**Principia College ASME Custom Tank Control Unit: Quick start Guide**

**April 21, 2017**

**Description**

This quick start guide is intended to be used by the Principia College engineering department for setting up, programming, configuring, and testing the ASME group’s custom tank control unit (CTCU). Any questions regarding the CTCU should be directed to the Math/Natural Sciences Operations Manager, Steve Ward. This guide is for educational purposes only and the provided information, design(s), algorithms, and software are not intended for commercialization, resale, or redistribution without expressed consent from Principia College and the contributing author(s).

**Resources/Reference Materials**

Radio Controllers and Pin Signaling

<http://www.impulseadventure.com/elec/attiny-spi-rc-pwm.html>

Interrupt Service Routines (ISRs) - High Speed Pin Change Detection

<http://www.engblaze.com/we-interrupt-this-program-to-bring-you-a-tutorial-on-arduino-interrupts/>

Pulse Width Modulation – Controlling Motors with PWM and Basic DC Motor Theory

<http://www.electronics-tutorials.ws/blog/pulse-width-modulation.html>

**Hardware**

(x1) Arduino Mega 2560 R3

(x2) Pololu VNH5019 PWM Motor Driver

(x1) Pololu S18V20F5 5v 2A Voltage Regulator

(x1) Principia ASME Arduino Mega 2560 PCB Shield 1.01

(x1) Turnigy 2.4 Ghz 9X 8Ch V2 Receiver

(x1) Turnigy 9X Transmitter

(x1) 2 Channel Opto-Isolated 5v Relay

(x5) 1x8 Pin Stackable Header Pins

(x2) 1x4 Pin Stackable Header Pins

(x3) 1x5 Pin Stackable Header Pins

(AR) 14AWG – 16AWG Red Lead Wire Approx. 36” Length

(AR) 14AWG – 16AWG Black Lead Wire Approx. 36” Length

(x2) Jumper Wires (20 AWG – 24AWG) Approx. ¼” Length

(x1) Arduino Programming Cable – (aka USB printer cable)

**Additional Hardware (Brushed Motors)**

*Track Motors*

(x2) 24V 150W 700RPM Groschopp PM6013-PS1905

*Launcher Motors*

(x1) 24V 2500RPM Groschopp PM6015 Motor Only

OR

(x2) 24V 2500RPM Groschopp PM6013 Motor Only

**Software**

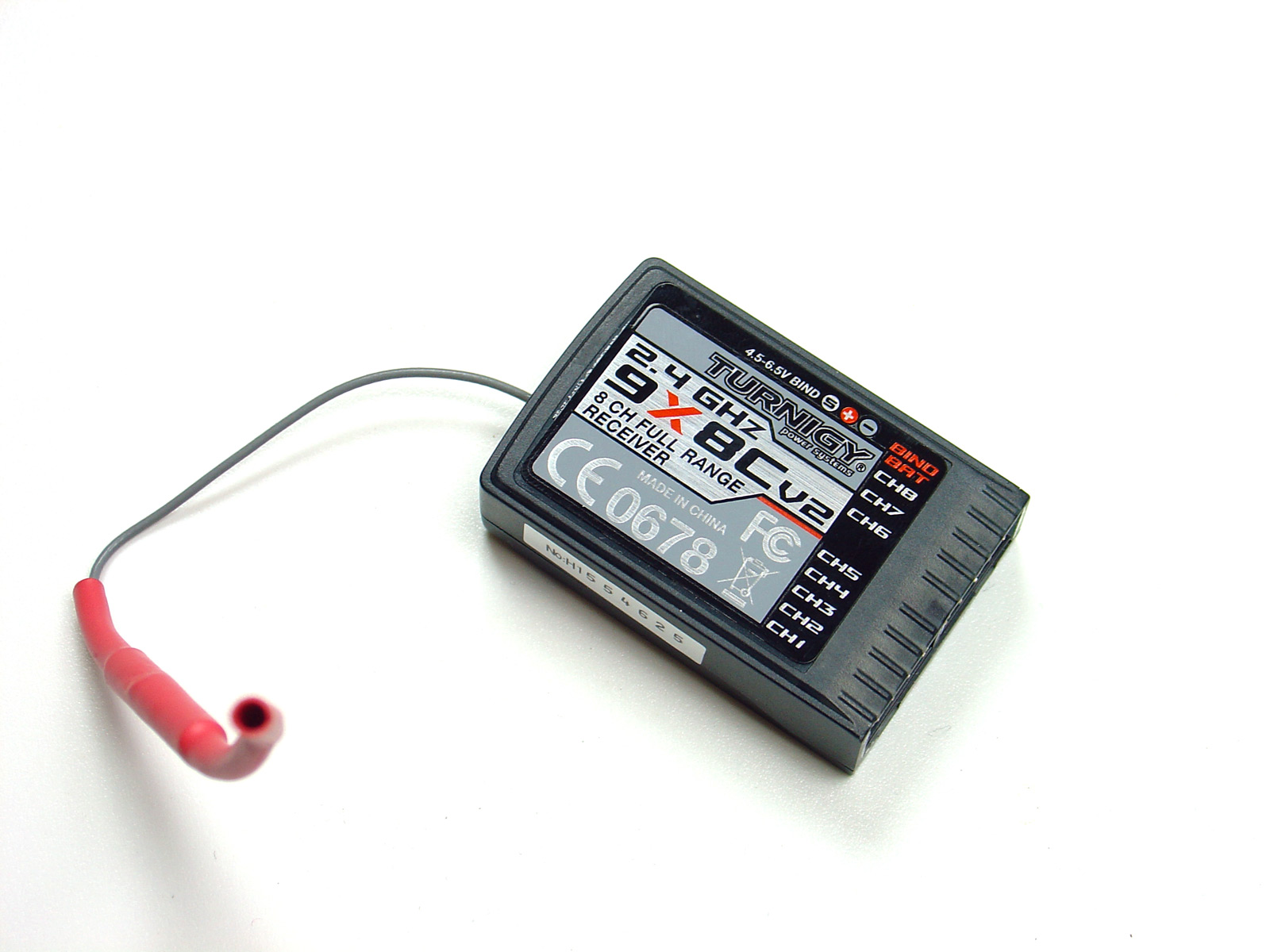
Arduino IDE/Bootloader - <https://www.arduino.cc/en/main/software>

