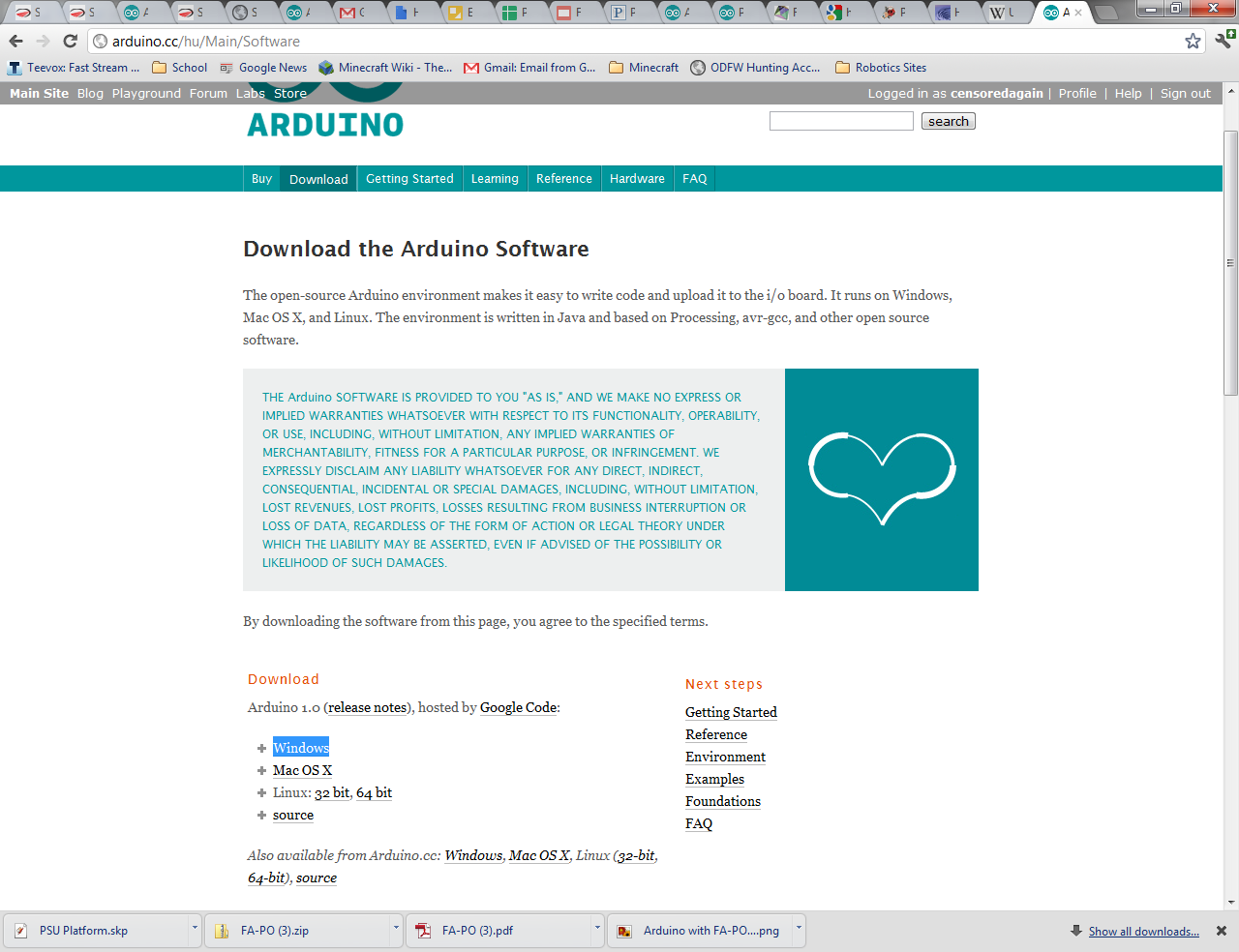
**Arduino**

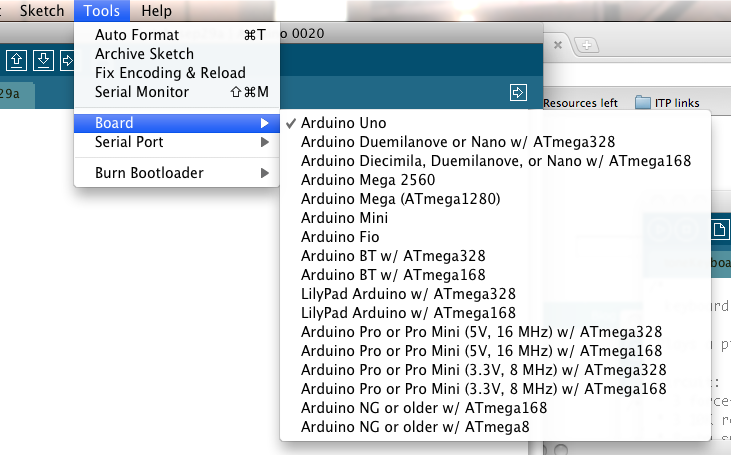
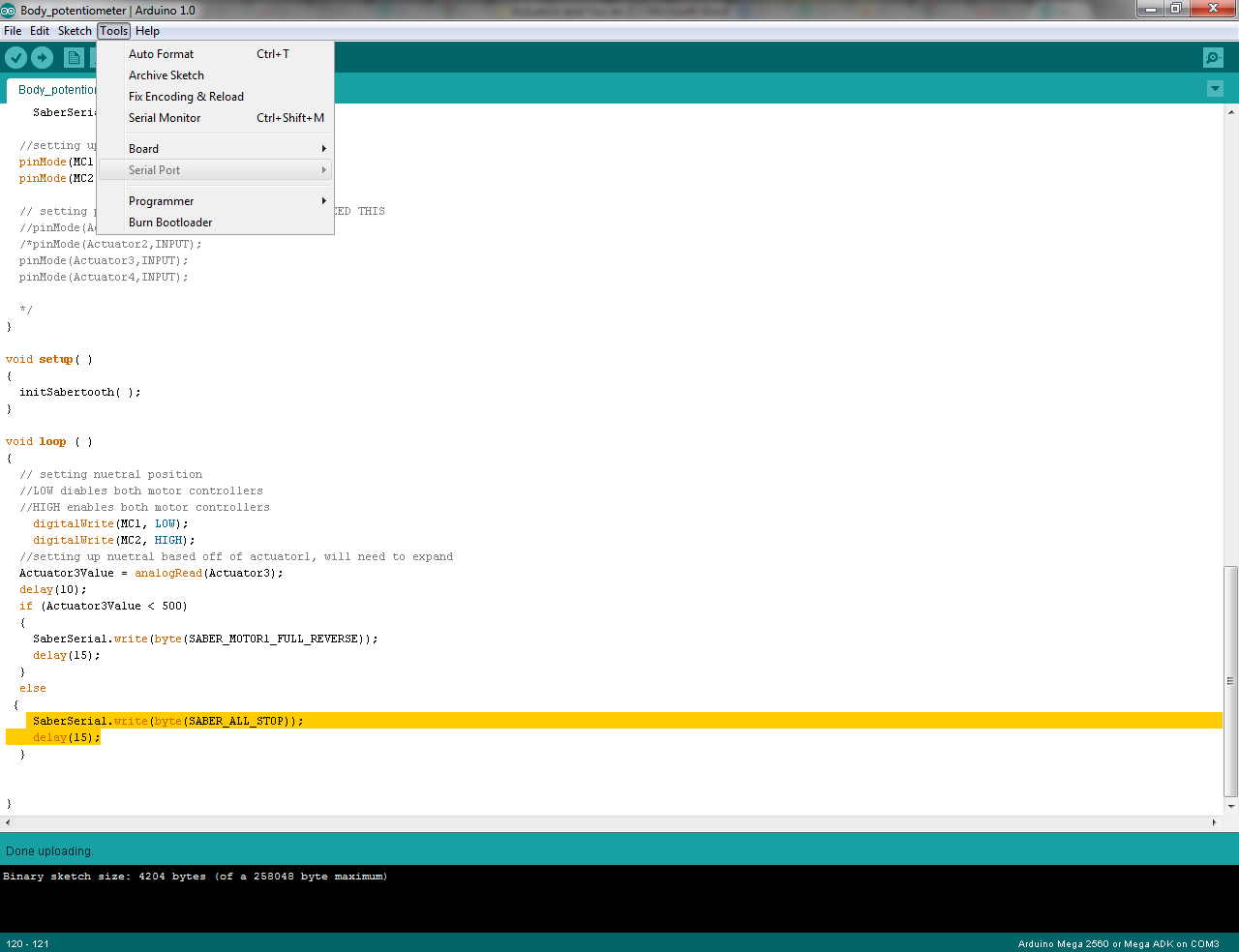
Materials required:

1. Arduino Board
2. USB A to B cable
3. Internet connection

Steps:

1. Download the Arduino software for your platform (<http://arduino.cc/hu/Main/Software>)



1. Click on the link, DL the zip file.
2. Go to <http://arduino.cc/en/Guide/HomePage> and select your platform if you struggle with the installation
3. Plug in your Arduino, and install the drivers:
   1. Plug in your board and wait for Windows to begin it's driver installation process.  After a few moments, the process will fail, despite its best efforts
   2. Click on the Start Menu, and open up the Control Panel.
   3. While in the Control Panel, navigate to System and Security. Next, click on System. Once the System window is up, open the Device Manager.
   4. Look under Ports (COM & LPT).  You should see an open port named "Arduino UNO (COMxx)"
   5. Right click on the "Arduino UNO (COmxx)" port and choose the "Update Driver Software" option.
   6. Next, choose the "Browse my computer for Driver software" option.
   7. Finally, navigate to and select the Uno's driver file, named **"ArduinoUNO.inf"**, located in the "Drivers" folder of the Arduino Software download (not the "FTDI USB Drivers" sub-directory).
   8. Windows will finish up the driver installation from there.
4. Launch your arduino application
5. Select your board (Arduino Mega 2560 in our case)
6. Select your serial port. If you don’t know which COM port select, disconnect your device after noting which ports are showing. The one that disappeared is the one you want. 
7. Upload your program

**Parts**

Here is a list of materials required to complete this project:

Parts ordered online:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Source** | **Price** | **QTY** | | **Shipping** | **Total** |
| Arduino Mega 2560 Microcontroller rev 3 | http://www.trossenrobotics.com/p/arduino-mega-2560.aspx | 64.95 | | 1 | 0 | 64.95 |
| Arduino Ethernet Shield | http://www.sparkfun.com/products/9026 , http://www.trossenrobotics.com/p/arduino-ethernet-shield.aspx | 45.95 | | 1 | 0 | 45.95 |
| Arduino Box | http://www.trossenrobotics.com/p/arduino-project-box.aspx | 11.95 | | 1 | 0 | 11.95 |
| Pig tail for power | http://www.trossenrobotics.com/store/p/6612-Barrel-Jack-Female-Pigtail-Lead-2-1-5-5mm.aspx | 2.95 | | 2 | 0 | 5.9 |
| 6vDC power for board | http://www.trossenrobotics.com/p/power-supply-6vdc-2a.aspx | 9.95 | | 1 | 0 | 9.95 |
| 12v 5A pwr supply for actuators | http://www.trossenrobotics.com/p/power-supply-12vdc-5a.aspx | 19.95 | | 1 | 0 | 19.95 |
| Servo wire | http://www.trossenrobotics.com/store/p/6421-50-Feet-Standard-Servo-Wire.aspx | 12.95 | | 1 | 0 | 12.95 |
| Motor Controller | http://www.trossenrobotics.com/store/p/5104-Sabertooth-dual-5A-motor-driver-for-R-C.aspx | 59.99 | | 2 | 0 | 119.98 |
| Firgelli Actuators | http://www.firgelliauto.com/product\_info.php?cPath=109&products\_id=159 | 129.99 | | 4 | 0 | 519.96 |
| Actuator Mounting | http://www.firgelliauto.com/product\_info.php?products\_id=54 | 9 | | 4 | 0 | 36 |
| **Totals** |  |  | |  |  | **847.54** |

Parts purchased at local hardware store:

|  |  |  |
| --- | --- | --- |
| Item | Cost |  |
| Metal Knockout bit | 44.97 | For drilling out large metal holes |
| Dremmel carbide blades | 8.98 | For shaping holes |
| Dremmel EZ lock kit | 16.53 | For locking blades to dremmel |
| UPS shipping | 10.52 | For exchanging Motor driver (they sent the wrong part) |
| Terminal kit with tool | 9.04 | To make the terminal block |
| Terminals for 10-12, 14-16 gauge wires | 5.58 | To make the connections for the terminal block |
| Battery Clamps | 3.29 | For connecting terminal to power source |
| Washers | 1.50 | For spacing out the actuators on the plate |
| 2x terminal blocks | 15.98 | For making a easy to connect to power source |
| 14 gauge black wire | 1.50 | For terminal block and connections |
| 14 gauge red wire | 1.50 | For terminal block and connections |
| 10 gauge wire | 2.35 | For battery connections |
| Safety pins | 3.12 | For harness connections for aluminum plate and actuators |
| Total | 124.86 |  |

