Mechatronics 2017

MEAM 510
University of Pennsylvania

Lab 4 – FINAL PROJECT

Lab Due Date: 12/12/2017

Jeremy Saslaw

John Russ

Mechanical Design

The design intent and starting point for the design of the "Big Boi" robot was focused on overall utility rather than pure damage. A larger and heavier robot was opted for, with the purpose of controlling the high ground of the map and providing area of control that different teammates on the Meta Team could use to their advantage. With this in mind, the robot was designed with several key features that allowed for achieving these goals. Structurally, "Big Boi" was designed with a large front face intended for pushing, and a curved back that would be difficult to attack, as seen in Figure 1 below.

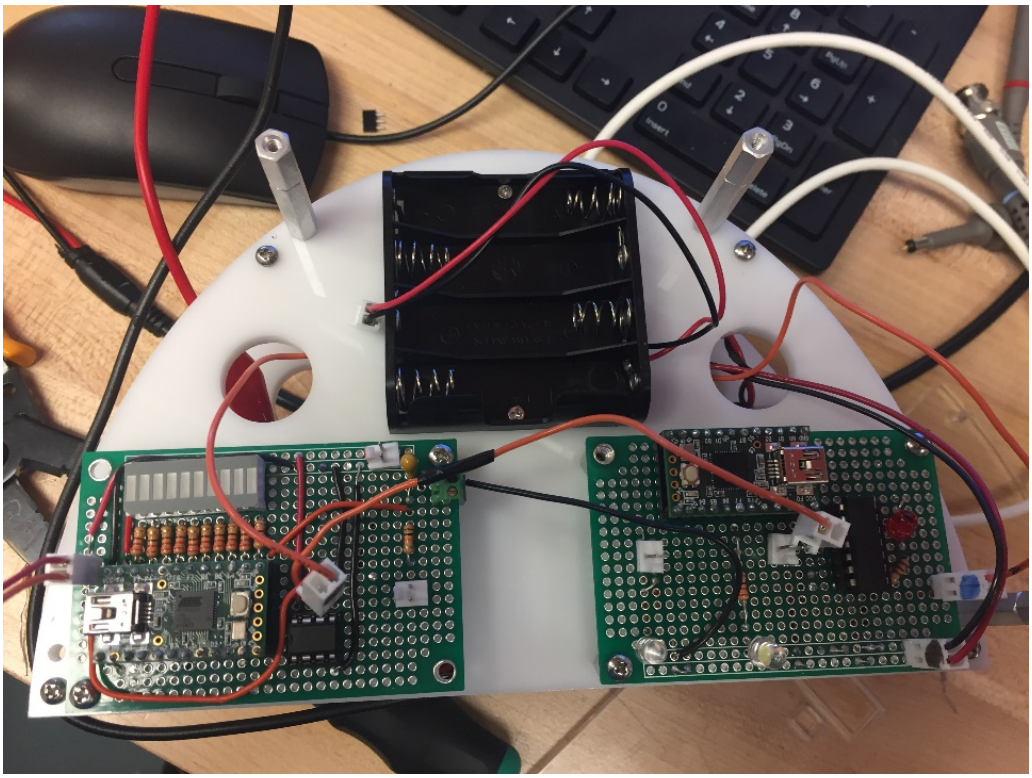
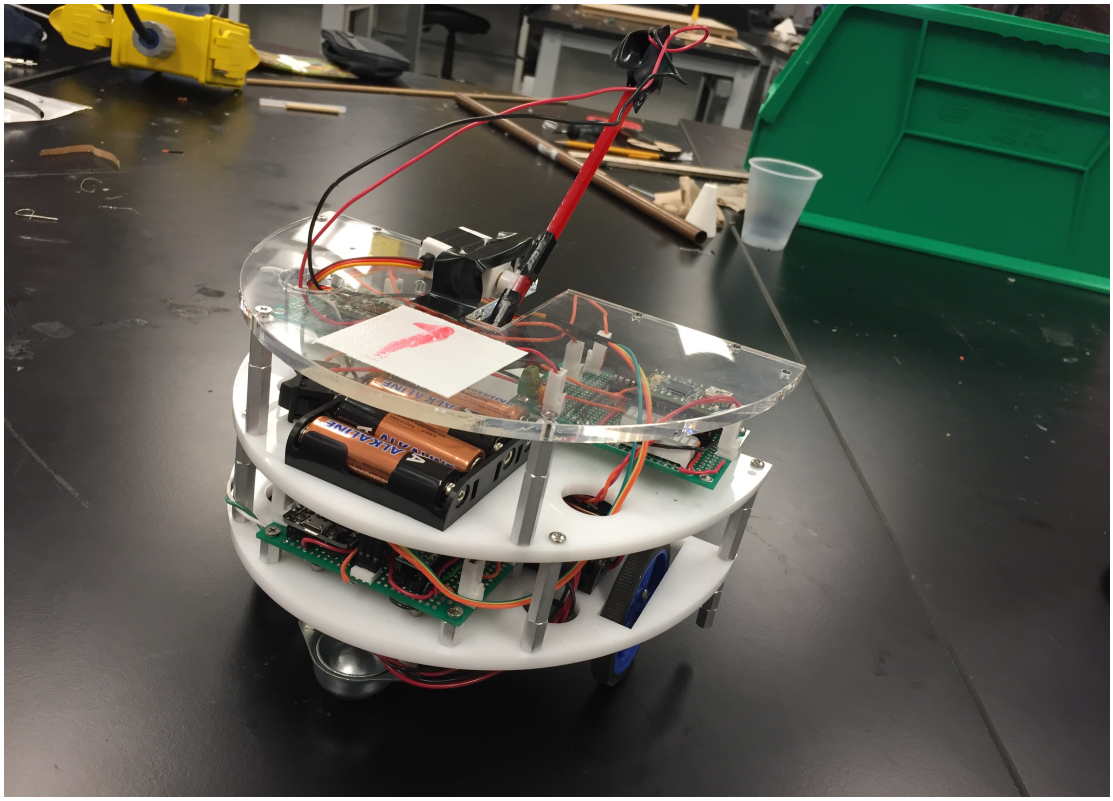