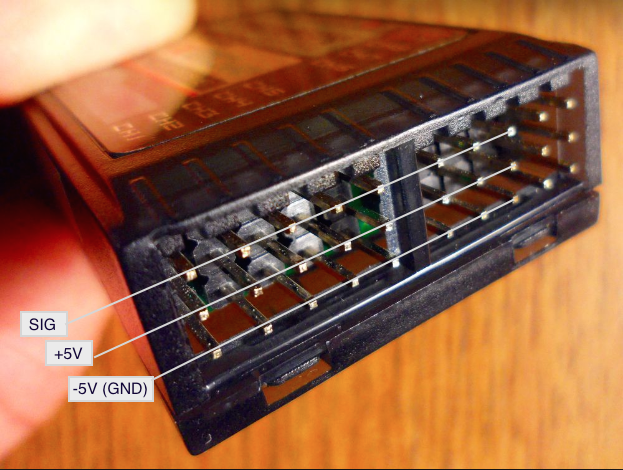
STEP 1: PAIRING THE RECEIVER AND TRANSMITTER

Manual - <https://hobbyking.com/media/file/725056143X2037269X20.pdf>

Video - <https://www.youtube.com/watch?v=zuJgfA3_Jqw>

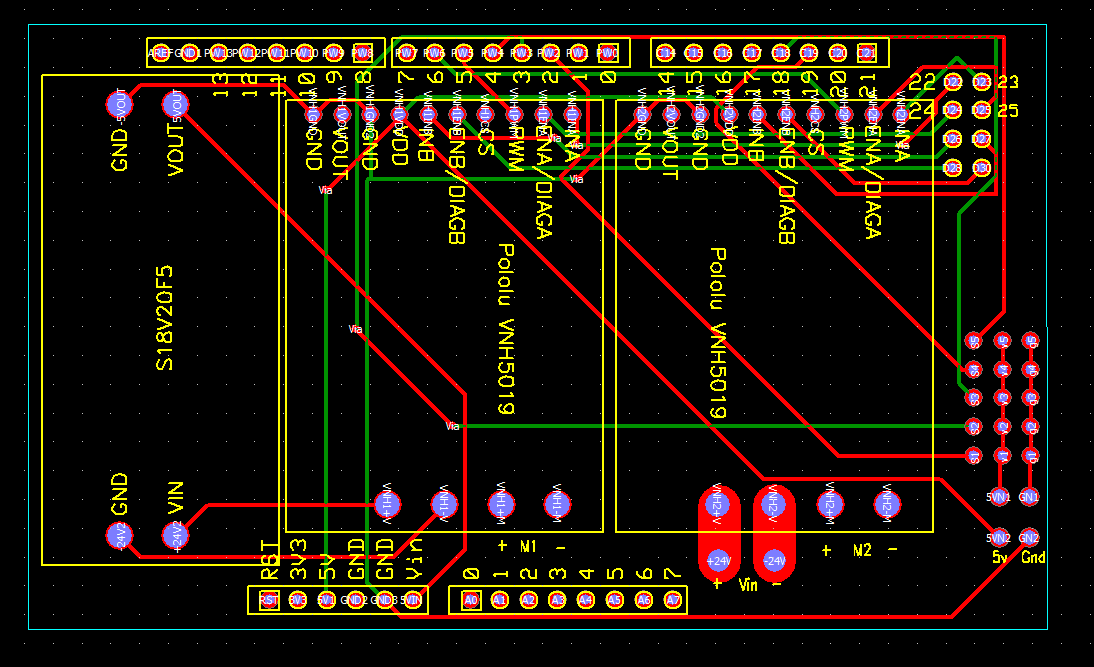
To attach a new receiver, First you must turn on the transmitter, select menu item “System Settings", then select "MODEVAT", set to “PPM” mode, and then turn off the transmitter. Insert the provided jumper cable into the receiver. The jumper cable should be in the “BIND” column as indicated on top of the receiver. Next, connect the receiver battery to the receiver in the “BAT” column as indicated on top of the receiver.

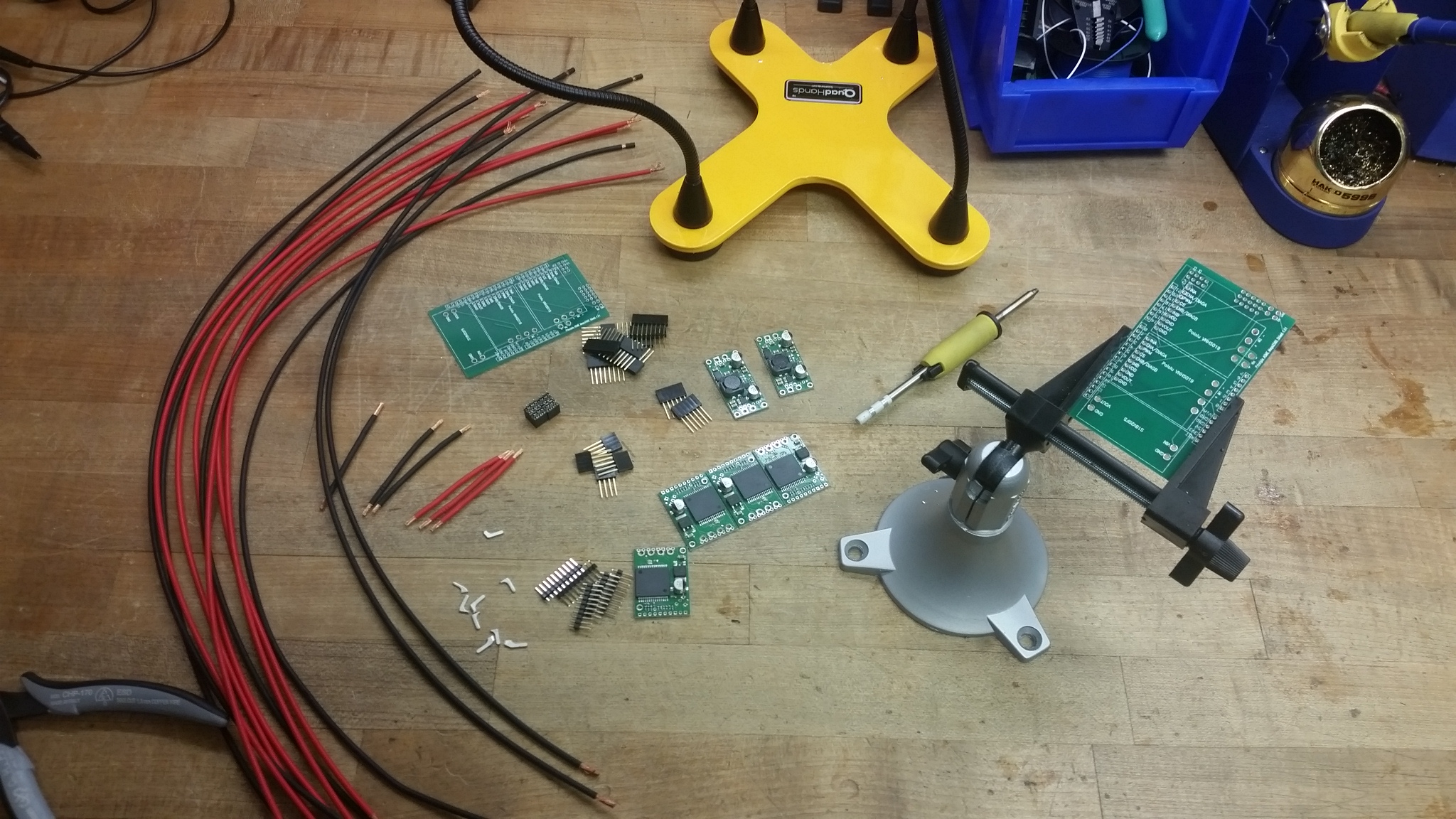
Turn on the transmitter while pressing the BIND button on the back of the transmitter. The red LED on the receiver should go from blinking (ready to bind) to solid red (bound), indicating that the receiver and transmitter have been paired. See pictures (*Left*) for easy receiver pin identification. Turn off the transmitter and disconnect the wires from the receiver.