**Procedure**

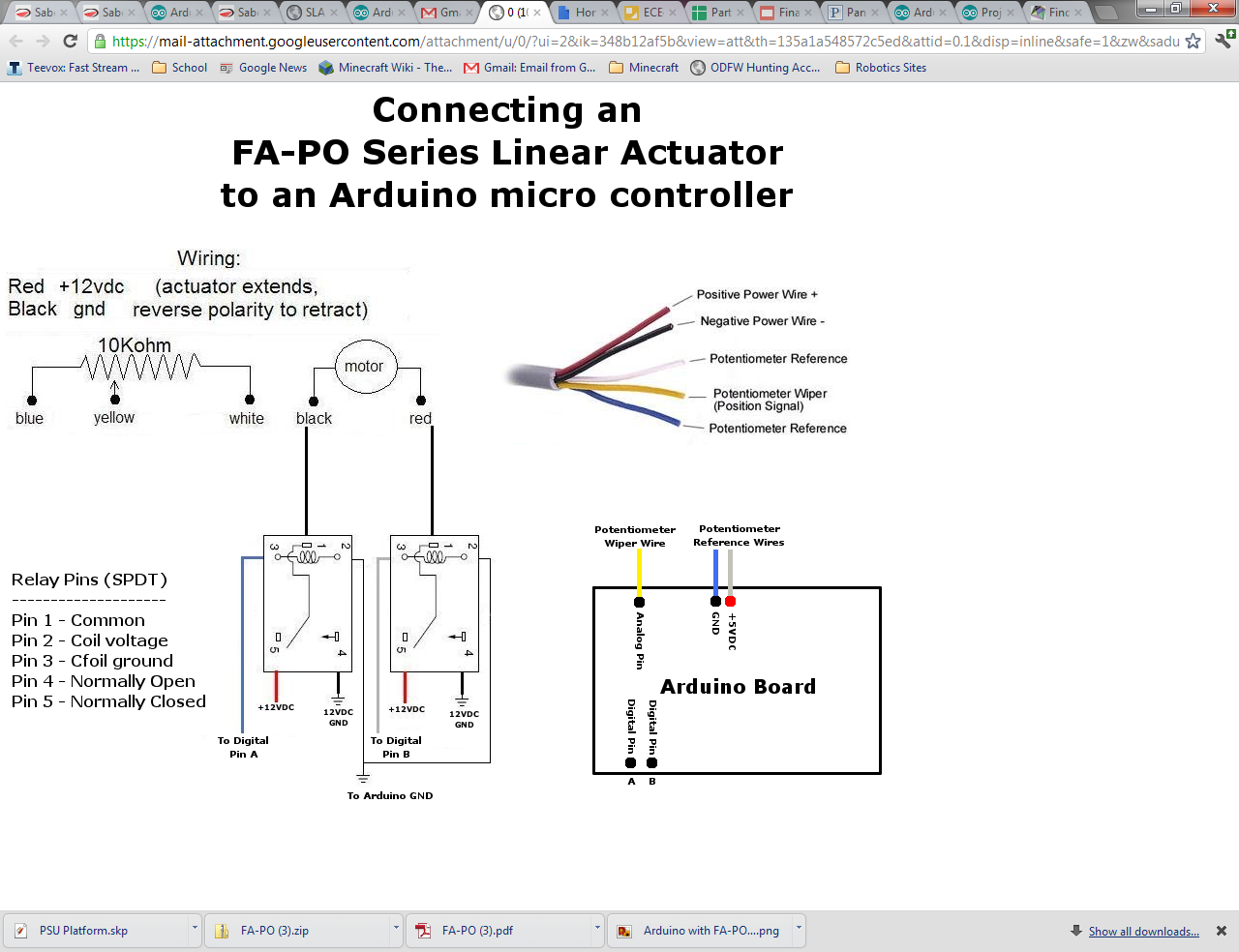
**Materials**(for sources, see parts list in appendix):

1. 2x Sabertooth 2x5 motor controllers
2. 4 Firgelli actuators
3. 1 Arduino Mega
4. 2x Aluminum plates, with four square holes cut into them (as described above).

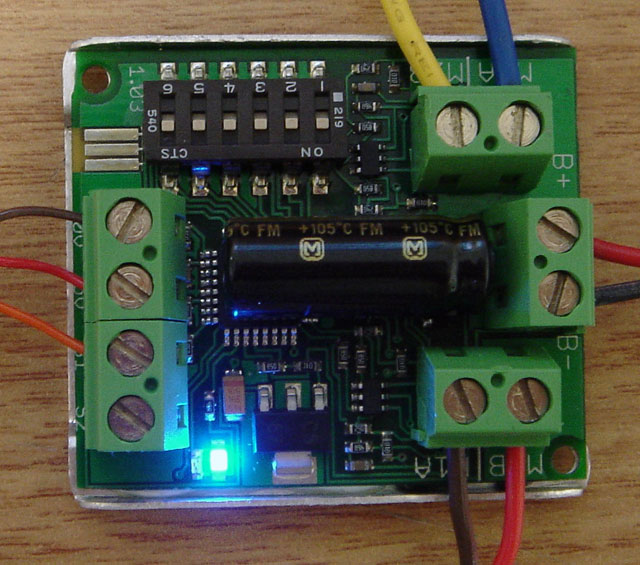
**Actuators**

Set the actuators in the mounting of your support plate. Place the washers over the ends sticking out of the bottom of the plate, and insert the cotter pins so they don’t wiggle around.

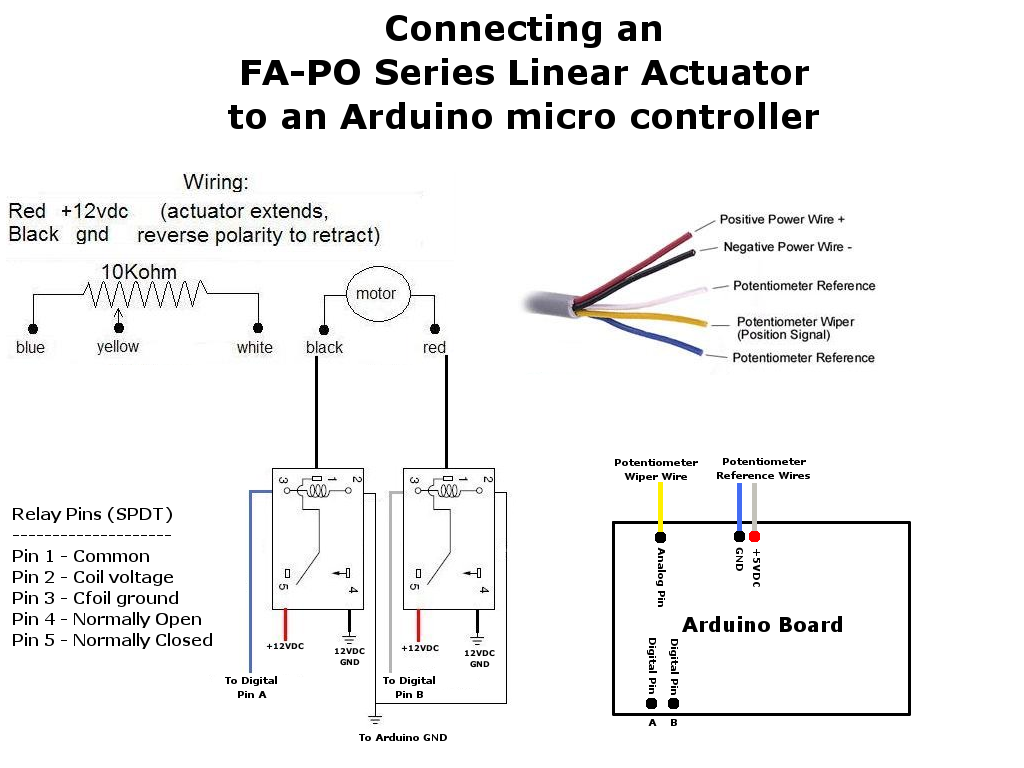
Next, you will want to start getting out the wires. The red and black wires are your power wires, and the yellow wire is your wiper for your pot, the grey and the blue wires are your reference wires for the pot. (See below)



Connect your red and black wires to the sabertooth motor controllers. The wiring here won’t damage your board, but it will reverse the polarity, making it go the opposite direction. Connect your actuator to the M1A, M1B, M2A, or M2B ports. You want your actuator to be connected to either M1, or M2, not both. See below.



Once your actuator is connected to your motor controller, connect it to the Arduino. You will need to use the analog ports. See below for specifics on the yellow, blue and grey wires.

****  
The actuators should now be fully connected. You will want to keep track of the analog pins used as well as which motor ports and motor controllers each one is connected too. This will make it easier when programming.

My actuators were connected to the following pins:

int Actuator1 = A8;

int Actuator2 = A9;

int Actuator3 = A10;

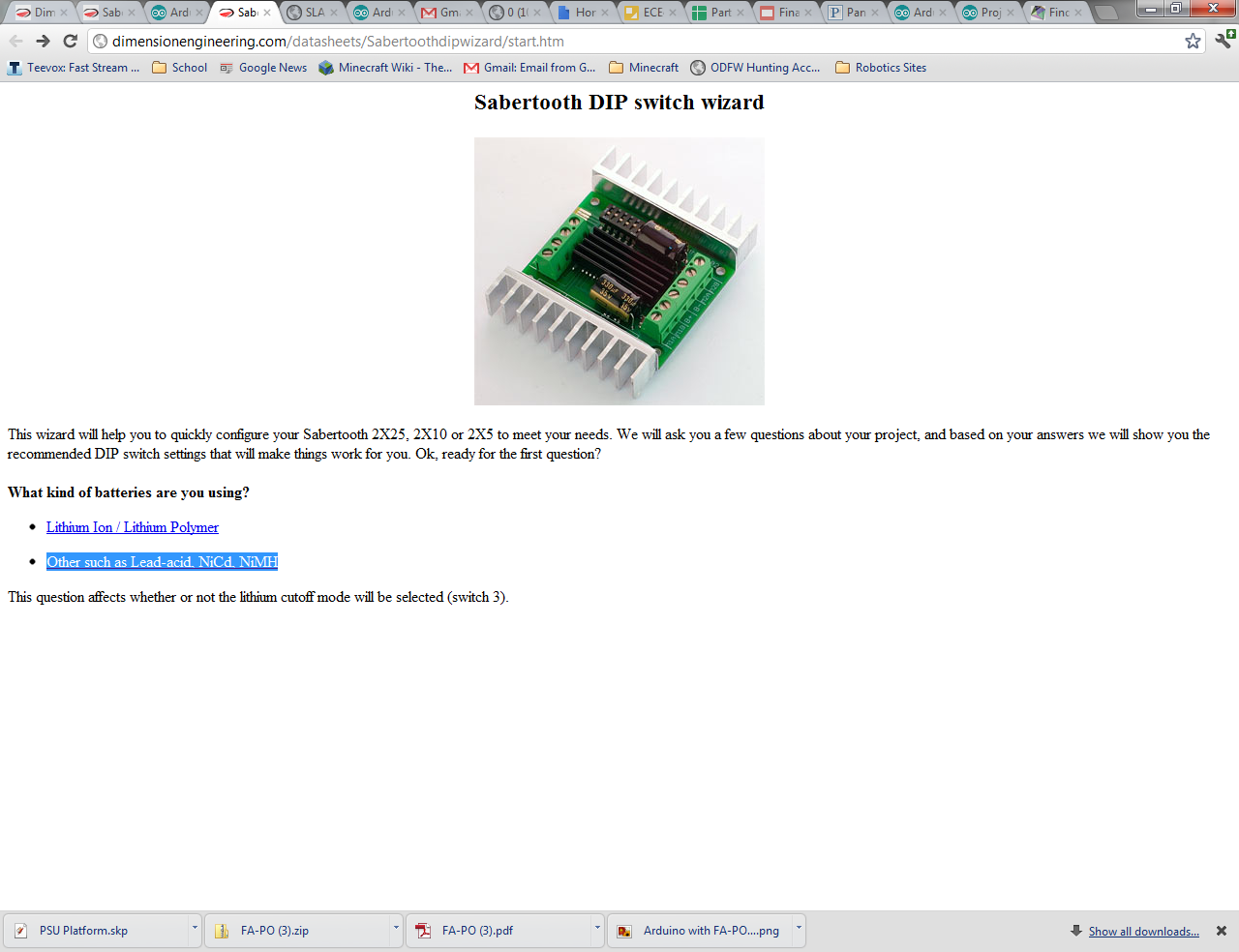
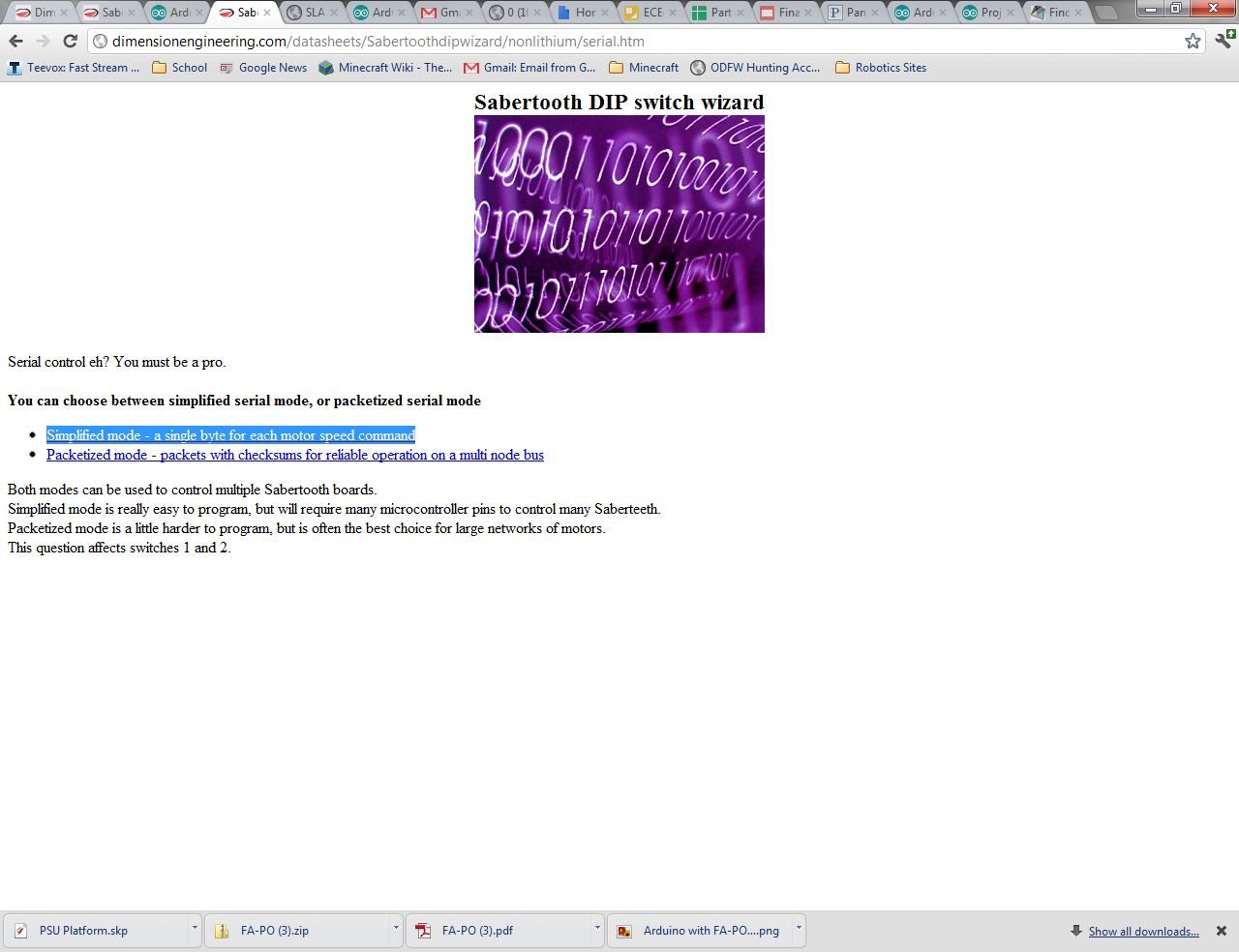
int Actuator4 = A11;

I used the board ground, the 5v coming off the motor controller for the 5v reference wire.

**Motor Controllers**

The motor controllers need to be connected serially, and the first thing you need to do is visit the dip switch wizard to determine what your dip switch positions should be (<http://dimensionengineering.com/datasheets/Sabertoothdipwizard/start.htm>).