Figure 1 - Base design of Big Boi robot.

As seen above, the basis of the design was based on the iRobot Roomba's concept, except cut in half due to the space requirement. It was fully intended to make use of the maximum allowable space constraint, as Big Boi was 8 inches wide by 5 inches deep. As a result of this large size, steps needed to be taken to maintain maneuverability, where a differential drive similar to that of a Roomba was chosen. In this model, each wheel is capable of moving forwards and backwards independently, creating the ability to spin in place at maximum turning radii. Furthermore, extremely high-torque motors were chosen that would have no trouble churning their way up a hill, or through any opposition. In order to power these motors, motor drivers were required as up to 2.1 Amps of stall current had to be planned for. The final design of the Big Boi robot can be seen on the following page in Figure 2.



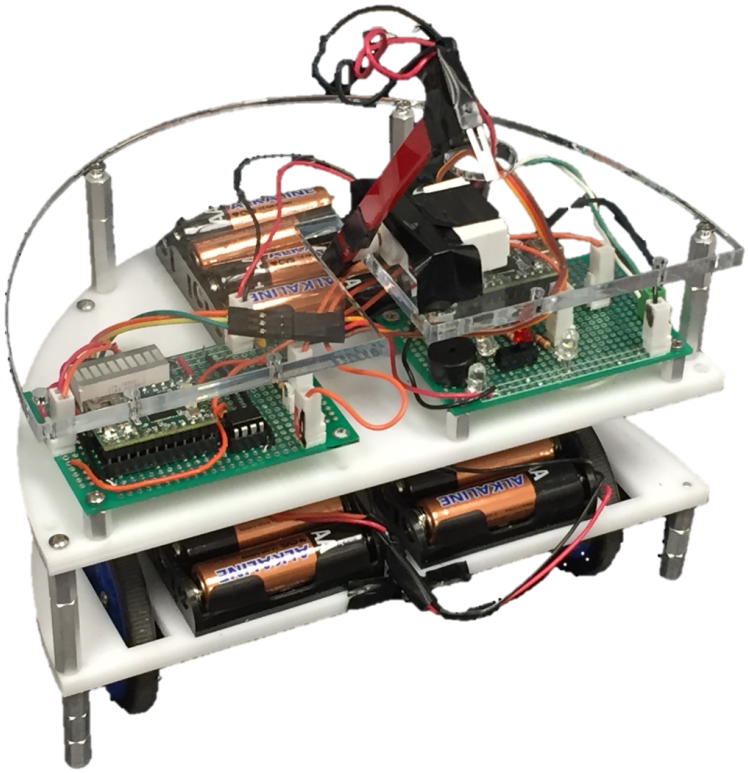


Figure 2 - Final design of Big Boi.

The other mechanical piece of the project is the controller which remotely controls the car via differential drive and a melee button. Inspiration for the controller design came from an Xbox controller, where there are two "handles" to hand on to, as well as a depth to the controller that makes for comfortable holding ability. Two slide potentiometers were implemented for differential drive of the robot, and a button was used to control the melee command. The final controller can be seen below in Figure 3:

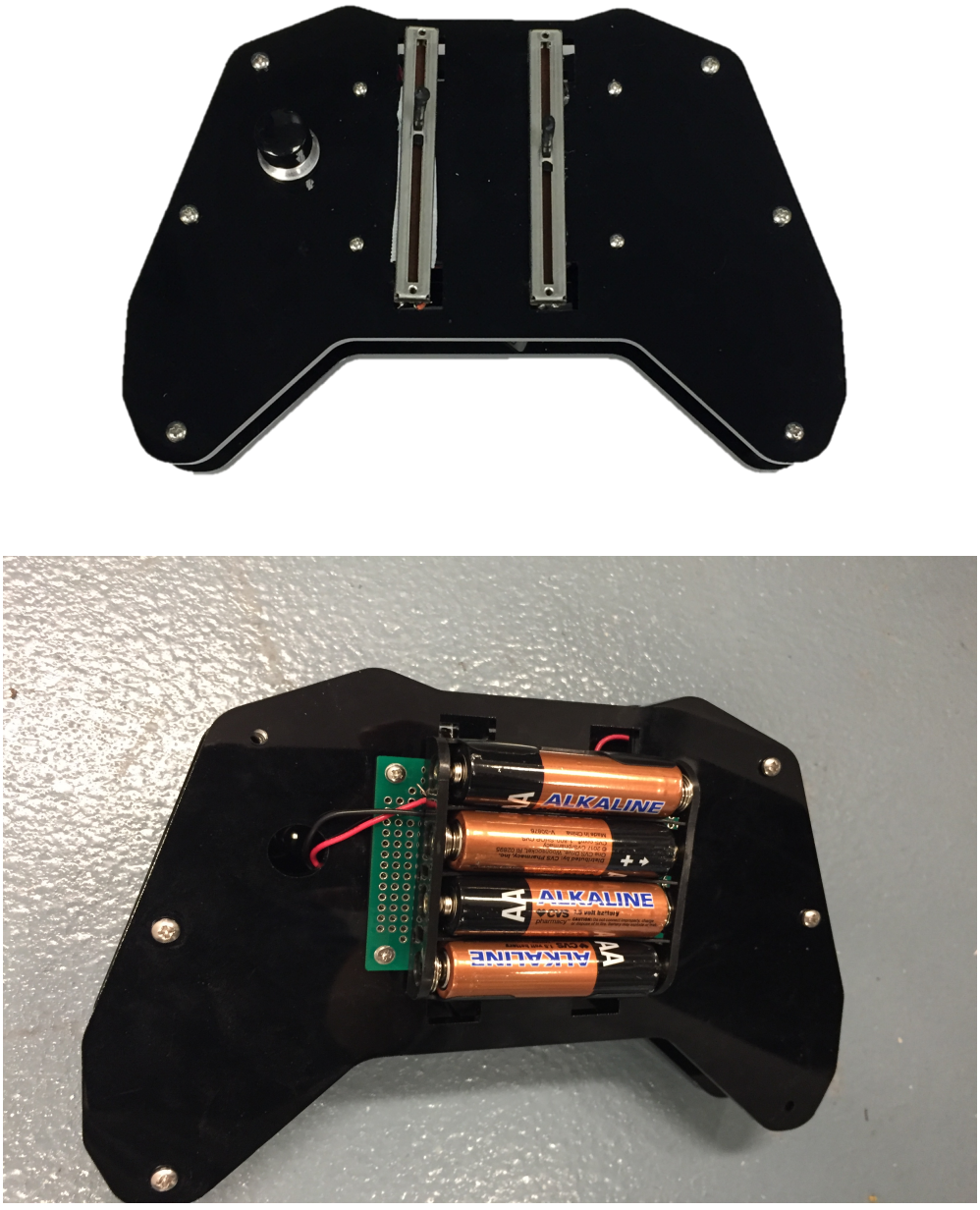


Figure 3 - Final controller design.

Electrical Design

For these powerful motors that were chosen, dedicated motor drivers which would deliver the necessary current were chosen. Additionally, these motors were most useful at higher voltages, and as a result, it was chosen to use two quad-AA-battery packs that supplied 12V in total when fully charged. With this choice came a design constraint though, the logic behind the motor drivers, in addition to the Teensy's cannot make use of such a high voltage, so instead of using a regulator to step down the voltage, it was decided to implement another dedicated energy source directly for all logic applications via a separate quad-AA-battery pack with 4.5 volts. A very high-level energy architecture can be below in Figure 4:

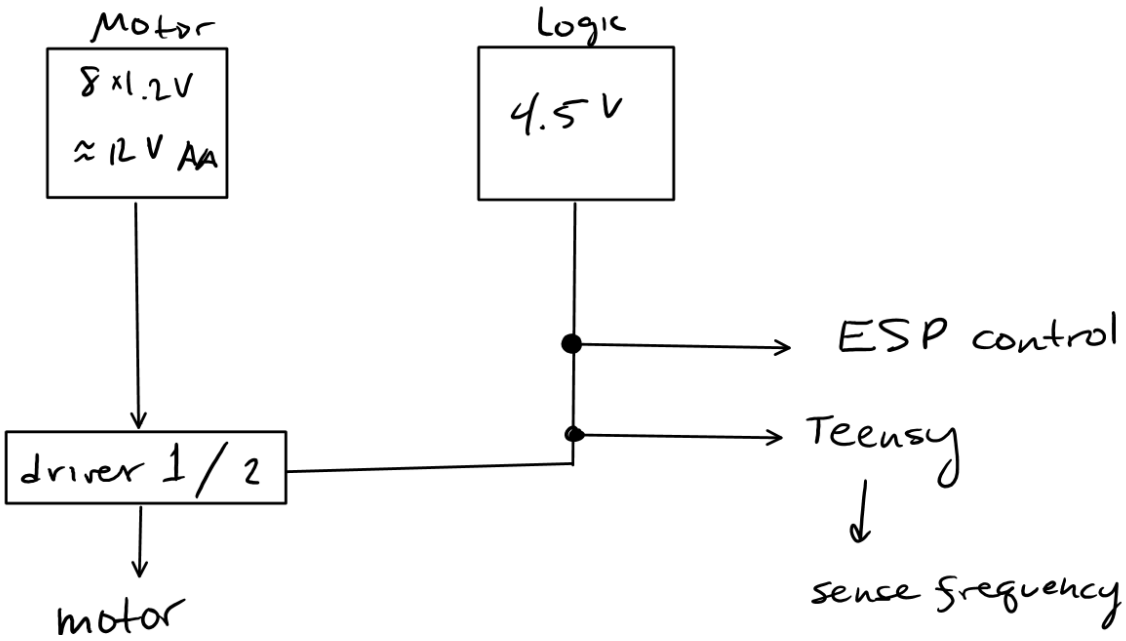


Figure 4 - Energy architecture of Big Boi.