STEP 2: ASSEMBLE THE CIRCUIT BOARD (PCB)

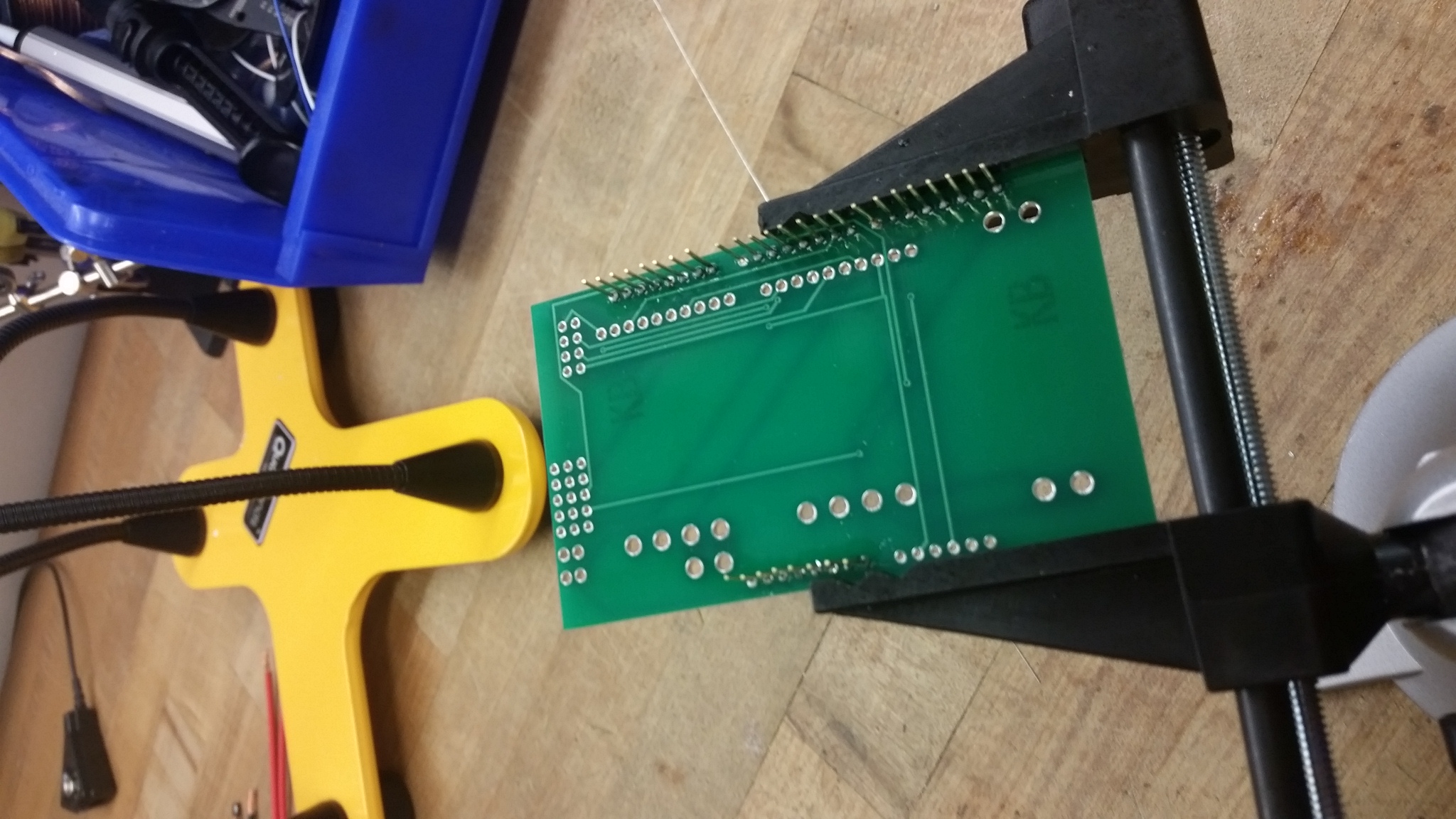
The diagram below shows the circuit traces connecting the pins between the Pololu VNH5019 modules, the S18V20F5 voltage regulator, and the custom PCB shield.