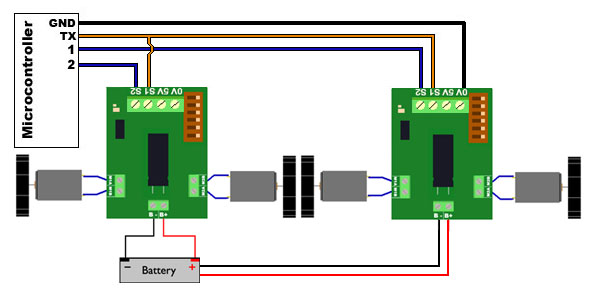
It will take you through the following steps, as seen below (the highlighted were selected):

1. The first page will take you to battery selection. We are planning on using Lead Acid batteries, and selected accordingly. 
2. Our communications method was to be Serial.
3. We chose Simplified Serial mode for ease of use. 
4. We wanted to be able to select multiple motor controllers, so we entered more than one. Per serial line, the Arduino can support up to 8 motor controllers. That’s a lot of motors!
5. We chose a 9600 baud rate, and you want yours to match that dip configuration unless you want a higher BAUD. 

Once your dip switch selection is complete, you will want to look into wiring the board with your arduino. We used the following configuration:

|  |  |
| --- | --- |
| Arduino Pin | Where too |
| TX 18 pin | Serial transmit pin to motor controller |
| RX 19 pin | Left floating, not needed |
| Digital I/O pin 52 | Used to select Motor controller 1 |
| Digital I/O pin 53 | Used to select Motor controller 2 |

See the picture below on how to hook up the motor controllers, it is important that the ground only be connected to motor controller 1 (hooked up to select pin 1).



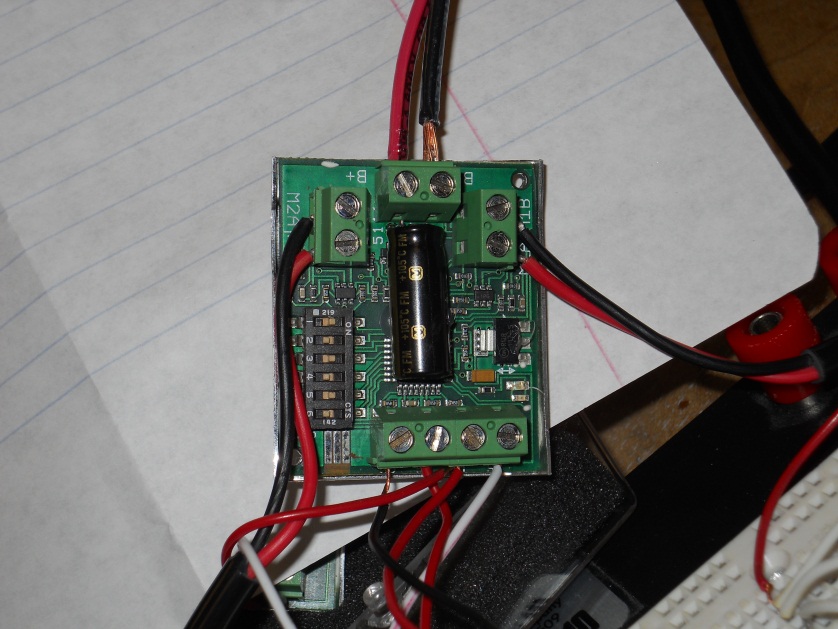
The complete pin setup for the motor controllers can be seen below:

|  |  |  |
| --- | --- | --- |
| Pins | I/O | Use |
| 0v | I | 0v Signal ground, to be connected to ground of device sending signals. |
| 5v | O | 5v output for pots |
| S1 | I | Input signal |
| S2 | I | Input signal |
| B+ | Input | Battery+ terminal |
| B- | Input | Batter- terminal |
| M1B | O | Motor driving outputs, can reverse to change direction |
| M1A | O | Motor driving outputs, can reverse to change direction |
| M2B | O | Motor driving outputs, can reverse to change direction |
| M2A | O | Motor driving outputs, can reverse to change direction |

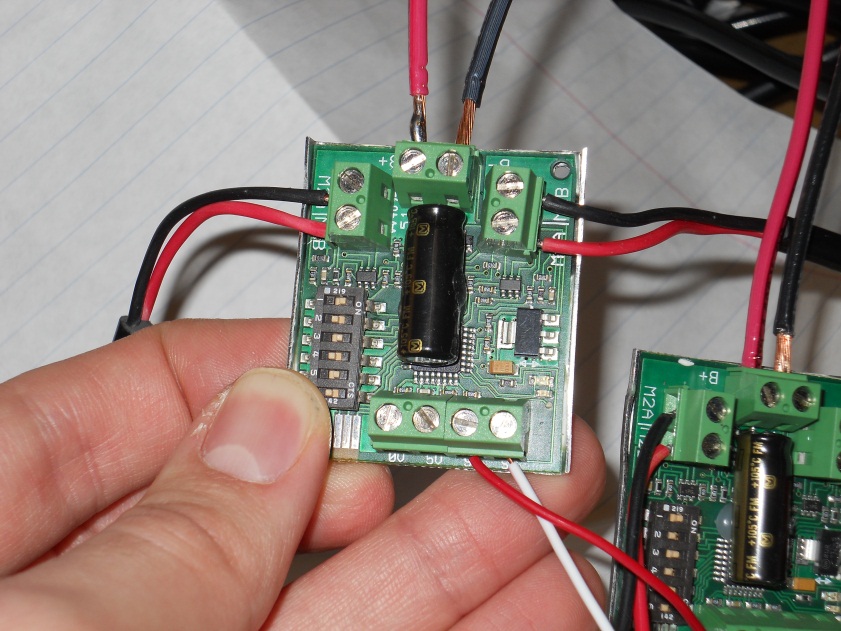
**Complete Wiring Setups**

When you are done, the wiring should look roughly as shown below:

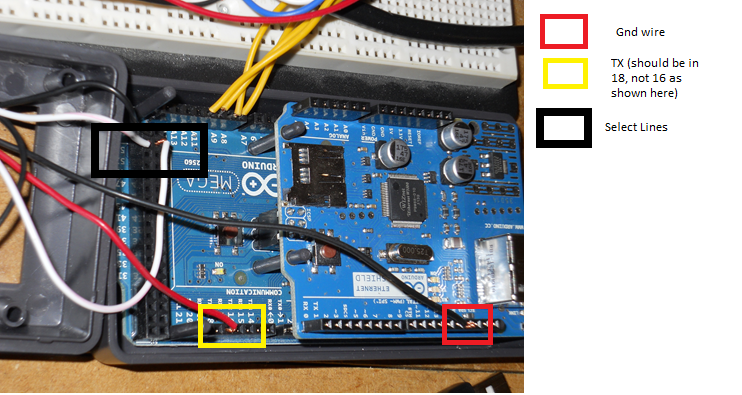
Motor controller 1:

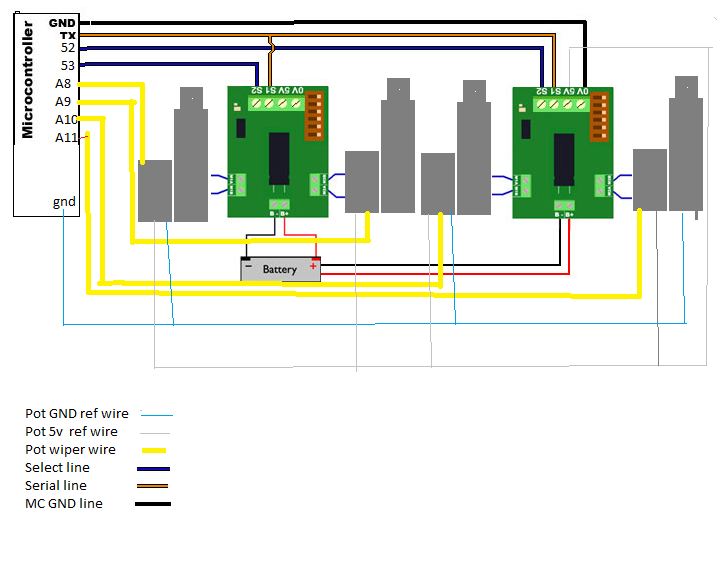


Motor controller 2

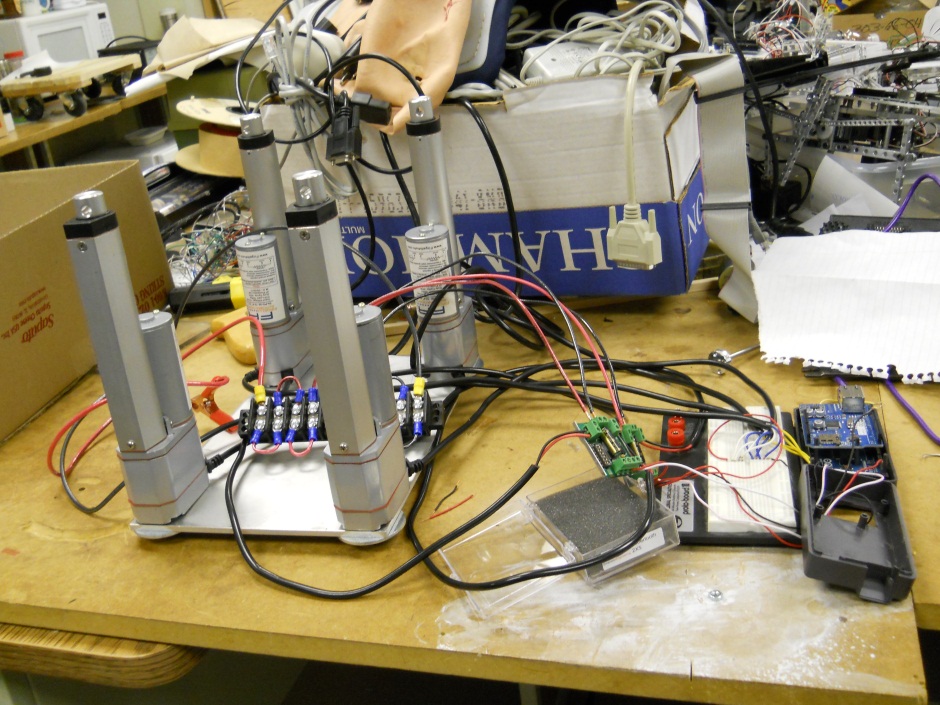


Arduino connection





Please note, the only wires spliced together are the serial TX pins, everything, and optionally the gnd and voltage pins. Below is a picture of the real life circuit, including the terminal block (described below):



**Terminal Block for Power**

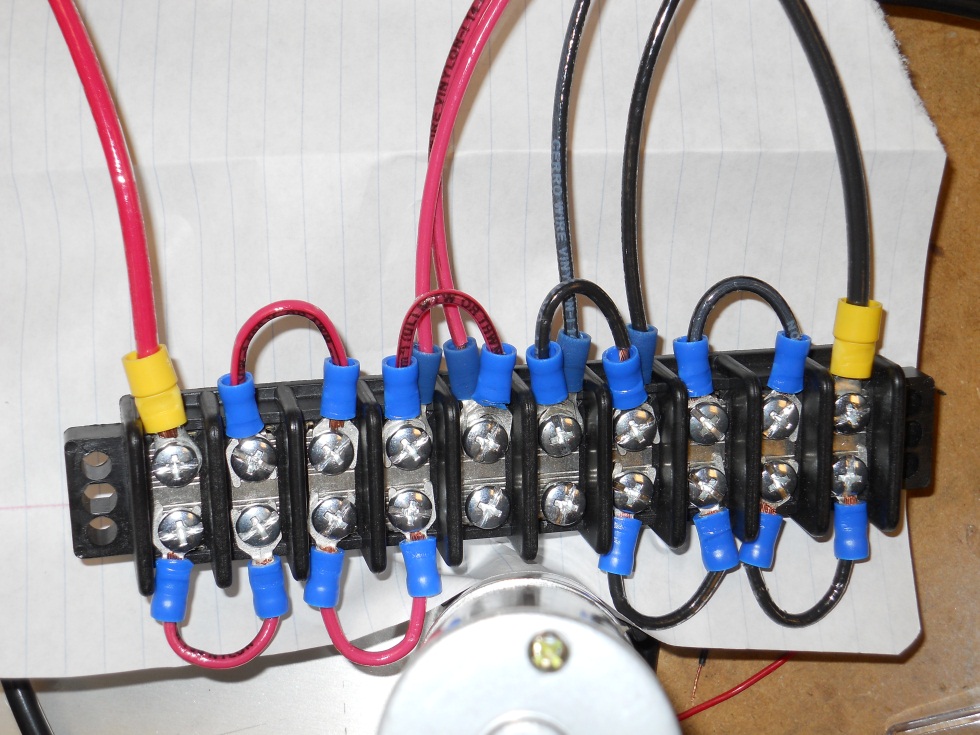
Equipment needed:

1. Yellow and blue terminal leads
2. Terminal crimping tool 
3. 10 and 14 gauge wire, red and black
4. 1 terminal block rated for at least 30 amps 

Once you have all your tools laid out, start out by stripping your wire to have bare leads that can easily insert into the terminals without showing any bare wire. Start with equal lengths of the 10 gauge wire. The yellow leads are needed for the 10 gauge wire, and the blue for the 14. Attached the yellow terminal leads to the wire by inserting the wire in the leads, the crimping the metal part to the wire is firmly in place. Once you have done this, remove the screws on opposite ends of your terminal block:



Next you are ready to create your jumpers. We want our block to have ½ red, and ½ back wiring for easy connection to other circuits. Each wire you get is a little different, so you will have to use your judgment on jumper length. Mine ended up being about 1 inch. Use the 14 gauge wire and attach blue terminal leads to both ends. Remove the screws on the terminal you want to bridge, and place the leads under the screw. Depending on the type of lead you have, you may not need to remove the screws completely. Make enough jumpers, and attach them until the middle two terminals are connected to either red or black. **Do not create a jumper between the middle two terminals**. This will short your circuit. In the end, your total circuit should look something like this:



Next, take your battery clamps, and solder them onto your battery connectors. These will eventually be replaced with a more permanent solution, and will be used for their ease of changing until a permanent battery is selected.