It was determined that the use of separate battery systems was extremely useful as the motors were able to be isolated from the rest of the logic system, which helped to reduce noise and the risk of components breaking due to errors. To satisfy the various requirements of the competitions, it was chosen to use an ESP microcontroller as well as two Teensy microcontrollers to control various functions of the robot. This method created a wide number of GPIO pins on the ESP and I/O pins on the Teensy, as well as multiple options for ADC pins.

A high-level system architecture diagram can be seen on the following page in Figure 5, where each component has its own circuitry which will be displayed below in greater detail.

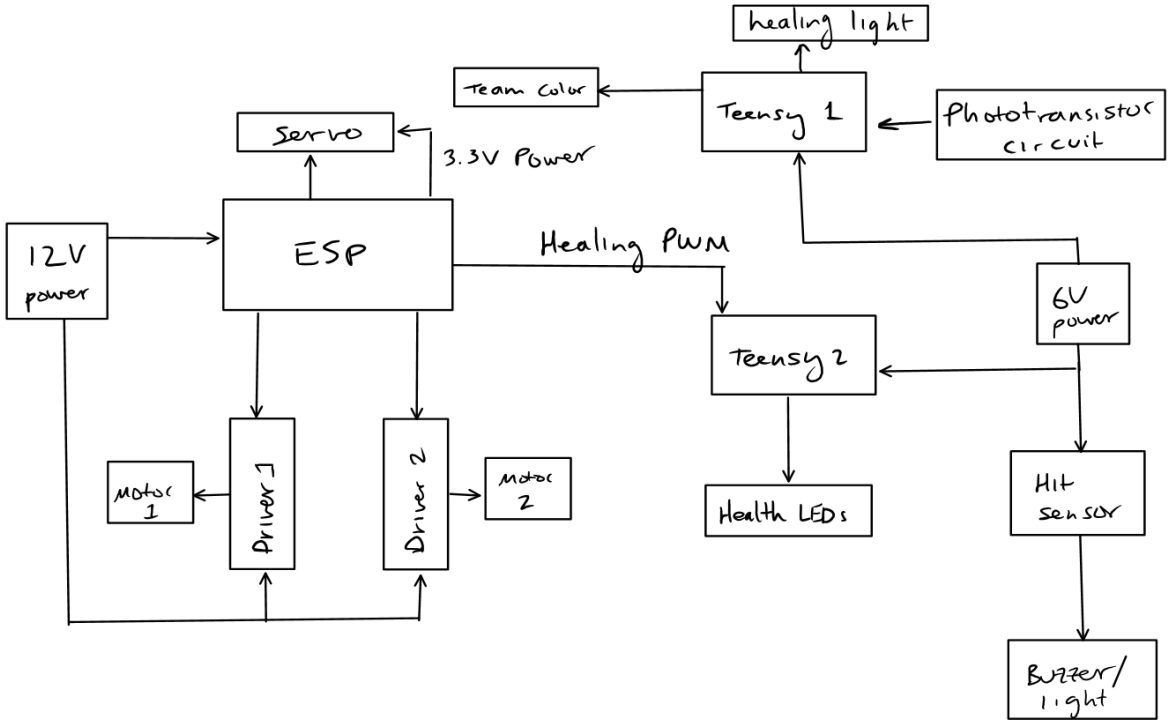


Figure 5 - System architecture of the Big Boi.

The phototransistor circuit block is expanded below in Figure 6. A Teensy 2.0's ADC pin was utilized for this purpose, in addition to an amplification circuit. As part of the healing circuit, an additional port was utilized to power an LED which indicated when healing was occurring.

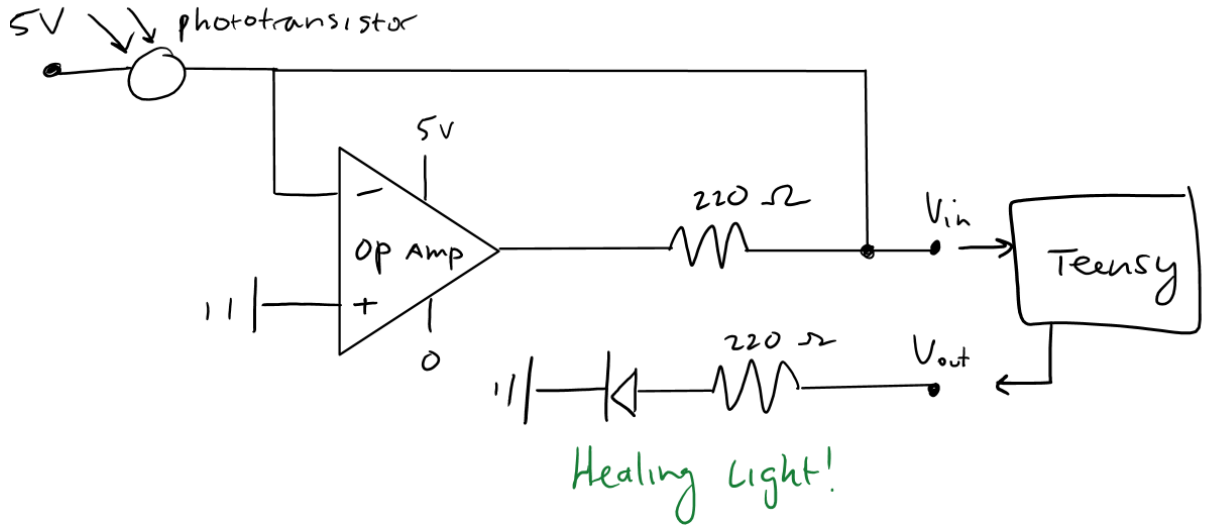


Figure 6 - Frequency sensing circuit for healing.