****

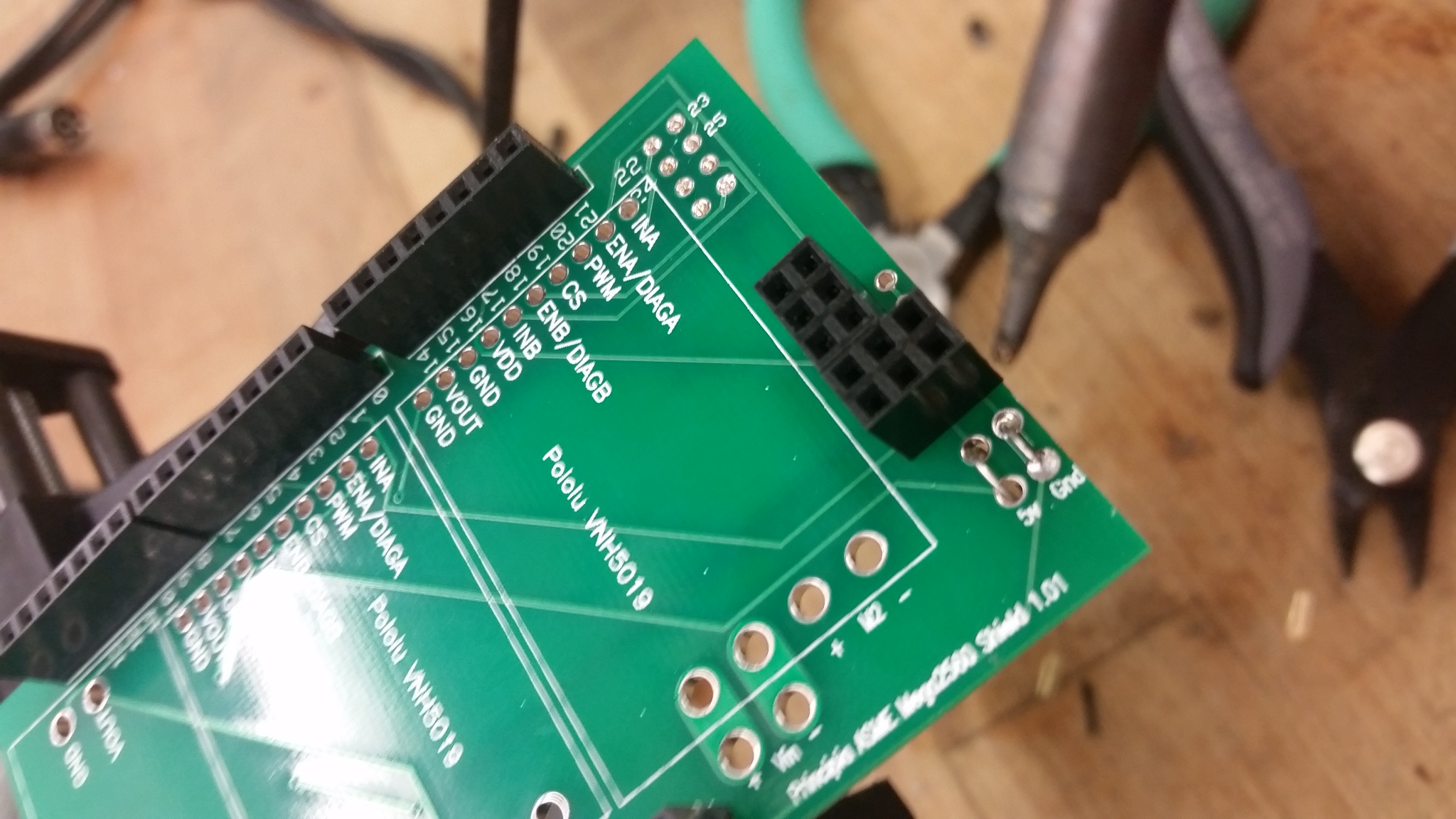
**Solder Header Pins to Shield**

Start by soldering in the header shield pins into the PCB shield. You will need: (4) 8 pin headers, (1) 6 pin header, (3) 5 pin headers, and (2) 4 pin headers. Once the headers have been soldered to the bottom side of the PCB shield, trim pins, S1 – S5, V1 – V5, and g1 – g5, on the bottom side of the PCB shield.

**

Next, solder small jumper wires into the PCB shield connecting pins 5VN2 with 5VN1, and GN2 with GN1. Check the soldered connections to make sure that a good electrical connection has been made to the PCB shield.

*Note: Be careful not to overheat the shield pins or the circuit board thru holes. Overheating these parts can lead to poor or non-functioning electrical connections.*

**

**Solder S18V20F5 to Shield**

Place the voltage regulator on the topside of the PCB shield. Make sure that the GND and VIN on the voltage regulator is on the same side as the GND and VIN as indicated on the PCB shield.

*Note: Placing the voltage regulator in the wrong direction will result in a damaged and non-functioning board.*

Place jumpers through the 4 holes on the overlapped PCBs and solder connections to the top of the voltage regulator. Next, flip the PCB shield over and solder the jumpers to the bottom side of the PCB shield board. Make sure that the solder has flowed through to make a good connection to the voltage regulator module.

Trim the excess jumper wires from the top and bottom sides of the PCB shield.

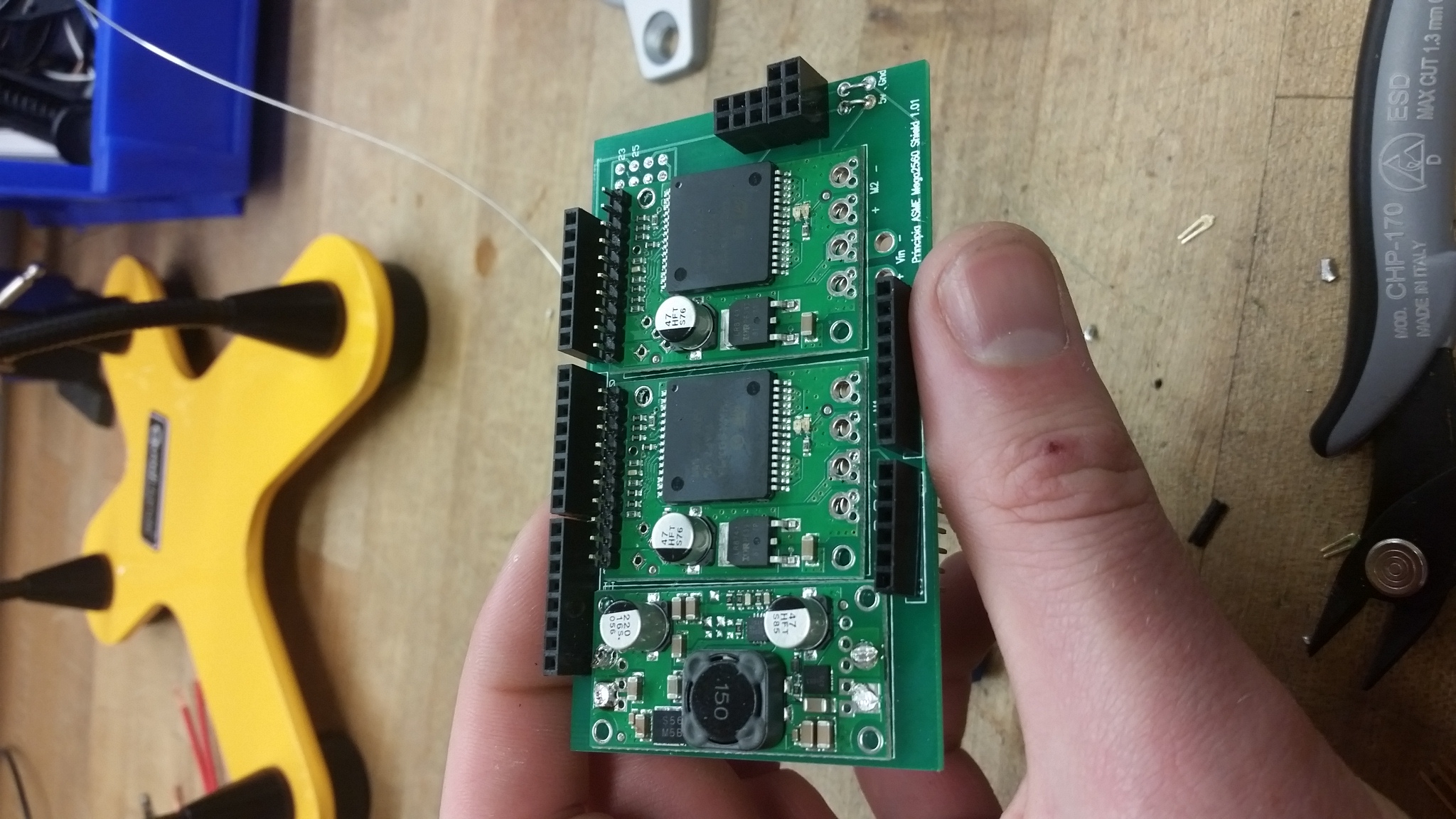


**Solder the Motor Drivers to Shield**

Place one of the Pololu VNH5019 motor drivers onto the PCB shield and solder the small header pins to the bottom side of the PCB shield.