**Arduino**

Once everything is wired up, you are ready to start programming your arduino. It is recommended that you start without the feedback at first, so you can troubleshoot the circuit as needed. The motor controllers have a range of values in order to move or stop the motors they control. They are important, and can be found in the appendix in detail, or below:

|  |  |
| --- | --- |
| Range | Reaction |
| 0 | Full stop on both motors |
| 64 | Stop on M1 (A/B) |
| 65-127 | 127 is full forward on M1(A/B), all else increments from stop |
| 1-63 | 1 is full backwards on M1 (A/B), all else increments from stop |
| 192 | Stop on M2 (A/B) |
| 193-255 | 127 is full forward on M2(A/B), all else increments from stop |
| 128-191 | 1 is full backwards on M2 (A/B), all else increments from stop |

The actuators come equipped with limit switches that will not allow you to over extend or over retract your actuators. This allows you to experiment! Start off with a simple forward command. Remember to watch the LED’s on your Saberteeth. Blue means things are going according to plan, red means there is an error.

Use the following code the test out your circuit setup. This code only utilizes one Sabertooth 2x5 motor controller.

First, setup the Serial pins and libraries. You can do this by simply using the serial library, as seen below:

#include <SoftwareSerial.h>

#define SABER\_TX\_PIN 18 // this sets up the tx pin 18 on the arduino

#define SABER\_RX\_PIN 19 // this sets up the rx pin 19 on the arduino

#define SABER\_BAUDRATE 9600 // sets up the BAUD rate

Next you will want to setup your constants for the ranges of the motors:

// Simplified serial Limits for each motor

#define SABER\_MOTOR1\_FULL\_FORWARD 127

#define SABER\_MOTOR1\_FULL\_REVERSE 1

#define SABER\_MOTOR2\_FULL\_FORWARD 255

#define SABER\_MOTOR2\_FULL\_REVERSE 128

// Motor level to send when issuing the full stop command

#define SABER\_ALL\_STOP 0

You now need to actually setup your software serial:

SoftwareSerial SaberSerial = SoftwareSerial( SABER\_RX\_PIN,

SABER\_TX\_PIN );

void initSabertooth( void )

{

// Init software UART to communicate

// with the Sabertooth 2x5

pinMode( SABER\_TX\_PIN, OUTPUT );

SaberSerial.begin( SABER\_BAUDRATE );

// 2 second time delay for the Sabertooth to init

delay( 2000 );

// Send full stop command

setAction( SABER\_ALL\_STOP );

}

The above code initializes the sabertooth. The first part sets up the software serial variable, specifying it’s receive and transmit pin, in that order. The next function sets the pin modes and initializes the SaberSerial connection. The motor controllers needs a minimum of 50 us, and we gave it 2 seconds to play it safe. We then make sure everything is stopped.

Next, we have our action function. This function will take in a range from -100 to 100. -100 is a full reverse command, and 100 is a full forward. It utilizes the map () function, which takes a variable within certain ranges(original range) and converts it to an equivalent value within another specified range (range 2). Map(numberpassed in, -100, 100, Full reverse constant (1), Full forward constant (127)). When the function passes in certain values, it can convert them into speed for our motors to go:

void setAction( signed char cNewMotorSpeed )

{

unsigned char cSpeedVal\_Motor1 = 0;

unsigned char cSpeedVal\_Motor2 = 0;

// Check for full stop command

if( cNewMotorSpeed == 0 )

{

// Send full stop command for both motors

SaberSerial.write( byte(0));

return;

}

// Calculate the speed value for motor 1

if( cNewMotorSpeed >= 100 )

{

cSpeedVal\_Motor1 = SABER\_MOTOR1\_FULL\_FORWARD;

cSpeedVal\_Motor2 = SABER\_MOTOR2\_FULL\_FORWARD;

}

else if( cNewMotorSpeed <= -100 )

{

cSpeedVal\_Motor1 = SABER\_MOTOR1\_FULL\_REVERSE;

cSpeedVal\_Motor2 = SABER\_MOTOR2\_FULL\_REVERSE;

}

The snippet above starts off by looking for the min and max values. If -100 is passed in, then the motors are to go into full reverse. The motor constants are stored into the function variable.

else

{

// Calc motor 1 speed (Final value ranges from 1 to 127)

cSpeedVal\_Motor1 = map( cNewMotorSpeed,

-100,

100,

SABER\_MOTOR1\_FULL\_REVERSE,

SABER\_MOTOR1\_FULL\_FORWARD );

// Calc motor 2 speed (Final value ranges from 128 to 255)

cSpeedVal\_Motor2 = map( cNewMotorSpeed,

-100,

100,

SABER\_MOTOR2\_FULL\_REVERSE,

SABER\_MOTOR2\_FULL\_FORWARD );

}

// Fire the values off to the Sabertooth motor controller

SaberSerial.write(byte(cSpeedVal\_Motor1));

SaberSerial.write(byte(cSpeedVal\_Motor2));

}else

{

// Calc motor 1 speed (Final value ranges from 1 to 127)

cSpeedVal\_Motor1 = map( cNewMotorSpeed,

-100,

100,

SABER\_MOTOR1\_FULL\_REVERSE,

SABER\_MOTOR1\_FULL\_FORWARD );

// Calc motor 2 speed (Final value ranges from 128 to 255)

cSpeedVal\_Motor2 = map( cNewMotorSpeed,

-100,

100,

SABER\_MOTOR2\_FULL\_REVERSE,

SABER\_MOTOR2\_FULL\_FORWARD );

}

// Fire the values off to the Sabertooth motor controller

SaberSerial.write(byte(cSpeedVal\_Motor1));

SaberSerial.write(byte(cSpeedVal\_Motor2));

}

This second snippet looks for things that aren’t full forward or reverse, and uses the map function to convert them to speeds the SaberTooth controllers can deal with. In more technical terms, the map function looks like:

#### Map(Value, fromLow, fromHigh, toLow, toHigh);

#### Parameters

value: the number to map

fromLow: the lower bound of the value's current range

fromHigh: the upper bound of the value's current range

toLow: the lower bound of the value's target range

toHigh: the upper bound of the value's target range

We are now ready to setup our main loop, and implement our functions and serial communication:

void setup( )

{

initSabertooth( );

}

void loop ( )

{

//0 stops all actuators

//100 is full forward

setAction( 100 );

}

Once you troubleshoot the above code, you should be ready to start controlling your actuators. We will be utilizing the same commands above, and also using our analog input pins. The setup will look very similar to the above, with the exception of using analog pins, and motor controller select lines, as well as analog inputs.

// Digital pin 18 is the serial transmit pin to the

// Sabertooth 2x5

#define SABER\_TX\_PIN 18

// NOT USED (but still init'd)

// Digital pin 19 is the serial receive pin from the

// Sabertooth 2x5

#define SABER\_RX\_PIN 19

// Set to 9600 through Sabertooth dip switches

#define SABER\_BAUDRATE 9600

// Simplified serial Limits for each motor

//64 is soft stop

#define SABER\_MOTOR1\_FULL\_FORWARD 127

#define SABER\_MOTOR1\_FULL\_REVERSE 1

//192 is soft stop

#define SABER\_MOTOR2\_FULL\_FORWARD 255

#define SABER\_MOTOR2\_FULL\_REVERSE 128

// Motor level to send when issuing the full stop command

#define SABER\_ALL\_STOP 0

SoftwareSerial SaberSerial = SoftwareSerial( SABER\_RX\_PIN,

SABER\_TX\_PIN );

We next add a new bit of code, which will initialize the analog pins for the actuators. The aliases for these pins match directly to those found on the board.

int Actuator1 = A8; // selects the input pins for the actuator pt

int Actuator2 = A9;

int Actuator3 = A10;

int Actuator4 = A11;

// places to store actuator loaction values

int Actuator1Value = 0;

int Actuator2Value = 0;

int Actuator3Value = 0;

int Actuator4Value = 0;

And then, we must finally set up the motor controller select lines. With both of these high, whatever command you send will go to both motor controllers. See below:

|  |  |  |  |
| --- | --- | --- | --- |
| Select 1 (pin 52) | Select 2(pin 53) | Motor controller 1 | Motor controller 2 |
| Low | Low | Off | Off |
| Low | High | Off | On |
| High | Low | On | Off |
| High | High | On | on |

Setup of motor controller select lines in code:

//setting up select pins. 52 in motor controller 1,

// 53 is motor controller 2

int MC1 = 52;

int MC2 = 53;

Next we have to initialize like we did above, but we a few more commands. We need to use the pinMode() command to set our digital IO pins MC1 and MC2 to an output. Pins can be set as an input using this command as well.

void initSabertooth( void )

{

// Init software UART to communicate

// with the Sabertooth 2x5

pinMode( SABER\_TX\_PIN, OUTPUT );

SaberSerial.begin( SABER\_BAUDRATE );

// 2 second time delay for the Sabertooth to init

delay( 2000 );

// Send full stop command

SaberSerial.write(byte(SABER\_ALL\_STOP));

//setting up pinmodes for select pins

pinMode(MC1, OUTPUT);

pinMode(MC2, OUTPUT);

}

Next we call out setup function:

void setup( )

{

initSabertooth( );

}

Now we can execute our main loop. This code will simply lift all the actuators into the neutral position, which is about 2 inches extended. We will utilize the analogRead() command to read the values off of the potentiometers as we wired above.

void loop ( )

{

// Reading in values for actuator position, lower is more extended

Actuator1Value = analogRead(Actuator1);

Actuator2Value = analogRead(Actuator2);

Actuator3Value = analogRead(Actuator3);

Actuator4Value = analogRead(Actuator4);

//64 will stop the motor 1

//192 is stop for motor 2

//LOW disables both motor controllers

//HIGH enables both motor controllers

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

//All MC's active to set a nuetral position

if (Actuator1Value > 700)

{

digitalWrite(MC1, HIGH);

digitalWrite(MC2, LOW);

SaberSerial.write(byte(SABER\_MOTOR2\_FULL\_FORWARD));

//delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

else //writing to act 1 to stop

{

digitalWrite(MC1, HIGH);

digitalWrite(MC2, LOW);

SaberSerial.write(byte(192));

// delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

if (Actuator2Value > 700)

{

digitalWrite(MC1, HIGH);

digitalWrite(MC2, LOW);

SaberSerial.write(byte(SABER\_MOTOR1\_FULL\_FORWARD));

//delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

else // writing to act 2 to stop

{

digitalWrite(MC1, HIGH);

digitalWrite(MC2, LOW);

SaberSerial.write(byte(64));

// delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

if (Actuator3Value > 700)

{

digitalWrite(MC1, LOW);

digitalWrite(MC2, HIGH);

SaberSerial.write(byte(SABER\_MOTOR1\_FULL\_FORWARD));

//delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

else // writing to act 3 to soft stop

{

digitalWrite(MC1, LOW);

digitalWrite(MC2, HIGH);

SaberSerial.write(byte(64));

//delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

if (Actuator4Value > 700)

{

digitalWrite(MC1, LOW);

digitalWrite(MC2, HIGH);

SaberSerial.write(byte(SABER\_MOTOR2\_FULL\_FORWARD));

// delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

else

{ //writing to Act 4 to soft stop

digitalWrite(MC1, LOW);

digitalWrite(MC2, HIGH);

SaberSerial.write(byte(192));

// delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

}

One of the struggles that occurred while coding this was understanding how the readings came off of the potentiometer. It will read a value of 1023 when it is fully retracted, and approach zero as it extends. It also appears to not be linear. There was no documentation to be found on how the potentiometers acted. Trial and error provided the numbers you saw in the code above. A reading of 700 is roughly 2 inches from the bottom. Each actuator will be a bit different, and will require a bit of calibration.

**Future Development**

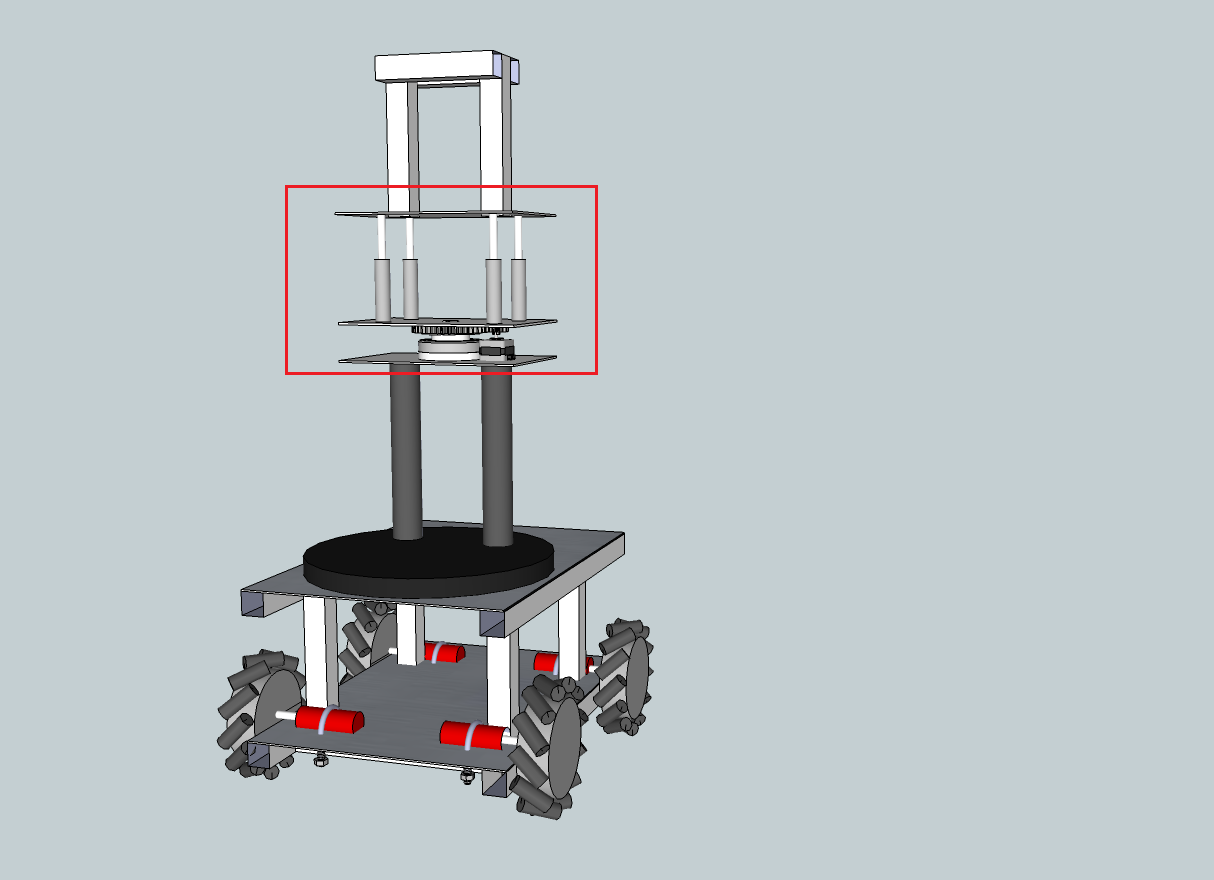
There were many difficulties encountered during the development of this project. Some of these difficulties will require additional time and resources.