A key component of Big Boi was the ability to communicate between the ESP and one of the Teensy circuits. In order to do this, a PWM signal with a low pass filter was chosen rather than SPI communication. Specifically, because the ESP was the primary logic board which received the health signal from the overall router, this signal needed to be transferred to the Teensy 1, which handled lighting up the health bar. A mapped out ADC value was sent to the Teensy, which would pass through this low pass filter seen below in Figure 7, which averaged the voltage of the PWM and turned that into a ratio out of 100 which would indicate the amount of health that the robot should display.

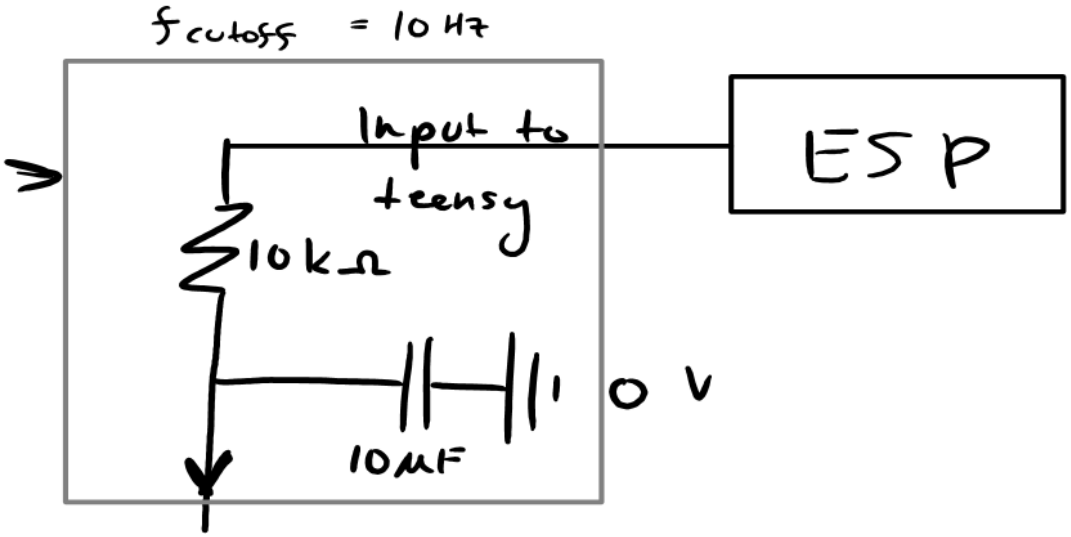


Figure 7 - ESP to Teensy signal transferring diagram.

An additional key block from the high-level system diagram is the health boards circuit, which can be seen below in Figure 8. The Teensy's many I/O pins were taken advantage of, where each LED of the 10-stage LED bar corresponded with an I/O pin, allowing for full control of the light bar.

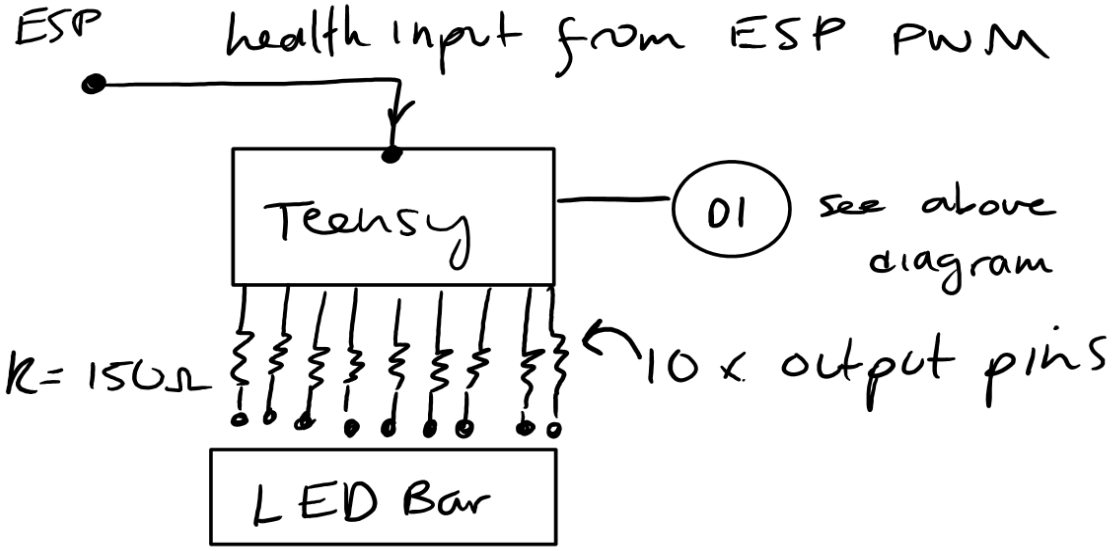


Figure 8 - Health Bar LED circuit.