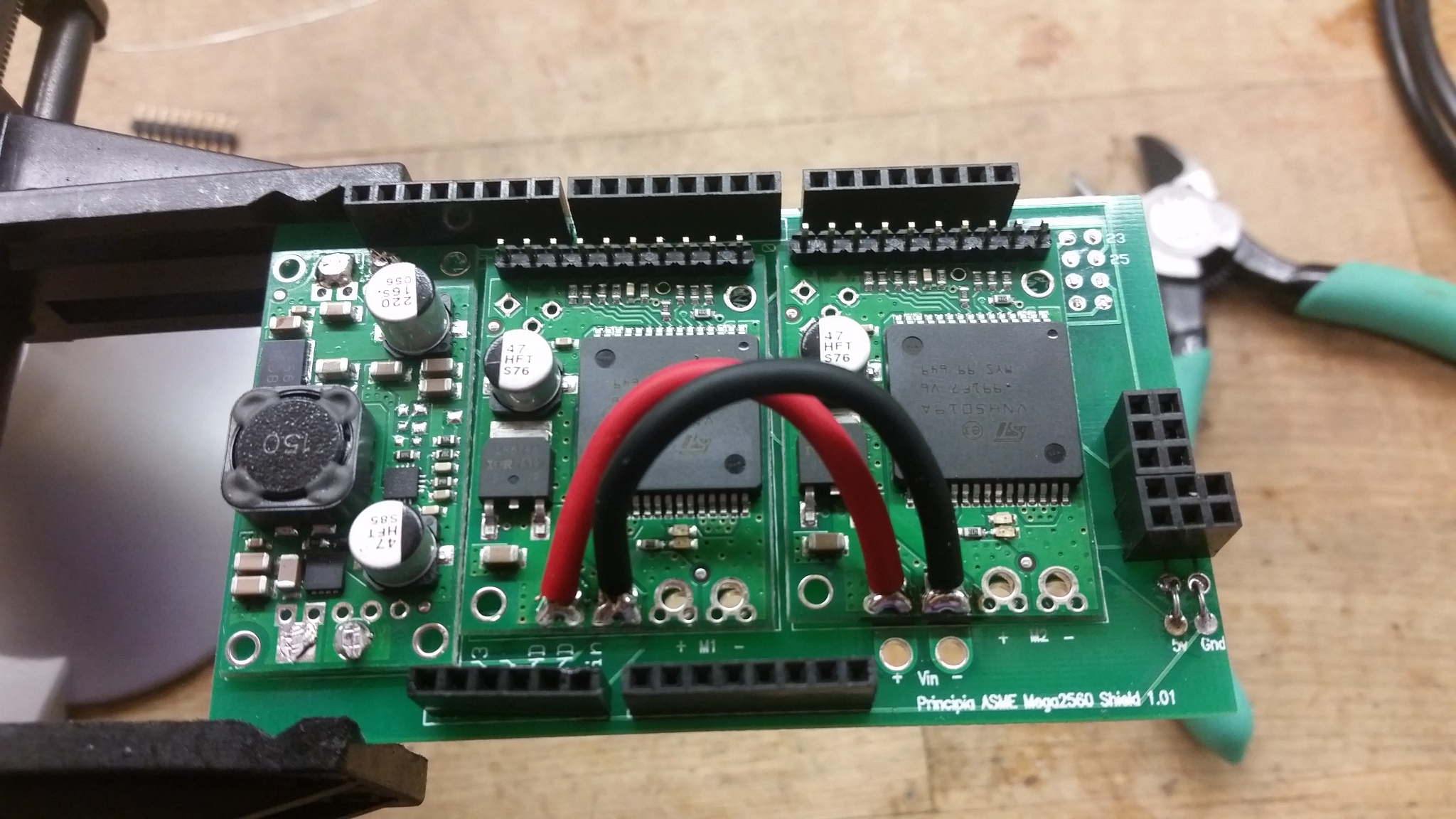
*Note: Be careful to not overheat the motor driver. The soldering iron will easily heat up the pins/traces and can drastically increase the temperature of the motor driver chip. Excess heat will damage the chip.*

Next, solder the second Pololu VNH5019 motor driver onto the PCB shield. Use the same steps as before.



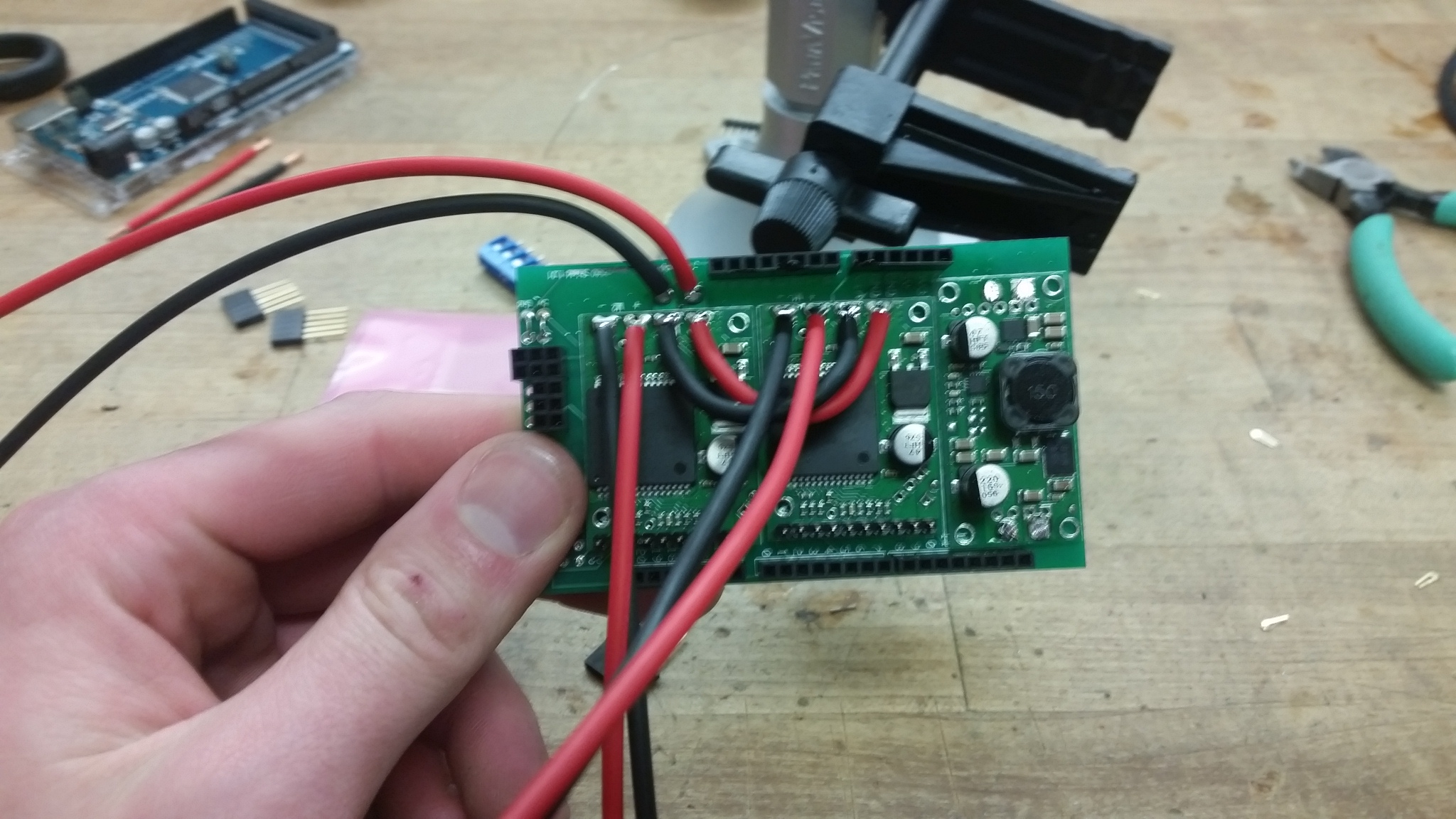
Take (2) 1” – 2” segments of 14AWG jumper wires and place one end through VNH2+V and the other end through VNH1+V. Take the other jumper wire and place one end through VNH1-V and the other end through VNH2-V. Make sure enough of the wire has been stripped to allow the copper strands to pass all the way through both boards.   
  