**Battery**

The required power for both the base as well as the body joint would entail about 30A per hour worst case. LiPo batteries would meet the weight requirements, but be really expensive. An 11.1V 8AH battery would cost 254.99. We would need four of these to cover our needs. We next moved onto Lead acid batteries. These were tricky because they use strange ratings, or look at them differently. When you see a Lead acid battery that says 30 AH, what they really mean is that they look at them over a period of 10-20 hours, and take the average. Over a period of an hour, it is usually about 65% of that value. As a result, we were force to look at batteries that could power 70AH. These proved to be heavy. The next class will need to evaluate project needs, and find the most elegant and cost efficient solutions for power.

**Twisting**

As seen in the guidebot above in the introduction, a twisting motion can be implemented. This would allow for the waist not only bend in any direction. Currently the waist can lean side to side at most 21.9 degrees, and lean forward/backwards at most 40.51 degrees. This would allow a twisting motion, giving it more degrees of freedom. This can be done with a stepper motor, and feedback using the arduino board. Another motor controller will have to be purchased as well.

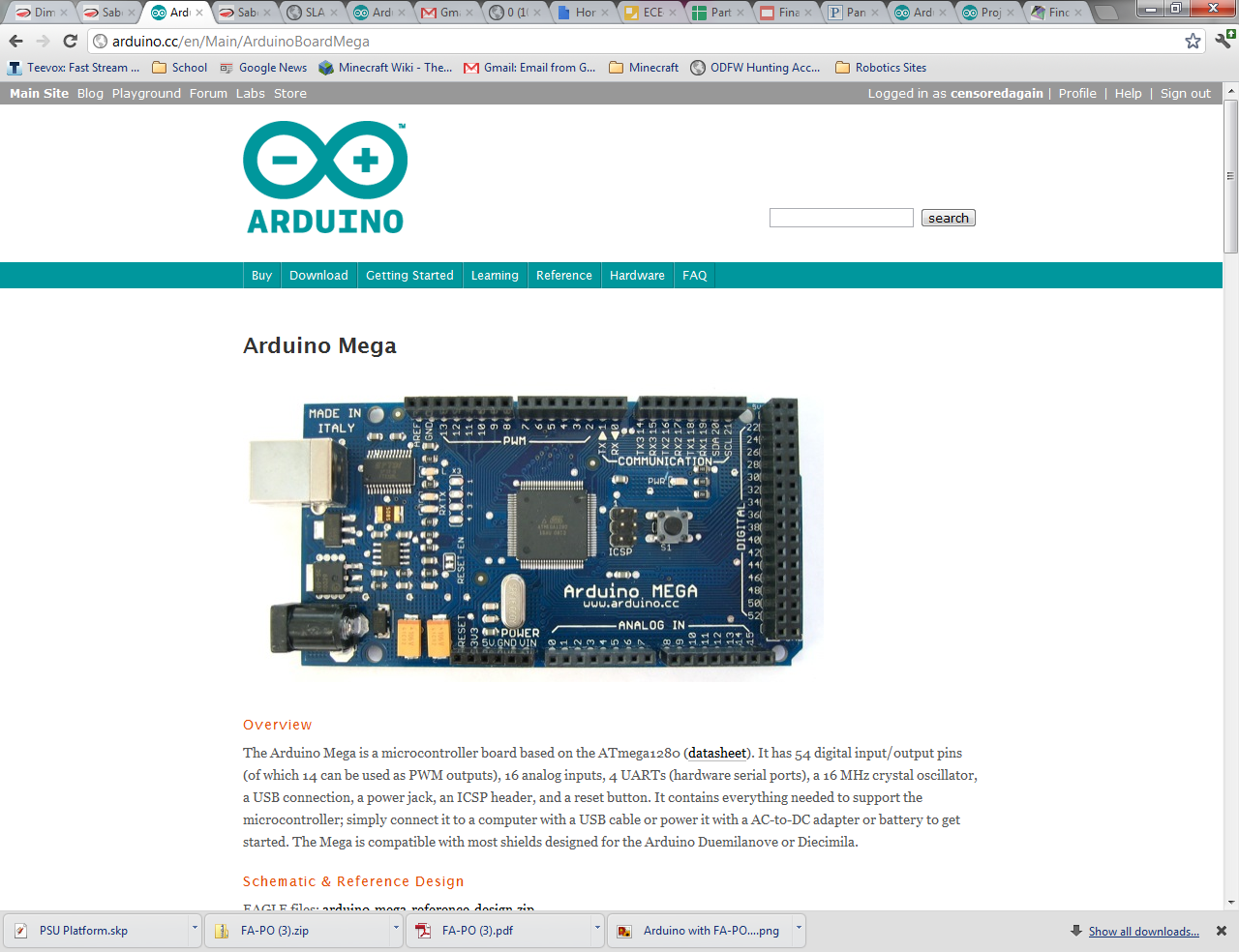


**Mechanical Construction**

The future developers of the project will need to build and connect this part with the rest of the parts as seen in the introduction above. They will also need to build adequate support for the other modules, as they are still developing their portions. They will also need to work on Ethernet control using the Ethernet shield to receive commands from the router. The next class will also need to build the top plate in such a way that it can flex and provide enough support to lift 75 LBS. Currently this is not possible, as the actuators did not have enough of a lip to rest on. You cannot simply attach the plate directly to the actuator without the plate causing damage, as it cannot flex the needed directions to move.

**Appendix**

**Arduino Board**

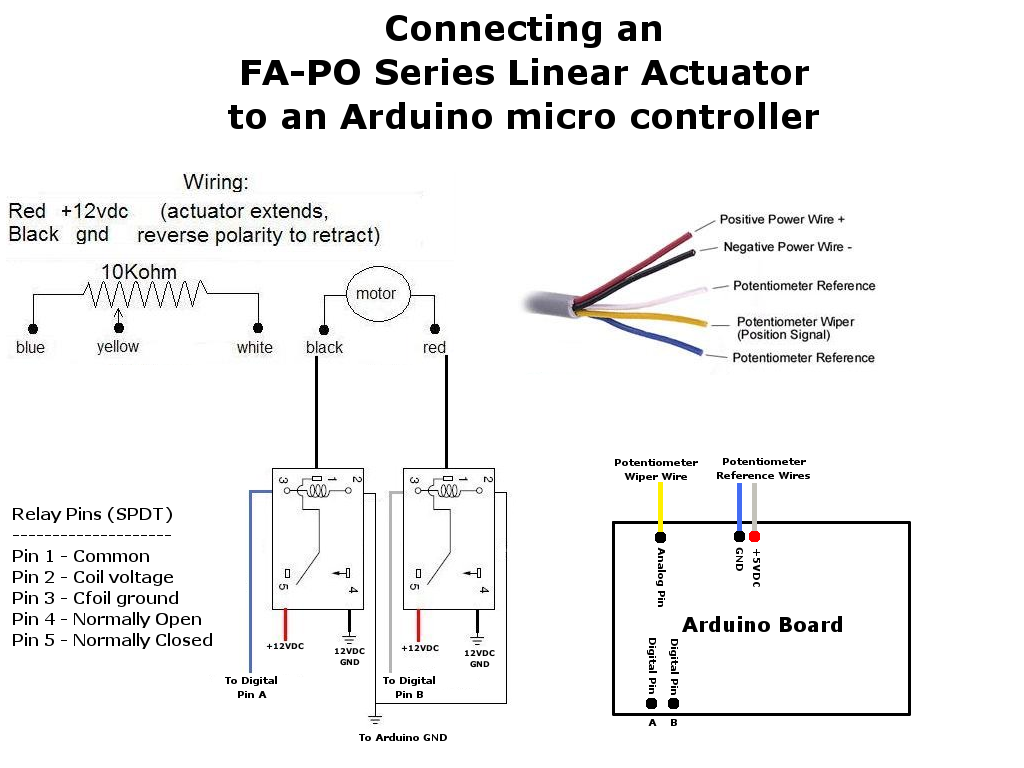


**Specs:**

|  |  |
| --- | --- |
| Microcontroller | ATmega2560 |
| Operating Voltage | 5V |
| Input Voltage (recommended) | 7-12V |
| Input Voltage (limits) | 6-20V |
| Digital I/O Pins | 54 (of which 14 provide PWM output) |
| Analog Input Pins | 16 |
| DC Current per I/O Pin | 40 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 256 KB of which 8 KB used by bootloader |
| SRAM | 8 KB |
| EEPROM | 4 KB |
| Clock Speed | 16 MHz |

**Actuators**

**Pot Hookup:**

****

**Spec Sheet:**

PRODUCT INFORMATION



**Firgelli Automations, 1465 Slater Road, Ferndale, WA 98248**

**Tel:** 1 866-226-0465

**Fax:** 1 866 226-1649Email  [sales@firgelliauto.co](mailto:sales@firgelliauto.com)m  [www.FirgelliAuto.co](http://www.FirgelliAuto.com/)m



* Low noise design
* Enhanced corrosion resistance
* Aluminum outer and inner tube
* Zinc alloy housing
* Powder metal gears
* Lubrication for longer life
* Small compact Design
* Low Price

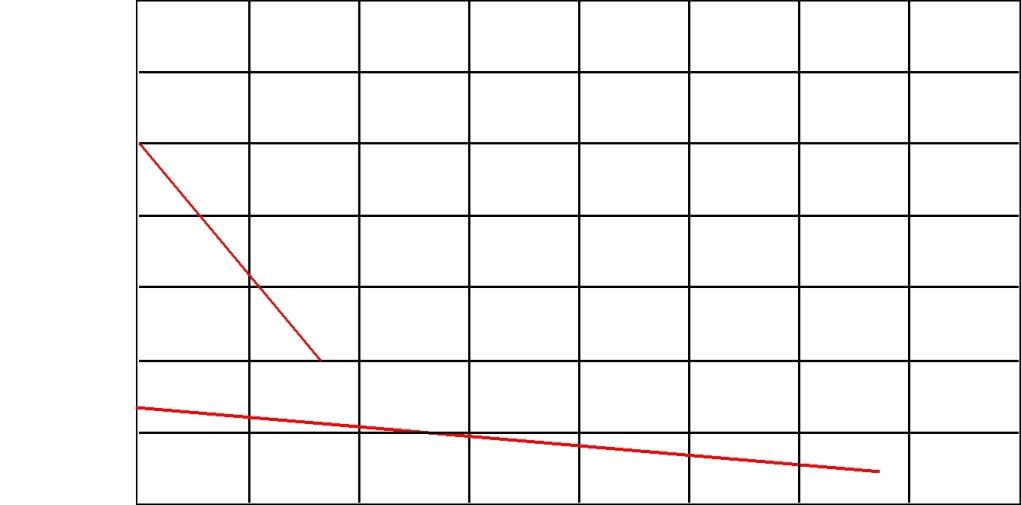
**Specifications**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **FA-PO-35-12-xx”** |  |  | **FA-PO-150-12-xx”** |  |
|  | **(High Speed)** |  |  | **(Standard Force)** |  |
| Input Voltage |  | 12 VDC | | |  |
| Load Capacity | 35 lbs |  |  | 150lbs |  |
| Static Load | 2 x max load capacity | | | |  |
| Stroke Length |  | 2” to 12” | | |  |
| Speed at No Load | 2”/sec (50mm/s) |  |  | ½”/sec (12mm/sec) |  |
| Feedback | 10K ohm 3 wires Potentiometer | | | |  |
| Clevis Ends | 6.3mm diameter | | | |  |
| Screw | ACME screw | | | |  |
| Gear Ratio | 5:1 |  |  | 20:1 |  |
| Duty Cycle | 20% *(2 minutes continuous use in 10 minute period)* | | | |  |
| Operation Temperature | -26ºC~65ºC (-15ºF~150ºF) | | | |  |
| Range |  |  | | |  |
| Limit Switch | Built-in (Factory Preset) Not movable | | | |  |
| IP Grade | IP54 (dust and splash resistant) | | | |  |

**PRODUCT INFORMATION**

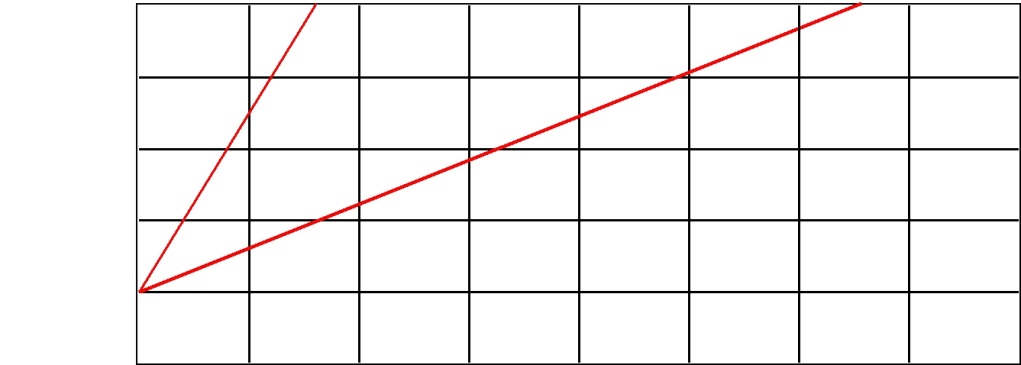
**LOAD/SPEED**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 12VDC |  |  | **Speed V.S. Load** | | |  |  |  |  |
|  | 70 |  |  |  |  |  |  |  |  |  |
|  | 60 | 5:1 |  |  |  |  |  |  |  |  |
| **Speed(mm/S)** |  |  |  |  |  |  |  |  |  |
| 50 |  |  |  |  |  |  |  |  |  |
| 40 |  |  |  |  |  |  |  |  |  |
| 30 |  |  |  |  |  |  |  |  |  |
| 20 |  |  |  | 20:1 | |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  | 10 |  |  |  |  |  |  |  |  |  |
|  | 0 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 |  |
|  |  |  |  |  | **Load(N)** |  |  |  |  |  |