Furthermore, for the servo and hit sensing portion of our system, a signal from the ESP was utilized that could be relayed from the controller ESP to the robot. As a part of this circuit seen in Figure 5, an LED and a buzzer was hooked up in series with a switch from ground, and thus when this hit sensor registered a hit, the LED and buzzer became enabled as well.

Finally, the motor driver circuit can be seen below in Figure 9 (taken from its datasheet). The motor driver purchased is capable of driving two motors, but in that case, it can only handle 1.2A per motor. As specified on its data sheet, one of these Pololu 2135 motor drivers can be used to drive a single motor at a stall current of 2.4A by paralleling the outputs, which is how they were utilized for the Big Boi design.

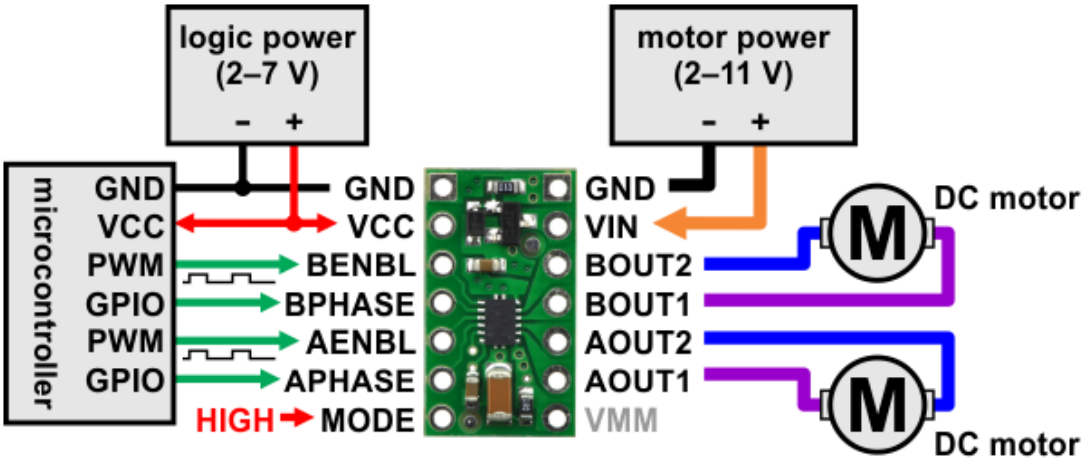


Figure 9 - Motor driver circuit diagram.

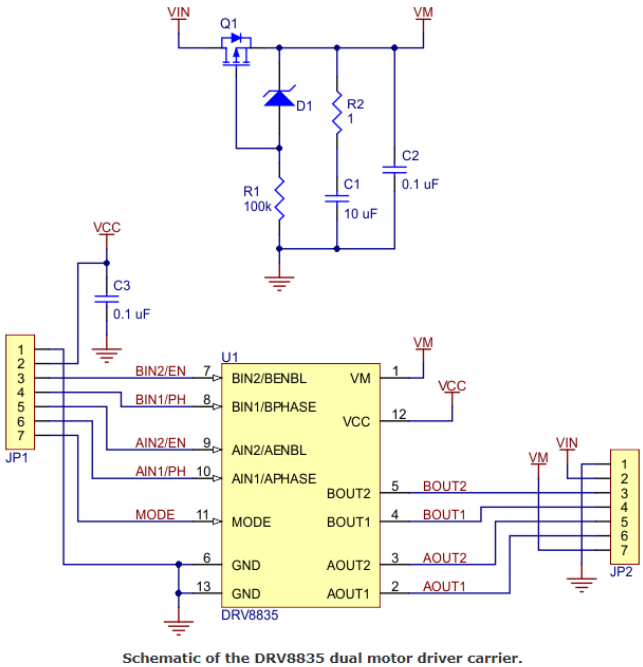


Figure 10 - Schematic of the DRV8835 dual motor driver carrier.

Electrical

Bill of Materials

Table 1 - Bill of Materials

Wheels	Pololu Wheel 60 x 8 mm Pair (1)_
Wheel mounts	Pololu 20D mm Metal Gearmotor Bracket Pair (1)
Wheel mounting hub	Pololu Universal Aluminum Mounting Hub for 4mm Shaft, #4-40 Holes (2)
Motor Driver	DRV8835 Dual Motor Driver Carrier (2)
Motor	100:1 Metal Gearmotor 20Dx44L mm 6V (2)
Battery	Duracell AA Battery (12)
Servo motor	SERVOMOTOR 5VDC TOWERPRO SG92R (1)
Wifi microprocessor	ESP (1)
Additional microprocessor	Teensy (2)

Extra Datasheets, Specs, and Schematics

Pololu 20D x 44L mm 6V motor
Datasheet: https://www.pololu.com/product/3457

Dimensions

Size:	20D × 44L mm
Weight:	45 g
Shaft diameter:	4 mm

General specifications

Gear ratio:	100:1
Free-run speed @ 6V:	140 rpm
Free-run current @ 6V:	170 mA
Stall current @ 6V:	2.9 A[1]
Stall torque @ 6V:	90 oz·in[1]
Extended motor shaft?:	N
Motor type:	2.9A stall @ 6V