Leave about ¼” of exposed copper wire on the top side of the motor drivers and solder the jumper wires to the top of the motor driver boards. Take care to make sure that the solder flows into/around the copper wire. Once a good electrical connection has been made, flip over the PCB shield and solder the jumper wires to the bottom section of the PCB shield. Trim the excess lead wire from the bottom of the PCB shield as necessary.

*Note: Make sure that excess solder does not accidentally connect any of the +V pins with the –V pins. Accidental connection will result in a dead short that will damage the board and the battery powering it.*

**

**Solder Lead Wires**

For this next step you will need sections of 14AWG lead wire. As with the previous step, make sure that enough of the wire has been stripped to ensure the copper strands will pass all of the way through both boards.   
  
Solder individual lead wires to VNH1+M, VNH1-M, VNH2+M, and VNH2-M, using the same steps as discussed in the previous section.   
  
Finally, solder individual lead wires to +24V and -24V on the PCB shield. The board is now assembled and ready to be programmed.

******

STEP3: PROGRAM THE MICROCONTROLLER (Mega 2560)

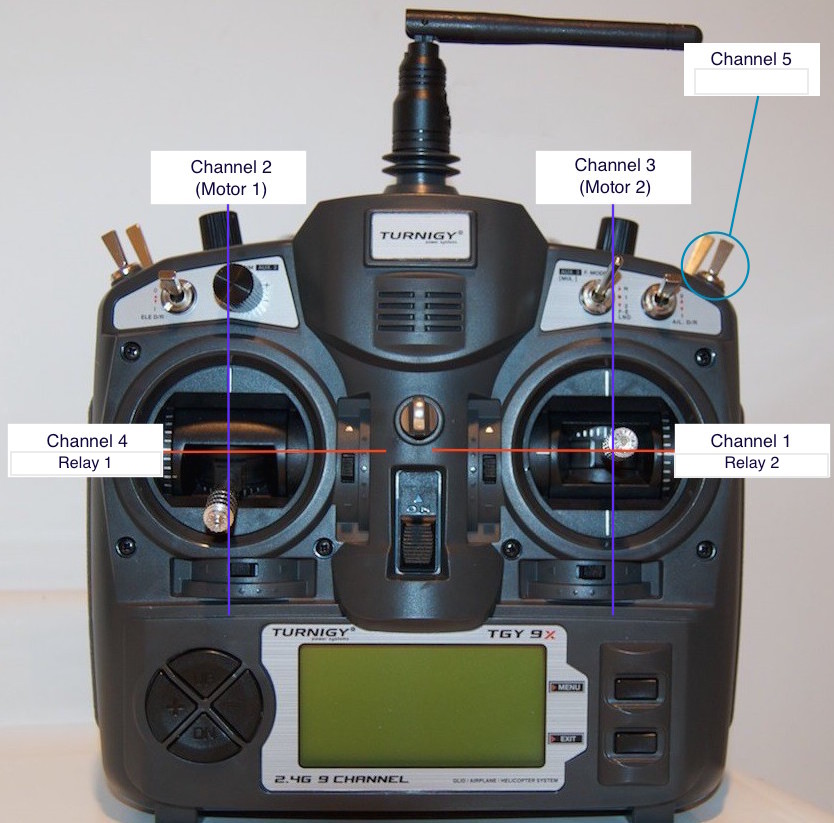
Upload the code from the end of this document to the Arduino Mega 2560 using the compatible Arduino IDE software.

For more details on getting started with Arduino visit the link below. <https://www.arduino.cc/en/Guide/HomePage>

STEP4: CONNECT THE PCB SHIELD TO MICROCONTROLLER AND CONNECT THE MOTORS AND BATTERY.

|  |  |  |
| --- | --- | --- |
| ***Pin Mapping*** | | |
| **2 Ch Relay** |  | **Arduino Shield** |
| Gnd | 🡪 | GND |
| IN1 | 🡪 | Pin 8 |
| IN2 | 🡪 | Pin 9 |
| VCC | 🡪 | 5V |
|  |  |  |
| **Turnigy 9X 8Ch Receiver** |  |  |
| Ch1 | 🡪 | S1 |
| Ch2 | 🡪 | S2 |
| Ch3 | 🡪 | S3 |
| Ch4 | 🡪 | S4 |
| Ch5 | 🡪 | S5 |
| 5V | 🡪 | V1 - V5 |
| GND | 🡪 | g1 - g5 |

STEP 5: TRANSMITTER CONTROL LAYOUT AND CHANNEL MAPPING

  
  
The Turnigy 9X transmitter comes equipped to handle up to 8 channels output. The controller can be set to different modes that change where on the transmitter these channels are mapped. For the stock CTCU set up, the transmitter should be set to “Mode 1”.

**Right Stick Forward – M2 CW Rotation**

**Right Stick Backward – M2 CCW Rotation**

**Right Stick Right – Relay 2 OFF**

**Right Stick Left – Relay 2 ON**

**Left Stick Forward – M1 CCW Rotation**

**Left Stick Backward – M1 CW Rotation**

**Left Stick Right – Relay 1 OFF**

**Left Stick Left – Relay 1 ON**

**CODE**

/\*

Author: Alec Korver

Date: 4/08/2017

Description:

Principia College ASME Tank Competition Arduino Mega 2560 Shield

The following code is for controlling (2) track motors and (1 or 2) tennis

ball launcher motors.

Each shield uses

(1) Custom PCB Shield

(2) Pololu VNH5019 motor drivers (5V - 24V) 12A Cont. 30A Peak 20kHz max PWM

(1) Turnigy 9x8C V2 Receiver - 8 Channel

(1) Pololu 5V S18V20F5 Voltage Regulator

To control the motors you will need (1) Turnigy 9x 8 Channel handheld

remote control. The controll will need to be synced with the reciever prior

to use. Only 3 channels are initially used for this system. There are 5

additional channels, which can be used for future expansions.

Handheld remote control info

https://hobbyking.com/media/file/725056143X2037269X20.pdf

//Good information on reading the PWM signals from the reciever

http://www.impulseadventure.com/elec/attiny-spi-rc-pwm.html

IMPORTANT NOTE ABOUT LAUNCHER

The launcher motor should be up to full speed before a tennis ball is dropped in.

This ensures that it will work similar to a pitching machine. Otherwise the launcher will be

underpowered. To increase the firing distance of the launcher there are a few options:

1)Increase the wheel inertia (increase mass of wheel)

2)Increase diameter of the launcher wheel (Larger diameter means higher speed at edge)

3)Both 1 and 2

4)Increase the voltage to the motor (Do not run continuously at higher voltage because the motor WILL burn up!!)

\*/

//CODE FOR CUSTOM CONTRLOL

#include <avr/wdt.h> //WATCHDOG TIMER

//https://tushev.org/articles/arduino/5/arduino-and-watchdog-timer

//Pins for Pololu PWM motor driver 1

//The first 4 pins need to be set either high or low

//How they are set determines whether the motor will be

//rotating CW or CCW or if it is braking or regenerating.