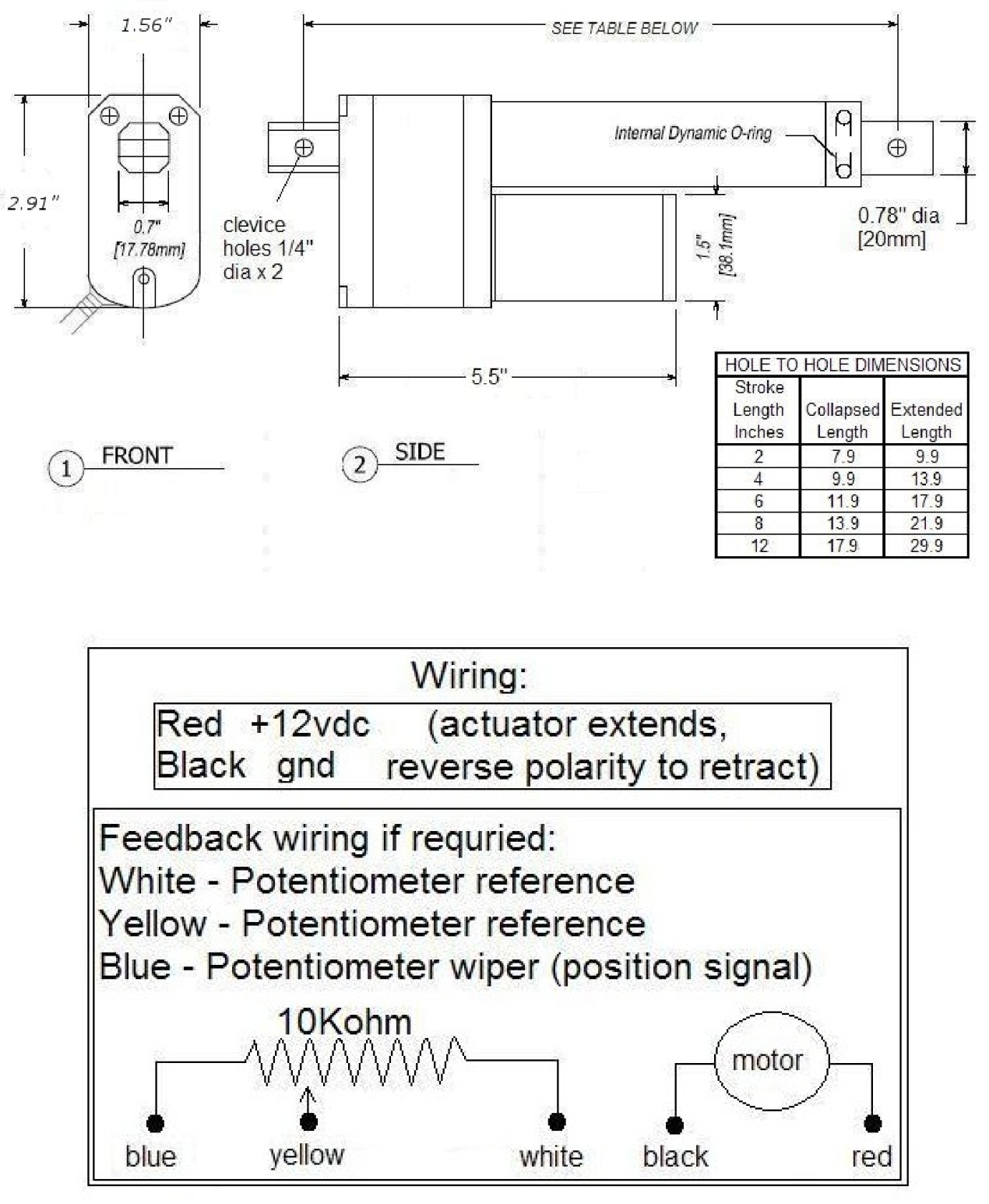
**LOAD/CURRENT**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 12VDC | **Current V.S. Load** | | | |  |  |  |  |  |
|  | 5.0 |  |  |  |  |  |  |  |  |  |
| **Current(A)** | 4.0 | 5:1 |  | 20:1 |  |  |  |  |  |  |
| 3.0 |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
| 2.0 |  |  |  |  |  |  |  |  |  |
| 1.0 |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | 0 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 |  |

**Load (N)**

**PRODUCT INFORMATION**

**Dimensions:**



 This model actuator uses Firgelli Automations MB1 Brackets on both ends of the actuator.

 Visit <http://www.firgelliauto.com/product_info.php?cPath=70&products_id=54> for more details

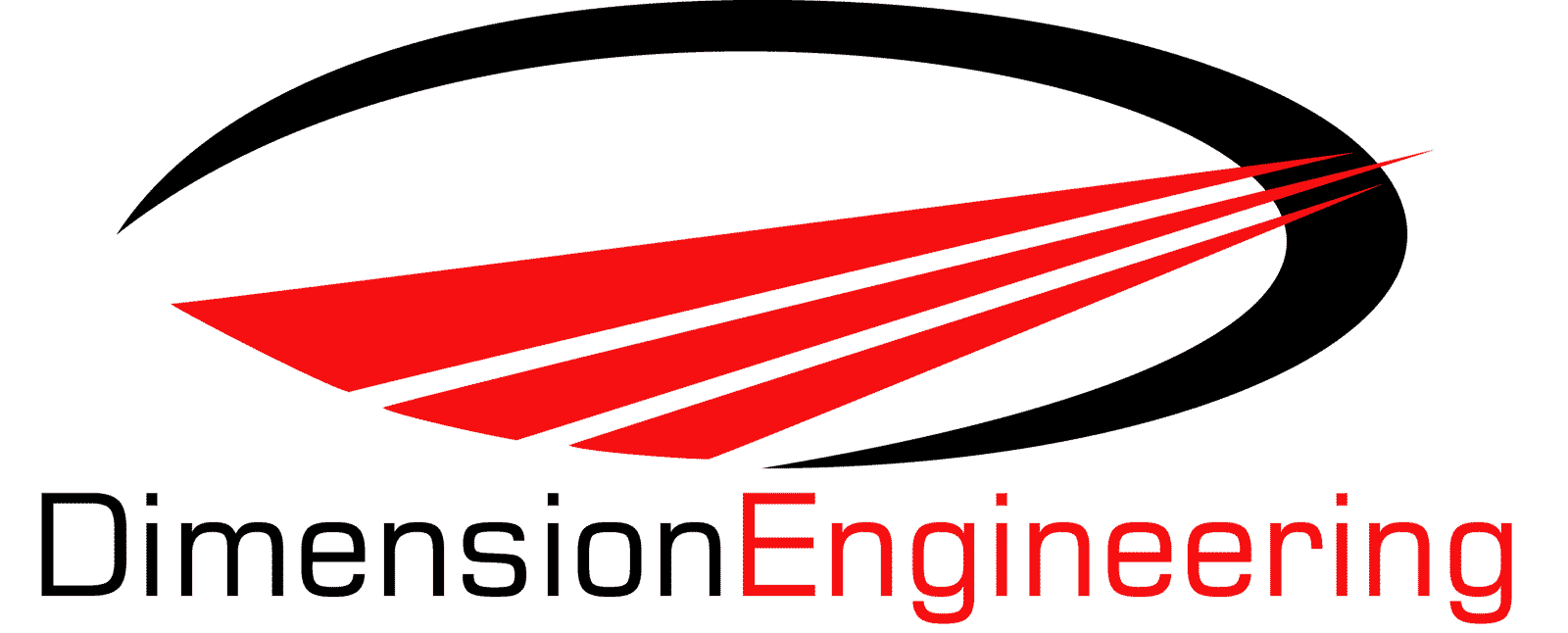
Please visit our website for latest prices and to purchase  [www.FirgelliAuto.co](http://www.FirgelliAuto.com/)m

**Motor Controllers**

**Dip Switch Wizard**

<http://dimensionengineering.com/datasheets/Sabertoothdipwizard/start.htm>

**Motor Controller Spec Sheets**

****

Sabertooth 2x5 User’s Guide

February 2007

|  |
| --- |
| Main-photo |

**Input voltage:** 6-18V nominal, 20V absolute max.

**Output Current:** Up to 5A continuous per channel. Peak loads may be up to 10A per channel for a few seconds. Current limit prevents drawing more than 10A per channel.

**Recommended power sources are:**

* 5 to 12 cells NiMH or NiCd
* 2s to 4s lithium ion or lithium polymer. Sabertooth motor drivers have a lithium battery mode to prevent cell damage due to over-discharge of lithium battery packs.
* 6v to 12v lead acid
* 6v to 18v power supply (when in parallel with a suitable battery).

**Dimensions:**

|  |
| --- |
| Size: 1.6” x 1.8” x .5” 40 x 45 x 13mm  Weight: .7oz/19g |

Features

**Mixed and independent options:**

Sabertooth features mixed modes designed especially for differential drive robots, where two motors provide both steering and propulsion. It also has independent options in all operating modes. This is useful for if you have two motors to control, but they aren’t necessarily being used to drive a differential drive robot. The motors do not need to be matched or even similar, as long as they both are within Sabertooth’s operating limits.

**Synchronous regenerative drive:**

Going one step farther than just regenerative braking, a Sabertooth motor driver will return power to the battery any time a deceleration or motor reversal is commanded. This can lead to dramatic improvements in run time for systems that stop or reverse often, like a placement robot or a vehicle driving on hilly terrain. This drive scheme also saves power by returning the inductive energy stored in the motor windings to the battery each switching cycle, instead of burning it as heat in the motor windings. This makes part-throttle operation very efficient.

**Ultra-sonic switching frequency:**

Sabertooth 2x5 features a PWM frequency of 32kHz, which is well above the maximum frequency of human hearing. Unlike some other motor drivers, there is no annoying whine when the motor is on, even at low power levels.

**Thermal and overcurrent protection:**

Sabertooth features a temperature sensor and overcurrent sensing. It will protect itself from failure due to overheating, overloading and short circuits.

**Easy mounting and setup:**

Sabertooth has screw terminals for all inputs and outputs. There are two mounting holes, which accept 2-56 screws. Mounting hardware is included. All operating modes and options are set with DIP switches – there are no jumpers to struggle with or lose. No soldering is required.

**Compact Size:**

Sabertooth utilizes surface mount construction to provide the most power from a compact package. Its small size and light weight means you have more space for cargo, batteries, or can make your robot smaller and more nimble than the competition.

**Carefree reversing:**

Unlike some other motor drivers, there is no need for the Sabertooth to stop before being commanded to reverse. You can go from full forward immediately to full reverse or vice versa. Braking and acceleration are proportional to the amount of reversal commanded, so gentle or rapid reversing is possible.

**Many operating modes:**

With analog, R/C and serial input modes, as well as dozens of operating options, the Sabertooth has the flexibility to be used over and over, even as your projects grow more sophisticated. Yet it is simple enough to use for your first robot project.

# Hooking up the Sabertooth motor driver

All connections to the Sabertooth are done with screw terminals. This makes it easy to set up and reconfigure your project. If you’ve never used screw terminal connections before, here is a quick overview.

|  |  |
| --- | --- |
| Wirestrip | openingterminal |
| **Step 1:** Strip the wire which you are using approximately ¼” The wires may be 14 gauge to 30 gauge | **Step 2:** With a small screwdriver, turn the top screw counter-clockwise until it stops gently. |
|  |  |
| insertingwire | screwinginwire |
| **Step 3:** Insert the stripped portion of the wire into the opening in the screw terminal | **Step 4:** Turn the top screw clockwise until you encounter resistance, then tighten the screw firmly. Pull on the wire gently to ensure that it is secured. |

|  |
| --- |
| motorsconnected |
| The motors connect to terminals M1A/B and M2A/B |

Motor1 Terminals

Motor 1 is connected to terminals M1A and M1B as shown at right. If the motor runs in the opposite way that you want, you may reverse the motor wires to reverse rotation.

Motor 2 is connected to terminals M2A and M2B

|  |
| --- |
| pot-connected |
| The input signals connect to terminal S1 and/or S2 |

Signal Input Terminals

S1 and S2

The input signals that control the Sabertooth are connected to terminals S1 and S2. If you are running in analog mode, it is important to have both the signal inputs connected before applying power to the device. Otherwise, the motors may start unexpectedly.

|  |
| --- |
| battery-pack |
| The battery connects to terminals B+ and B- |

Battery Terminals

B+ and B-

The battery or power supply is connected to terminals B- and B+. B- connects to the negative side of the battery (usually black.) B+ connects to the positive side of the battery (usually red or yellow.) It is usually best to connect the battery through a connector instead of directly to the motor driver. This makes it easy to unplug the battery for charging, and prevents plugging in the battery backwards.

**Warning! Be very careful to wire and plug in the battery and connector correctly. Connecting the battery backwards will destroy the Sabertooth and will void the warranty**.

|  |
| --- |
| everything-connected |
| The 5V terminal can be used to power small loads, like a potentiometer or a radio receiver. The 0V signal must be connected to the ground of the device generating the input signal. |

Power terminals

0V and 5V

The 0V and 5V connections are used to power and interface to low-power control circuits.

The 5V connection is a 5v power **output**. This is useful for supplying power to low-current devices, such as a potentiometer or a radio receiver. The 5v terminal is capable of supplying 100 milliamps if the source battery is 12.6v or less. If the source battery is greater than 12.6 volts, the 5v terminal is capable of supplying 10 milliamps. If more power is needed, we recommend using a ParkBEC or DESW050 to supply the needed 5V power to the rest of the robot.

The 0V connection is the signal ground for the Sabertooth. In order to receive input signals correctly, it must be connected to the ground of the device sending the signals.

Using the 0V and 5v connections to power a radio receiver in R/C mode and potentiometer in analog mode is shown in Figures 2.1 and 2.2. If you are using multiple Sabertooth 2x5s running from the same radio receiver, only one should have the 5v line connected.

|  |  |
| --- | --- |
| pot-connected | RC-receiver |
| Figure 2.1: Analog input using a potentiometer powered from terminal 5V | Figure 2.2: R/C input using a receiver powered from terminal 5V |

|  |
| --- |
| status-LED |
| Status LED on |

Status and Error LEDs

Sabertooth 2x5 has two indicator LEDs.

The blue LED is used to communicate various information about the current state. In most cases Status1 acts as a power indicator. In R/C mode, it glows dimly if there is no RC link present and brightly if there is an RC link. In lithium mode, the blue LED will blink out the number of lithium cells detected.

The red LED illuminates if the Sabertooth has detected a problem. It will light if the driver has shut down due to a depleted battery or due to overheating or overcurrent. If you are using a NiCd or NiMH battery, and commanding an acceleration causes the motor to jerk and the Error LED to flash on and off, the battery is depleted.

|  |
| --- |
| driverwood |
| Figure 2.3: Mounted to a wood frame using standoffs |

Mounting your Sabertooth 2x5

The Sabertooth is supplied with two mounting holes. These can be used to attach it to your robot. The centers of the mounting holes form a 1.25” x 1.5” rectangle. The holes are .09” inches in diameter. The proper size screw is a 2-56 round head machine or wood screw. two 1/2” long machine screws and nuts are included.

|  |
| --- |
| metal |
| Figure 2.4: Mounted directly to a metal frame |

If your robot or device is constructed from insulating materials such as wood or plastic and you are runnin, it may be necessary to mount the Sabertooth on standoffs to allow air to circulate. This is shown in Figure 2.3

If your robot or device is constructed from metal, it is usually better to attach the bottom heat spreader of the Sabertooth directly to the frame, without standoffs. This will allow your frame to act as a heat sink and will cause the Sabertooth to run cooler. This is shown in Figure 2.4

# Operating Modes Overview

## **Mode 1: Analog Input**

Analog input mode takes one or two analog inputs and uses those to set the speed and direction of the motor. The valid input range is 0v to 5v. This makes the Sabertooth easy control using a potentiometer, the PWM output of a microcontroller (with an RC filter) or an analog circuit. Major uses include joystick or foot-pedal controlled vehicles, speed and direction control for pumps and machines, and analog feedback loops.