Pololu 2135 DRV8835 Motor Driver
Datasheet: https://www.pololu.com/file/0J570/drv8835.pdf

General specifications

Motor driver:	DRV8835
Motor channels:	2
Minimum operating voltage:	0 V
Maximum operating voltage:	11 V
Continuous output current per channel:	1.2 A[2]
Peak output current per channel:	1.5 A
Continuous paralleled output current:	2.4 A[2]
Maximum PWM frequency:	250 kHz
Minimum logic voltage:	2 V
Maximum logic voltage:	7 V
Reverse voltage protection?:	Y

TowerPro SG92R Servo Motor
Datasheet: http://www.towerpro.com.tw/product/sg92r-7/

SG92R

SG92R
TowerPro 9g digital servo New version of SG90.
POM with Carbon fiber gears set and control arms are more
durable use than SG90 nylon gear and more torque.
It is a good choice for park size 3D models , high speed jets and
R/C airplane 250 450 RC helicopter

Specifications:

Weight: 9g
Dimension: 23×12.2x27mm
Stall torque: 2.5kg /cm(4.8v)
Gear type: POM with carbon fiber
Operating speed: 0.1sec /60degree (4.8v)
Operating voltage: 4.8v
Temperature range: 0°C_ 55°C
Dead band width:1us
servo wire length: 25 cm
Servo Plug: JR (Fits JR and Futaba)
servo arms &screws included and fit with Futaba servo arm
It's universal "S" type connector that fits most receivers, including
Futaba, JR, Hitec ,GWS, Cirrus, Blue Bird, Blue Arrow, Corona,
Berg, Spektrum.
CE &RoHS approved

Software Implementation and Code

There are four stages of software implementation, each which handles various components:

1. **ESP board on controller**
    a. Senses the user inputs such as each slide potentiometer for each wheel's speed and direction and the melee button.
    b. Sends this signal via Wifi to the ESP board on the robot.
2. **ESP board on robot**
    a. Receives the inputs from the ESP board on controller.
    b. Handles both motor drivers (and motors) as well as the Servo.
    c. The main handler of logic, such as parsing the health signal.
3. **Teensy 1 on robot**
    a. Handles the phototransistor and lights a corresponding LED to signal health.
4. **Teensy 2 on robot**
    a. Receives an ADC signal through a low-pass filter from the ESP board on robot which tells this Teensy how much health to display, and displays the corresponding health on a health bar.
    b. Also handles the red or blue team light, as well as the hit button, LED, and buzzer.

Some of the more integral and not-direct software implementations will be discussed below, after which all of the code for each board will be displayed.

First, the robot drove via differential drive, where each slide potentiometer determines the direction and speed of the corresponding wheel. The software behind this uses mapping tools on the controller, where the top half of the slide potentiometer corresponds with forwards movement mapped from not moving to full speed ahead, while the back half has the same implementation for moving backwards. Additionally, a center area on the pot was designated as the no moving zone, to allow for ease of use to stop the car.

Another crucial portion of the code involved sensing frequency. In order to accomplish this, an internal timer on the Teensy was utilized which would interrupt the program every 0.1 seconds. Throughout the main loop, the ADC input signal was used to count how many rises and falls (corresponding to one peak) there were on the incoming signal. Once the interrupt occurred, an ISR was implemented which multiplied the amount of counts by 10 (thus dividing by 0.1 seconds), which created a frequency reading.

1 – ESP board on controller

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>

//set the ssid and pass of the wifi
const char* ssid = "metateam3"; //"modlab1";
const char* pass = "YayFunFun"; //"ESAP2017";

//for sending
IPAddress ipSendto(192,168,1,20);
unsigned int udpRemotePort = 1497;
unsigned int udplocalPort = 1490;
const int UPD_PACKET_SIZE = 6;
char udpBuffer[UPD_PACKET_SIZE];
WiFiUDP udp;

//construct a boolean that will switch eZvery iteration of the loop
//this determines whether we send power to pot 1 or pot 2 for each of the two motors
boolean firstMotor = false;

//string value for the team color
char teamColor = 'r';

//melee or not
boolean melee = false;

//set up everything and wifi
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);
  delay(10);

  //connect to wifi
  Serial.print(ssid);
  WiFi.begin(ssid,pass);
  WiFi.config(IPAddress(192,168,1,63),
            IPAddress(192,168,1,1),
            IPAddress(255,255,255,0));
  while (WiFi.status()!=WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }

  //check if wifi connected
  Serial.println("WiFi Connected");
  Serial.print("IP address: ");
  Serial.println(WiFi.localIP());

  //start the UDP connection
  Serial.println("Starting UDP");
  udp.begin(udplocalPort);
  Serial.print("Local Port:");
  Serial.println(udp.localPort());
```

```
  //Set up the pinmode for ADC1 and ADC2 signals
  pinMode(A0, INPUT);

  //set the pinMode for the power of the two pots
  pinMode(D0, OUTPUT);

  //set the pinMode for the team color
  pinMode(D7, INPUT);

  //set the pinMode for the melee
  pinMode(D8, INPUT);
}

//note that we have two potentiometers that independently control the PWM of the two motors
//this is used for differential driving
//this code allows us to send two ADCs to a single ADC port, A0, on our ESP.
void loop() {
  int ADC1 = 0;
  int ADC2 = 0;

  //delay to not clog up wifi
  delay(50);