//See datasheet on motor driver for more details.

const int M1InA\_pin = 22;

const int M1EnA\_pin = 24;

const int M1InB\_pin = 26;

const int M1EnB\_pin = 28;

const int M1Pwm\_pin = 5;

//Pins for Pololu PWM motor driver 2

//The first 4 pins need to be set either high or low (1 or 0)

//How they are set determines whether the motor will be

//rotating CW or CCW or if it is braking or regenerating.

//See datasheet on motor driver for more details.

const int M2InA\_pin = 23;

const int M2EnA\_pin = 25;

const int M2InB\_pin = 27;

const int M2EnB\_pin = 29;

const int M2Pwm\_pin = 6;

//Pins for Turnigy 9x8C V2 Reciever

//Each pin maps to one channel of the receiver

//The input from the reciever to each pin will be a PWM signal

//The width of the PWM signal is set by the hand held remote control

//Checking this width tells us the different thumbstick and switch positions

const int Ch1\_pin = 2;//Default Right Thumbstick L/R

const int Ch2\_pin = 3;//Default Right Thumbstick Up/Down

const int Ch3\_pin = 18;//Default Left Thumbstick Up/Down

const int Ch4\_pin = 19;//Default Left Thumbstick L/R

const int Ch5\_pin = 4;//Default mapped to right gear switch

//Pins for launcher output - aka they trip the relay

const int relay1 = 8;

const int relay2 = 9;

//Variables to store the PWM values read from each receiver channel(pin) - (micro seconds)

volatile long Ch1Pwm;

volatile long Ch2Pwm;

volatile long Ch3Pwm;

volatile long Ch4Pwm;

volatile long Ch5Pwm;

//Maximum pulse width (micro seconds) - TEST AND SET THESE VALUES!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

long Ch1MaxPw = 1864;

long Ch2MaxPw = 1864;

long Ch3MaxPw = 1864;

long Ch4MaxPw = 1864;

long Ch5MaxPw = 1826;

//Middle value for pulse width (micro seconds) - This is the center position for the thumbsticks - TEST AND SET THESE VALUES!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

int Ch1MidPw = 1448;

int Ch2MidPw = 1448;

int Ch3MidPw = 1448;

int Ch4MidPw = 1448;

int Ch5MidPw = 1448;

//Min value for pulse width (micro seconds) - This is the center position for the thumbsticks - TEST AND SET THESE VALUES!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

int Ch1MinPw = 1028;

int Ch2MinPw = 1028;

int Ch3MinPw = 1028;

int Ch4MinPw = 1028;

int Ch5MinPw = 1028;

const float M\_Ratio = 0.607;

//Variables for holding the PWM value (0 - 255) that we will send to the

//motor driver. This will set our left and right motor speeds

//255 is 100% output aka FULL SPEEED (FULL 24V)

long M1\_Pwm;

long M2\_Pwm;

//These are global variables which are used to transfer data from the

//ISR to the main code. These must be declared as VOLATILE.

volatile long Ch1RisingEdgeTime, Ch1FallingEdgeTime, Ch1TimeDif;

volatile long Ch2RisingEdgeTime, Ch2FallingEdgeTime, Ch2TimeDif;

volatile long Ch3RisingEdgeTime, Ch3FallingEdgeTime, Ch3TimeDif;

volatile long Ch4RisingEdgeTime, Ch4FallingEdgeTime, Ch4TimeDif;

//SETUP CODE - RUNS ONE TIME AT STARTUP TO INITIALIZE MICRO SETTINGS

void setup() {

Serial.begin(9600);//This tells our microcontroller what our clock rate is. Allows for communication protocols.

//UNCOMMENT TO USE CH1 ISR

attachInterrupt(digitalPinToInterrupt(Ch1\_pin),Channel1DataRising,RISING);

attachInterrupt(digitalPinToInterrupt(Ch1\_pin),Channel1DataFalling,FALLING);

//Checks Motor 1 for rising and falling edge to determine PWM length

attachInterrupt(digitalPinToInterrupt(Ch2\_pin),Channel2DataRising,RISING);

attachInterrupt(digitalPinToInterrupt(Ch2\_pin),Channel2DataFalling,FALLING);

//Checks Motor 2 for rising and falling edge to determine PWM length

attachInterrupt(digitalPinToInterrupt(Ch3\_pin),Channel3DataRising,RISING);

attachInterrupt(digitalPinToInterrupt(Ch3\_pin),Channel3DataFalling,FALLING);

//UNCOMMENT TO USE CH4 ISR

attachInterrupt(digitalPinToInterrupt(Ch4\_pin),Channel4DataRising,RISING);

attachInterrupt(digitalPinToInterrupt(Ch4\_pin),Channel4DataFalling,FALLING);

pinMode(M1InA\_pin,OUTPUT);

pinMode(M1EnA\_pin,OUTPUT);

pinMode(M1InB\_pin,OUTPUT);

pinMode(M1EnB\_pin,OUTPUT);

pinMode(M1Pwm\_pin,OUTPUT);

pinMode(M2InA\_pin,OUTPUT);

pinMode(M2EnA\_pin,OUTPUT);

pinMode(M2InB\_pin,OUTPUT);

pinMode(M2EnB\_pin,OUTPUT);

pinMode(M2Pwm\_pin,OUTPUT);

pinMode(Ch1\_pin,INPUT);

pinMode(Ch2\_pin,INPUT);

pinMode(Ch3\_pin,INPUT);

pinMode(Ch4\_pin,INPUT);

pinMode(Ch5\_pin,INPUT);

pinMode(relay1,OUTPUT);

pinMode(relay2,OUTPUT);

//Sets the motor driver to forward and reverse functions only

digitalWrite(M1EnA\_pin,HIGH);

digitalWrite(M1EnB\_pin,HIGH);

digitalWrite(M2EnA\_pin,HIGH);

digitalWrite(M2EnB\_pin,HIGH);

//Sets motor1 and motor2 to forward direction on startup

digitalWrite(M1InA\_pin,HIGH);

digitalWrite(M1InB\_pin,LOW);

digitalWrite(M2InA\_pin,HIGH);

digitalWrite(M2InB\_pin,LOW);

wdt\_enable(WDTO\_1S);

}

//MAIN LOOP - THIS WILL BE RUNNING AND LOOPING CONTINUOUSLY

void loop() {

wdt\_reset();//Resets watchdog timer

MotorControl();//Sets the output values to all motors

Channel5Data();//Checks the state of the launcher input

launcher();//Turns the launcher on or off

}

//CUSTOM FUNCTIONS - ALL CODE BELOW HERE IS CUSTOM FUNCTIONS WHICH

//ARE CALLED DURING THE PROGRAM OPERATION

//https://www.arduino.cc/en/Tutorial/PWM

void MotorControl(){

// Serial.print("\n");

// Serial.print("Ch2: ");

// Serial.print(Ch2Pwm);

//Motor 1

//If thumbstick is down

if(Ch2Pwm < Ch2MidPw && Ch2Pwm >= 1000){

M1\_Pwm = M\_Ratio \* ((Ch2MidPw) - (Ch2Pwm));

if (M1\_Pwm >= 240){

M1\_Pwm = 255;