## **Mode 2: R/C Input**

R/C input mode takes two standard R/C channels and uses those to set the speed and direction of the motor. There is an optional timeout setting. When timeout is enabled, the motor driver will shut down on loss of signal. This is for safety and to prevent the robot from running away should it encounter interference and should be used if a radio is being used to control the driver. If timeout is disabled, the motor driver will continue to drive at the commanded speed until another command is given. This makes the Sabertooth easy to interface to a Basic Stamp or other low-speed microcontrollers.

**Mode 3: Simplified serial.**

Simplified serial mode uses TTL level RS-232 serial data to set the speed and direction of the motor. This is used to interface the Sabertooth to a PC or microcontroller. If using a PC, a level converter such as a MAX232 chip must be used. The baud rate is set via DIP switches. Commands are single-byte. There is also a Slave Select mode which allows the use of multiple Sabertooth 2x10 from a single microcontroller serial port.

## **Mode 4: Packetized serial**

Packetized serial mode uses TTL level RS-232 serial data to set the speed and direction of the motor. There is a short packet format consisting of an address byte, a command byte, a data byte and a 7 bit checksum. Packetized serial automatically detects the transmitted baud rate based on the first character sent, which must be 170. Address bytes are set via dip switches. Up to 8 Sabertooth motor drivers may be ganged together on a single serial line. This makes packetized serial the preferred method to interface multiple Sabertooths to a PC or laptop. Because Sabertooth uses the same protocol as our SyRen single motor drivers, both can use used together from the same serial master.

|  |
| --- |
| dipswitch00010009 |
| Lithium Cutoff enabled |

Lithium cutoff:

Switch 3 of the DIP switch block selects lithium cutoff. If switch 3 is in the down position as shown the Sabertooth will automatically detect the number of series lithium cells at startup, and set a cutoff voltage of 3.0 volts per cell. The number of detected cells is flashed out on the Status LED. If the number of cells detected is too low, your battery is in a severely discharged state and must be charged before operation. Failure to do so may cause damage to the battery pack. When 3.0V per cell is reached, the Sabertooth will shut down, preventing damage to the battery pack. This is necessary because a lithium battery pack discharged below 3.0v per cell will lose capacity and batteries discharged below 2.0v per cell may not ever recharge. Lithium cutoff mode may also be useful to increase the number of battery cycles you can get when running from a lead acid battery in non-critical applications. Because the system will continue to draw some power, even with the motor shut down, it is important to unplug the battery from the Sabertooth promptly once the cutoff is reached. If the Sabertooth is being run from any other power source, switch 3 should be in the up position.

# Mode 1: Analog Input

Analog input mode is selected by setting switches 1 and 2 to the UP position. Switch 3 should be either up or down, depending on the battery type being used. Inputs S1 and S2 are configured as analog inputs. The output impedance of the signals fed into the inputs should be less than 10k ohms for best results. If you are using a potentiometer to generate the input signals, a 1k, 5k or 10k linear taper pot is recommended. In all cases, an analog voltage of 2.5V corresponds to no movement. Signals above 2.5V will command a forward motion and signals below 2.5V will command a backwards motion.

There are three operating options for analog input. These are selected with switches 4, 5 and 6. All the options can be used independently or in any combination.

|  |
| --- |
| dipswitch00010005 |
| Switch 4: Mixed or independent |

# Switch 4: Mixing Mode

If switch 4 is in the UP position, the Sabertooth 2x5 is in **Mixed** mode. This mode is designed for easy steering of differential-drive vehicles. The analog signal fed into S1 controls the forward/back motion of the vehicle, and the analog signal fed into S2 controls the turning motion of the vehicle. If Switch 4 is in the DOWN position, the Sabertooth 2x5 is in Independent mode. In Independent mode, the signal fed to S1 directly controls Motor 1 (outputs M1A and M1B) and the signal fed to S2 controls Motor 2.

|  |
| --- |
| dipswitch20003 |
| Switch 5: Exponential response |

# Switch 5: Exponential response

If switch 5 is in the DOWN position, the response to input signals will be exponential. This softens control around the zero speed point, which is useful for control of vehicles with fast top speeds or fast max turning rates. If switch 5 is in the UP position, the response is linear.

|  |
| --- |
| dipswitch20002 |
| Switch 6: 4x sensitivity |

# Switch 6: 4x sensitivity

If switch 6 is in the UP position, the input signal range is from 0v to 5v, with a zero point of 2.5v.

If switch 6 is in the DOWN position, 4x sensitivity mode is enabled. In this mode, the input signal range is from 1.875V to 3.125V, with a zero point of 2.5v. This is useful for building analog feedback loops

|  |
| --- |
| PWMfilter |
| Figure 4.1: Filtered PWM |

# Note on using filtered PWM in Analog Mode

If you are using a filtered PWM signal from a microcontroller to generate the analog voltage, an R/C filter with component values 10k ohms and at least .1uf is recommended as shown in **Figure 4.1.** Using a larger value filter capacitor such as 1uf or 10uf will result in smoother motor operation, at a cost of slower transient response. A PWM frequency higher than 1000Hz is recommended.

Mode 2: R/C Input

R/C input mode is used with a standard hobby Radio control transmitter and receiver, or a microcontroller using the same protocol. R/C mode is selected by setting switch 1 to the DOWN position and switch 2 to the UP position. If running from a receiver, it is necessary to obtain one or more servo pigtails and hook them up according to figure 5.1. If there are only motor drivers being used it is acceptable to power the receiver or microcontroller directly from the Sabertooth as shown. If the system also has to power servos or other 5v loads, we recommend a ParkBEC or a receiver battery pack, as shown in figure 5.2. If using a receiver pack, do not connect power to the 5V line of the Sabertooth because the maximum voltage it can tolerate is 6V.

|  |  |
| --- | --- |
| two-chan-receiver | using-a-parkbec |
| Figure 5.1: R/C connection | Figure 5.2: R/C with a ParkBEC |

There are three operating options for R/C mode. These are selected with switches 4, 5 and 6.

|  |
| --- |
| dipswitch00010037 |
| R/C Mixed or Independent |

# Switch 4: Mixing Mode

When Switch 4 is in the UP position, Mixed mode is selected. In this mode, the R/C signal fed to the S1 input controls the forward/backwards motion of the vehicle. This is usually connected to the throttle channel of a pistol grip transmitter, or the elevator channel of a dual stick transmitter. The R/C signal fed to the S2 input controls the turning of the vehicle.

When switch 4 is in the DOWN position, Independent mode is selected. In this mode, the signal fed to the S1 input directly controls Motor 1 (M1A and M1B) and the signal fed to S2 controls Motor 2.

|  |
| --- |
| dipswitch20035 |
| Exponential mode enabled |

# Switch 5: Exponential response

If switch 5 is in the UP position, the response is linear.

If switch 5 is in the DOWN position, the response to input signals will be exponential. This softens control around the zero speed point, which is useful for control of vehicles with fast top speeds or fast max turning rates.

|  |
| --- |
| dipswitch20034 |
| Microcontroller mode selected |

# Switch 6: R/C Mode/Microcontroller mode select

If switch 6 is in the UP position, then the Sabertooth is in standard R/C mode. This mode is designed to be used with a hobby-style transmitter and receiver. It automatically calibrates the control center and endpoints to maximize stick usage. It also enables a Timeout Failsafe, which will shut down the motors if the Sabertooth stops receiving correct signals from the receiver.

If switch 6 is set in the DOWN position, then Microcontroller mode is enabled. This disables the Timeout Failsafe and auto-calibration. This means that the Sabertooth will continue to drive the motor according to the last command until another command is given. If the control link is possibly unreliable – like a radio - then this can be dangerous due to the robot not stopping. However, it is extremely convenient if you are controlling the Sabertooth from a microcontroller. In this case, commanding the controller can be done with as little as three lines of code.

Output\_High(Pin connected to S1)

Delay(1000us to 2000us)

Output\_Low(Pin connected to S1)

# A note on certain microprocessor receivers

Some receivers, such as the Spektrum AR6000, will output servo pulses before a valid transmitter signal is present. This will cause the Sabertooth to autocalibrate to the receiver’s startup position which may not correspond to the center stick position, depending on trim settings. This may cause the motors to move slowly, even when the transmitter stick is centered. If you encounter this, either consult your receiver manual to reprogram the startup position, or adjust your transmitter trims until the motors stop moving. As a last resort, you can enter R/C microcontroller mode which will disable Sabertooth’s autocalibration.

Mode 3: Simplified Serial Mode

Simplified serial uses TTL level single-byte serial commands to set the motor speed and direction. This makes it easy to interface to microcontrollers and PCs, without having to implement a packet-based communications protocol. Simplified serial is a one-direction only interface. The transmit line from the host is connected to S1. The host’s receive line is not connected to the Sabertooth. Because of this, multiple drivers can be connected to the same serial transmitter. If using a true RS-232 device like a PC’s serial port, it is necessary to use a level converter to shift the –10V to 10V rs-232 levels to the 0v-5v TTL levels the Sabertooth is expecting. This is usually done with a Max232 type chip. If using a TTL serial device like a microcontroller, the TX line of the microcontroller may be connected directly to S1.

Because Sabertooth controls two motors with one 8 byte character, when operating in Simplified Serial mode, each motor has 7 bits of resolution. Sending a character between 1 and 127 will control motor 1. 1 is full reverse, 64 is stop and 127 is full forward. Sending a character between 128 and 255 will control motor 2. 128 is full reverse, 192 is stop and 255 is full forward.

Character 0 (hex 0x00) is a special case. Sending this character will shut down both motors.

Baud Rate Selection

Simplified Serial operates with an 8N1 protocol – 8 data bytes, no parity bits and one stop bit. The baud rate is selected by switches 4 and 5 from the following 4 options

|  |  |
| --- | --- |
| dipswitch00010017 | dipswitch00010021 |
| 2400 Baud: 01x00x | 9600 Baud: 01x10x |
| dipswitch00010019 | dipswitch00010023 |
| 19200 Baud: 01x01x | 38400 Baud: 01x11x |

What baud rate to use is dependent on what your host can provide and the update speed necessary. 9600 baud or 19200 baud is recommended as the best starting points. If communication is unreliable, decrease the baud rate. If communications are reliable, you may increase the baud rate. The maximum update speed on the Sabertooth is approximately 2000 commands per second. Sending characters faster than this will not cause problems, but it will not increase the responsiveness of the controller either.

The baud rate may be changed with power on by changing the DIP switch settings. There is no need to reset or cycle power after a baud rate change.

There are 2 operating options for Simplified Serial. These are selected by the position of Switch 6.

|  |
| --- |
| dipswitch00010017 |
| Standard Simplified Serial |

# Option 1: Standard Simplified Serial Mode

Serial data is sent to input S1. The baud rate is selected with switches 4 and 5. Commands are sent as single bytes. Sending a value of 1-127 will command motor 1 Sending a value of 128-255 will command motor 2. Sending a value of 0 will shut down both motors.

|  |
| --- |
| dipswitch20018 |
| Simplified Serial with Slave Select |

# Option 2: Simplified Serial with Slave Select

This mode is used when it is desirable to have multiple Sabertooth motor drivers running from the same serial transmitter, but you do not wish to use packetized serial. A digital signal (0v or 5v) is fed to the S2 input. This is controlled by the host microcontroller. If the signal on S2 is logic high (5v) when the serial command is sent, then the driver will change to the new speed. If the signal on S2 is not high when the command is sent, then command will be ignored. Pseudo-code demonstrating this is shown below. After sending the signal, allow about 50 us before commanding the Slave Select line to a logic LOW to allow time for processing. A hookup diagram and example pseudo-code are shown in **Figures 6.2** and **6.3**.

|  |  |
| --- | --- |
| Sabertooth2x5Slaceselect | //set controller 1’s speed  Output\_High (S2 pin on controller 1)  USART\_TX(controller 1 speed, 0 to 255)  Delay\_us(50)  Output\_Low (S2 pin on controller 1)  //set controller 2’s speed  Output\_High (S2 pin on controller 2)  USART\_TX(controller 2 speed, 0 to 255)  Delay\_us(50)  Output\_Low (S2 pin on controller 2) |
| **Figure 6.2:** Hookup for Slave Select | **Figure 6.3:** Pseudocode for Slave Select |

# Mode 4: Packetized Serial Mode

Packetized Serial uses TTL level multi-byte serial commands to set the motor speed and direction. Packetized serial is a one-direction only interface. The transmit line from the host is connected to S1. The host’s receive line is not connected to the Sabertooth. Because of this, multiple Sabertooth 2x10 motor drivers can be connected to the same serial transmitter. It is also possible to use SyRen and Sabertooth motor drivers together from the same serial source, as well as any other serial device, as long as it will not act on the packets sent to the Sabertooth. If using a true RS-232 device like a PC’s serial port, it is necessary to use a level converter to shift the –10V to 10V rs-232 levels to the 0v-5v TTL. Packetized serial uses an address byte to select the target device. The baud rate is selected automatically by sending the bauding character (170 in decimal, AA in hex) before any commands are sent.

Packet Overview

The packet format for the Sabertooth consists of an address byte, a command byte, a data byte and a seven bit checksum. Address bytes have value greater than 128, and all subsequent bytes have values 127 or lower. This allows multiple types of devices to share the same serial line.

An example packet and pseudo-code to generate it are shown in **Figures 7.1** and **7.2**

|  |  |
| --- | --- |
| Packet  Address: 130  Command : 0  Data: 64  Checksum: 66 | Void DriveForward(char address, char speed)  {  Putc(address);  Putc(0);  Putc(speed);  Putc((address + 0 + speed) & 0b01111111);  } |
| Figure 7.1: Example 50% forward | Figure 7.2: Pseudocode to generate 7.1 |

Baud Rate Selection:

Packetized Serial operates with an 8N1 protocol – 8 data bytes, no parity bits and one stop bit. The baud rate is automatically calculated by the first character sent. This character must be (170 in decimal) (binary 10101010) and must be sent before any serial communications are done. It is not possible to change the baud rate once the bauding character has been sent. The valid baud rates are 2400, 9600, 19200 and 38400 baud. Until the bauding character is sent, the driver will accept no commands and blue status LED will stay lit. Please note that Sabertooth may take up to a second to start up after power is applied, depending on the power source being used. Sending the bauding character during this time period may cause undesirable results. **When using Packetized Serial mode, please allow a two-second delay between applying power and sending the bauding character to the drivers.**

Address Byte Configuration:

Address bytes are set by switches 4, 5 and 6. Addresses start at 128 and go to 135. The switch settings for the addresses are shown in the chart below

|  |  |
| --- | --- |
| dipswitch00010049 | dipswitch00010053 |
| Address: 128 | Address: 129 |
|  |  |
| dipswitch20051 | dipswitch20055 |
| Address: 130 | Address: 131 |
|  |  |
| dipswitch20050 | dipswitch20054 |
| Address: 132 | Address: 133 |
|  |  |
| dipswitch20052 | dipswitch20056 |
| Address: 134 | Address: 135 |

Commands:

The command byte is the second byte of the packet. There are four possible commands in packetized serial mode. Each is followed by one byte of data

**0: Drive forward motor 1 (decimal 0, binary 0b00000000, hex 0h00)**

This is used to command motor 1 to drive forward. Valid data is 0-127 for off to full forward drive. If a command of 0 is given, the Sabertooth will go into power save mode for motor 1 after approximately 4 seconds.