}

else if(M1\_Pwm <= 20){//DEADBAND ZONE

M1\_Pwm = 0;

}

else{

M1\_Pwm = M1\_Pwm;

}

Serial.print("\n");

Serial.print("M1 Reverse: ");

Serial.print(M1\_Pwm);

Motor1Reverse();

}

//If thumbstick is in the middle

else if ((Ch2Pwm == Ch2MidPw or Ch2Pwm < 1000)){

M1\_Pwm = 0;

analogWrite(M1Pwm\_pin,M1\_Pwm);

}

//If thumbstick is up

else if (Ch2Pwm > Ch2MidPw){

M1\_Pwm = M\_Ratio \* ((Ch2Pwm) - (Ch2MidPw));

if (M1\_Pwm >= 240){

M1\_Pwm = 255;

}

else if(M1\_Pwm <= 30){//DEADBAND ZONE

M1\_Pwm = 0;

}

else{

M1\_Pwm = M1\_Pwm;

}

Serial.print("\n");

Serial.print("M1 Forward: ");

Serial.print(M1\_Pwm);

Motor1Forward();

}

else{

}

//Motor 2

//If thumbstick is down

if(Ch3Pwm < Ch3MidPw && Ch3Pwm >= 1000){

M2\_Pwm = M\_Ratio \* ((Ch3MidPw) - (Ch3Pwm));

if (M2\_Pwm >= 240){

M2\_Pwm = 255;

}

else if(M2\_Pwm <= 20){//DEADBAND ZONE

M2\_Pwm = 0;

}

else{

M2\_Pwm = M2\_Pwm;

}

Serial.print("\n");

Serial.print("M2 Reverse: ");

Serial.print(M2\_Pwm);

Motor2Reverse();

}

//If thumbstick is in the middle

else if ((Ch3Pwm == Ch3MidPw or Ch3Pwm < 1000)){

M2\_Pwm = 0;

analogWrite(M2Pwm\_pin,M2\_Pwm);

}

//If thumbstick is up

else if (Ch3Pwm > Ch3MidPw){

M2\_Pwm = M\_Ratio \* ((Ch3Pwm) - (Ch3MidPw));

if (M2\_Pwm >= 240){

M2\_Pwm = 255;

}

else if(M2\_Pwm <= 30){//DEADBAND ZONE

M2\_Pwm = 0;

}

else{

M2\_Pwm = M2\_Pwm;

}

Serial.print("\n");

Serial.print("M2 Forward: ");

Serial.print(M2\_Pwm);

Motor2Forward();

}

else{

}

}

//The code below is used for setting the dirction and speed of the track motors

//https://www.arduino.cc/en/Reference/digitalWrite

//https://www.arduino.cc/en/Reference/analogWrite

//Sets the direction of Motor1 to Forward

void Motor1Forward(){

digitalWrite(M1InA\_pin,HIGH);

digitalWrite(M1InB\_pin,LOW);

noInterrupts();

analogWrite(M1Pwm\_pin,M1\_Pwm);

interrupts();

}

//Sets the direction of Motor2 to Forward

void Motor2Forward(){

digitalWrite(M2InA\_pin,HIGH);

digitalWrite(M2InB\_pin,LOW);

noInterrupts();

analogWrite(M2Pwm\_pin,M2\_Pwm);

interrupts();

}

//Sets the direction of Motor1 to Reverse

void Motor1Reverse(){

digitalWrite(M1InA\_pin,LOW);

digitalWrite(M1InB\_pin,HIGH);

noInterrupts();

analogWrite(M1Pwm\_pin,M1\_Pwm);

interrupts();

}

//Sets the direction of Motor2 to Reverse

void Motor2Reverse(){

digitalWrite(M2InA\_pin,LOW);

digitalWrite(M2InB\_pin,HIGH);

noInterrupts();

analogWrite(M2Pwm\_pin,M2\_Pwm);

interrupts();

}

//Turns on the relay for the launcher motor

//Relay 1 is activated

void launcher(){

Serial.print("\n");

Serial.print("Ch5 PWM: ");

Serial.print(Ch5Pwm);

if (Ch5Pwm >= 1400){

digitalWrite(relay1,HIGH);

Serial.print("\n");

Serial.print("Launcher running");

}

else{

digitalWrite(relay1,LOW);

}

}

//Function to read the PWM signal from Channel 5

void Channel5Data(){

Ch5Pwm = pulseIn(Ch5\_pin,HIGH,2000);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INTERRUPT SERVICE ROUTINE FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//External ISRs are ideal to use because they are called whenever an external event happens

//This makes it unlikely for us to miss our signal. Otherwise we would have to constantly poll and check

//The input pins for signal changes.

//http://www.engblaze.com/we-interrupt-this-program-to-bring-you-a-tutorial-on-arduino-interrupts/

//ISR Function to read the PWM signal from Channel 1

void Channel1DataRising(){

Ch1RisingEdgeTime = micros();

attachInterrupt(digitalPinToInterrupt(Ch1\_pin),Channel1DataFalling,FALLING);

}

void Channel1DataFalling(){

Ch1FallingEdgeTime = micros();

Ch1TimeDif = Ch1FallingEdgeTime - Ch1RisingEdgeTime;

Ch1Pwm = Ch1TimeDif;

attachInterrupt(digitalPinToInterrupt(Ch1\_pin),Channel1DataRising,RISING);

}

//ISR Function to read the PWM signal from Channel 2

void Channel2DataRising(){

Ch2RisingEdgeTime = micros();

attachInterrupt(digitalPinToInterrupt(Ch2\_pin),Channel2DataFalling,FALLING);

}

void Channel2DataFalling(){

// Serial.print("Falling Edge ISR");

// Serial.print("\n");

Ch2FallingEdgeTime = micros();

Ch2TimeDif = Ch2FallingEdgeTime - Ch2RisingEdgeTime;

Ch2Pwm = Ch2TimeDif;

attachInterrupt(digitalPinToInterrupt(Ch2\_pin),Channel2DataRising,RISING);

}

//ISR Function to read the PWM signal from Channel 3

void Channel3DataRising(){

Ch3RisingEdgeTime = micros();

attachInterrupt(digitalPinToInterrupt(Ch3\_pin),Channel3DataFalling,FALLING);

}

void Channel3DataFalling(){

Ch3FallingEdgeTime = micros();

Ch3TimeDif = Ch3FallingEdgeTime - Ch3RisingEdgeTime;

Ch3Pwm = Ch3TimeDif;

attachInterrupt(digitalPinToInterrupt(Ch3\_pin),Channel3DataRising,RISING);

}

//ISR Function to read the PWM signal from Channel 4

void Channel4DataRising(){

Ch4RisingEdgeTime = micros();

attachInterrupt(digitalPinToInterrupt(Ch4\_pin),Channel4DataFalling,FALLING);

}

void Channel4DataFalling(){

Ch4FallingEdgeTime = micros();

Ch4TimeDif = Ch4FallingEdgeTime - Ch4RisingEdgeTime;

Ch4Pwm = Ch4TimeDif;

attachInterrupt(digitalPinToInterrupt(Ch4\_pin),Channel4DataRising,RISING);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*END OF INTERRUPT SERVICE ROUTINE FUNCTIONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*