**1: Drive backwards motor 1 (decimal 1, binary 0b00000001, hex 0h01)**

This is used to command motor 1 to drive backwards. Valid data is 0-127 for off to full reverse drive. If a command of 0 is given, Sabertooth will go into power save mode for motor 1 after approximately 4 seconds.

**2: Min voltage**  **(decimal 2, binary 0b00000010, hex 0h02)**

This is used to set a custom minimum voltage for the battery feeding the Sabertooth. If the battery voltage drops below this value, the output will shut down. This value is cleared at startup, so much be set each run. The value is sent in .2 volt increments with a command of zero corresponding to 6v, which is the minimum. Valid data is from 0 to 120. The function for converting volts to command data is

Value = (desired volts-6) x 5

**3: Max voltage (decimal 3, binary 0b0000011, hex 0h03)**

This is used to set a custom maximum voltage. If you are using a power supply that cannot sink current such as an ATX supply, the input voltage will rise when the driver is regenerating (slowing down the motor.) Many ATX type supplies will shut down if the output voltage on the 12v supply rises beyond 16v. If the driver detects an input voltage above the set limit, it will put the motor into a hard brake until the voltage drops below the set point again. This is inefficient, because the energy is heating the motor instead of recharging a battery, but may be necessary. The driver comes preset for a maximum voltage of 30V. The range for a custom maximum voltage is 0v-25v. The formula for setting a custom maximum voltage is

Value = Desired Volts\*5.12

If you are using any sort of battery, then this is not a problem and the max voltage should be left at the startup default.

**4: Drive forward motor 2 (decimal 4, binary 0b00000100, hex 0h04)**

This is used to command motor 2 to drive forward. Valid data is 0-127 for off to full forward drive. If a command of 0 is given, the Sabertooth will go into power save mode for motor 2 after approximately 4 seconds.

**5: Drive backwards motor 2 (decimal 5, binary 0b00000101, hex 0h05)**

This is used to command motor 2 to drive backwards. Valid data is 0-127 for off to full reverse drive. If a command of 0 is given, the Sabertooth will go into power save mode after approximately 4 seconds.

**6: Drive motor 1 7 bit (decimal 6, binary 0b00000110, hex 0h06)**

This command is used to drive motor 1. Instead of the standard commands 0 and 1, this one command can be used to drive motor 1 forward or in reverse, at a cost of lower resolution. A command of 0 will correspond to full reverse, and a command of 127 will command the motor to drive full forward. A command of 64 will stop the motor.

**7: Drive motor 2 7 bit (decimal 7, binary 0b00000111, hex 0h07)**

This command is used to drive motor 2. Instead of the standard commands 4 and 5, this one command can be used to drive motor 1 forward or in reverse, at a cost of lower resolution. A command of 0 will correspond to full reverse, and a command of 127 will command the motor to drive full forward. A command of 64 will stop the motor.

Mixed mode commands:

Sabertooth can also be sent mixed drive and turn commands. When using the mixed mode commands, please note that the Sabertooth requires valid data for both drive and turn before it will begin to operate. Once both initial data packets have been sent, then turn or drive commands may be sent as needed. You should design your code to either use the independent or the mixed commands. Switching between the command sets will cause the vehicle to stop until new data is sent for both motors.

**8: Drive forward mixed mode (decimal 8, binary 0b00001000, hex 0h08)**

This is used to command the vehicle to drive forward in mixed mode. Valid data is 0-127 for off to full forward drive.

**9: Drive backwards mixed mode (decimal 9, binary 0b00001001, hex 0h09)**

This is used to command the vehicle to drive backwards in mixed mode. Valid data is 0-127 for off to full reverse drive.

**10: Turn right mixed mode (decimal 10, binary 0b00001010, hex 0h0a)**

This is used to command the vehicle to turn right in mixed mode. Valid data is 0-127 for zero to maximum turning speed.

**11: Drive turn left mixed mode (decimal 11, binary 0b00001011, hex 0h0b)**

This is used to command the vehicle to turn left in mixed mode. Valid data is 0-127 for zero to maximum turning speed.

**12: Drive forwards/back 7 bit (decimal 12, binary 0b00001100, hex 0h0c)**

This is used to command the vehicle to move forwards or backwards. A command of 0 will cause maximum reverse, 64 will cause the vehicle to stop, and 127 will command full forward.

**13: Turn 7 bit (decimal 13, binary 0b00001101, hex 0h0d)**

This is used to command the vehicle to turn right or left. A command of 0 will cause maximum left turn rate, 64 will cause the vehicle to stop turning , and 127 will command maximum right turn rate. Checksum:

To prevent data corruption, each packet is terminated with a checksum. If the checksum is not correct, the data packet will not be acted upon. The checksum is calculated as follows:

Checksum = address byte +command byte +data byte

The checksum should be added with all unsigned 8 bit integers, and then Boolean AND-ed with the mask 0b01111111 in an 8 bit system.

Example of Packetized Serial:

The following is an example function for commanding two Dimension Engineering motor drivers using Packetized Serial Mode. **Figure 7.3** shows an example hookup and **Figure 7.4** shows an example function.

|  |  |
| --- | --- |
| Sabertooth-packetized-Seria | Void DriveForward(char address, char speed)  {  Putc(address);  Putc(0);  Putc(speed);  Putc((address + 0 + speed) & 0b01111111);  } |
| Figure 7.3: Packetized serial hookup | Figure 7.4: Packetized Serial Function |

Example: So in this function, if address is 130, command is 0 (for driving forward), speed is 64, the checksum should calculate as follows:

130+0+64 = 194

194 in binary is 0b11000010

0b11000010 & 0b01111111 = 0b01000010

Once all the data is sent, this will result in the Sabertooth with address 130 driving forward at roughly half throttle.

Emergency Stop:

In Packetized Serial mode, the S2 input is configured as an active-low emergency stop. It is pulled high internally, so if this feature isn’t needed, it can be ignored. If an emergency stop is desired, all the S2 inputs can be tied together. Pulling the S2 input low will cause the driver to shut down. This should be tied to an emergency stop button if used in a device that could endanger humans.

**Code**

**#include <SoftwareSerial.h>**

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\* DIP Switches as per the wizard:**

**\* - NiMh Battery**

**\* - TTL RS232**

**\* - Simplified Serial Mode**

**\* - Two Saberteeth connected**

**\* - 9600 baudrate**

**\***

**\* Pin 1 - ON**

**\* Pin 2 - OFF**

**\* Pin 3 - ON**

**\* Pin 4 - OFF**

**\* Pin 5 - ON**

**\* Pin 6 - OFF**

**\* Code created by Dave Gaskin for ECE 579, Winter Semester 2012**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**// Labels for use with the Sabertooth 2x5 motor controller**

**// Digital pin 13 is the serial transmit pin to the**

**// Sabertooth 2x5**

**#define SABER\_TX\_PIN 18**

**// NOT USED (but still init'd)**

**// Digital pin 12 is the serial receive pin from the**

**// Sabertooth 2x5**

**#define SABER\_RX\_PIN 19**

**// Set to 9600 through Sabertooth dip switches**

**#define SABER\_BAUDRATE 9600**

**// Simplified serial Limits for each motor**

**//64 is soft stop**

#define SABER\_MOTOR1\_FULL\_FORWARD 127

#define SABER\_MOTOR1\_FULL\_REVERSE 1

**//192 is soft stop**

#define SABER\_MOTOR2\_FULL\_FORWARD 255

#define SABER\_MOTOR2\_FULL\_REVERSE 128

**// Motor level to send when issuing the full stop command**

#define SABER\_ALL\_STOP 0

SoftwareSerial SaberSerial = SoftwareSerial( SABER\_RX\_PIN,

SABER\_TX\_PIN );

**// Setting up the actuator pots, connected to a8-a11:**

**//analogRead(Actuatorx) reads in the value from the pin.**

int Actuator1 = A8; // selects the input pins for the actuator pot

int Actuator2 = A9;

int Actuator3 = A10;

int Actuator4 = A11;

**// places to store actuator loaction values**

int Actuator1Value = 0;

int Actuator2Value = 0;

int Actuator3Value = 0;

int Actuator4Value = 0;

**//setting up select pins. 52 in motor controller 1, 53 is motor controller 2**

int MC1 = 52;

int MC2 = 53;

void initSabertooth( void )

{

**// Init software UART to communicate**

**// with the Sabertooth 2x5**

pinMode( SABER\_TX\_PIN, OUTPUT );

SaberSerial.begin( SABER\_BAUDRATE );

**// 2 second time delay for the Sabertooth to init**

delay( 2000 );

**// Send full stop command**

**SaberSerial.write(byte(SABER\_ALL\_STOP));**

**//setting up pinmodes for select pins**

pinMode(MC1, OUTPUT);

pinMode(MC2, OUTPUT);

}

void setup( )

{

initSabertooth( );

}

void loop ( )

{

**// setting nuetral position**

**//TS to reset positions**

**// digitalWrite(MC1, HIGH);**

**//digitalWrite(MC2, HIGH);**

**//SaberSerial.write(byte(SABER\_MOTOR1\_FULL\_REVERSE));**

**// SaberSerial.write(byte(SABER\_MOTOR2\_FULL\_REVERSE));**

**// SaberSerial.write(byte(SABER\_MOTOR1\_FULL\_FORWARD));**

**// SaberSerial.write(byte(SABER\_MOTOR2\_FULL\_FORWARD));**

**// Reading in values for actuator position, lower is more extended**

Actuator1Value = analogRead(Actuator1);

Actuator2Value = analogRead(Actuator2);

Actuator3Value = analogRead(Actuator3);

Actuator4Value = analogRead(Actuator4);

**// delay(10);**

**//64 will stop the motor 1**

**//192 is stop for motor 2**

**//LOW diables both motor controllers**

**//HIGH enables both motor controllers**

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

**//All MC's active to set a nuetral position**

if (Actuator1Value > 700)

{

digitalWrite(MC1, HIGH);

digitalWrite(MC2, LOW);

SaberSerial.write(byte(SABER\_MOTOR2\_FULL\_FORWARD));

**//delay(15);**

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

else //writing to act 1 to stop

{

digitalWrite(MC1, HIGH);

digitalWrite(MC2, LOW);

SaberSerial.write(byte(192));

**// delay(15);**

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

if (Actuator2Value > 700)

{

digitalWrite(MC1, HIGH);

digitalWrite(MC2, LOW);

SaberSerial.write(byte(SABER\_MOTOR1\_FULL\_FORWARD));

**//delay(15);**

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

else // writing to act 2 to stop

{

digitalWrite(MC1, HIGH);

digitalWrite(MC2, LOW);

SaberSerial.write(byte(64));

// delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

if (Actuator3Value > 700)

{

digitalWrite(MC1, LOW);

digitalWrite(MC2, HIGH);

SaberSerial.write(byte(SABER\_MOTOR1\_FULL\_FORWARD));

//delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

else // writing to act 3 to soft stop

{

digitalWrite(MC1, LOW);

digitalWrite(MC2, HIGH);

SaberSerial.write(byte(64));

//delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

if (Actuator4Value > 700)

{

digitalWrite(MC1, LOW);

digitalWrite(MC2, HIGH);

SaberSerial.write(byte(SABER\_MOTOR2\_FULL\_FORWARD));

// delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

else

{ //writing to Act 4 to soft stop

digitalWrite(MC1, LOW);

digitalWrite(MC2, HIGH);

SaberSerial.write(byte(192));

// delay(15);

digitalWrite(MC1, LOW);

digitalWrite(MC2, LOW);

}

}

**Time Log**

|  |  |
| --- | --- |
| **Tasks** | **Times Spent** |
| **Planning** | **10 hours** |
| **Part Selection** | **20 hours** |
| **Hardware building/Machining** | **20 hours** |
| **Shopping** | **5 Hours** |
| **Programming** | **10 hours** |
| **Report Writing** | **10 hours** |
| **Trouble Shooting** | **20 hours** |
| **Total** | **93 hours** |

**Parts List, Cost and Source**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Item** | **Source** | **Price** | **QTY** | | **Shipping** | **Total** |  |
| Arduino Mega 2560 Microcontroller rev 3 | http://www.trossenrobotics.com/p/arduino-mega-2560.aspx | 64.95 | | 1 | 0 | 64.95 |  |
| Arduino Ethernet Shield | http://www.sparkfun.com/products/9026 , http://www.trossenrobotics.com/p/arduino-ethernet-shield.aspx | 45.95 | | 1 | 0 | 45.95 |
| Arduino Box | http://www.trossenrobotics.com/p/arduino-project-box.aspx | 11.95 | | 1 | 0 | 11.95 |
| Pig tail for power | http://www.trossenrobotics.com/store/p/6612-Barrel-Jack-Female-Pigtail-Lead-2-1-5-5mm.aspx | 2.95 | | 2 | 0 | 5.9 |
| 6vDC power for board | http://www.trossenrobotics.com/p/power-supply-6vdc-2a.aspx | 9.95 | | 1 | 0 | 9.95 |
| 12v 5A pwr supply for actuators | http://www.trossenrobotics.com/p/power-supply-12vdc-5a.aspx | 19.95 | | 1 |  | 19.95 |
| Servo wire | http://www.trossenrobotics.com/store/p/6421-50-Feet-Standard-Servo-Wire.aspx | 12.95 | | 1 |  | 12.95 |
| Motor Controller | http://www.trossenrobotics.com/store/p/5104-Sabertooth-dual-5A-motor-driver-for-R-C.aspx | 59.99 | | 2 | 0 | 119.98 |
| Firgelli Actuators | http://www.firgelliauto.com/product\_info.php?cPath=109&products\_id=159 | 129.99 | | 4 | 0 | 519.96 |
| Actuator Mounting | http://www.firgelliauto.com/product\_info.php?products\_id=54 | 9 | | 4 | 0 | 36 |
| **Totals** |  |  | |  |  | **847.54** |