  //get the ADC values from the controller.
  //if we want to get the value from the first motor
  if (firstMotor){

    //provide power to the first ADC through inverter (using NAND gates)
    digitalWrite(D0, LOW);

    //read the ADC value
    ADC1 = analogRead(A0);

  } else{ //we want to get power from the second ADC

    //provide power to the other DDC through inverter
    digitalWrite(D0, HIGH);

    //get the reading from second ADC
    ADC2 = analogRead(A0);
  }

  //get whether the teamcolor is red or blue.
  //If red, switch is on
  //If blue, switch is off
  int teamColorValue = digitalRead(D7);
  if (teamColorValue == 1) {
    teamColor = 'b';
  } else {
    teamColor = 'r';
  }

  //get whether we want to melee or not
  int meleeValue = digitalRead(D8);
  if (meleeValue == HIGH) { // no melee
    melee = true;
```

```
  } else { //melee
    melee = false;
  }

  //actually send the information to the other ESP
  if (firstMotor){
    //send the ADC value associated with the first motor to the car.
    //note that we're sending one ADC value at a time during each iteration of the loop.
    //the second parameter tells us that this ADC is associated with the first motor.
    udpSend(String(ADC1), 1, melee, teamColor);
  }else{
    //send the ADC values associated with the second motor
    //note that we're sending one ADC value at a time during each iteration of the loop.
    //the second parameter tells us that this ADC is associated with the first motor.
    udpSend(String(ADC2), 2, melee, teamColor);
  }

  //reverse firstMotor so next iteration of the loop, we get the value of the other ADC
  firstMotor = !firstMotor;
}


/**
 * Send the String toSend to the ipSendTo
 * General Structure: 6 bits
 * [ADC, ADC, ADC, motorValue, melee, teamColor]
 * [0,   1,   2,   3,          4,     5]
 */
void udpSend(String ADC, int motorValue, boolean melee, char teamColor){
  //send the ADC
  for (int i = 0; i < ADC.length(); i++){
    udpBuffer[i] = (char) ADC[i];
  }

  //Add 'A's to denote if ADC is less than three digits
  int zeros = UPD_PACKET_SIZE - (ADC).length()-3;
  for (int i = ADC.length(); i < (zeros+ADC.length()); i++){
    udpBuffer[i] = (char) 'A';
  }

  //send which motor it is
  if (motorValue == 1){
      udpBuffer[UPD_PACKET_SIZE-3] = 'x';
  } else if (motorValue ==2){
    udpBuffer[UPD_PACKET_SIZE-3] = 'y';
  }

  //send the melee
  if (melee){
    udpBuffer[UPD_PACKET_SIZE-2] = 'm'; //melee
  } else{
    udpBuffer[UPD_PACKET_SIZE-2] = 'n'; //no melee
  }

  //send the teamColor
  udpBuffer[UPD_PACKET_SIZE-1] = teamColor; //melee
```

```
  udp.beginPacket(ipSendto,udpRemotePort);
  udp.write(udpBuffer, sizeof(udpBuffer));
  udp.endPacket();

  Serial.print("Sending!: ");
  Serial.println(udpBuffer);
}

/**
 *  Read the UDP
 */
String udpRead(){
  String packetFromOther = "";
  int c = udp.parsePacket();
  if (c){
    udp.read(udpBuffer, UPD_PACKET_SIZE);
    for (int i = 0; i < UPD_PACKET_SIZE; i++){
      packetFromOther += (char) udpBuffer[i];
    }
    Serial.print("Receiving! ");
    Serial.println(packetFromOther);
    Serial.println(".");
  }
  return packetFromOther;
}
```

2 – ESP board on robot

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <Servo.h>

//set up the wifi
const char* ssid = "metateam3"; //"modlab1";
const char* pass = "YayFunFun"; //"ESAP2017";

//set up local and remote ports and packet sizes
IPAddress ipSendto(192,168,1,63);
unsigned int udpRemotePort = 1490;
unsigned int udplocalPort = 1497;
const int UPD_PACKET_SIZE = 6;
char udpBuffer[UPD_PACKET_SIZE];
WiFiUDP udp;

//initialize the wifi for the health signals
WiFiUDP udpHealth;
unsigned int udpHealthRemotePort = 2390;
const int UPD_PACKET_SIZE_HEALTH = 22;
byte udpBufferHealth[UPD_PACKET_SIZE_HEALTH];

//initialize values of the motors
int motor1ADC;
int motor2ADC;

//initialize the melee values and team color values
boolean melee = false;
int meleeCounter = 0;

char teamColor = 'r';

//initialize the health, cooling period
int health = 99;

//implement cooldown period
int startTime = 0;
int endTime = 0;
int coolingPeriod = 3400;

//initially you can attack
boolean coolDownPassed = true;

//initialize the servo for melee
Servo myservo;

//runs once intially to set up wifi and pinModes
void setup() {
  // put your setup code here, to run once:
  Serial.begin(115200);

  //connect to wifi
  Serial.print(ssid);
```

```
WiFi.begin(ssid,pass);
WiFi.config(IPAddress(192,168,1,20),
          IPAddress(192,168,1,1),
          IPAddress(255,255,255,0));
while (WiFi.status()!=WL_CONNECTED){
  delay(500);
  Serial.print(".");
}

//check if wifi connected
Serial.println("WiFi Connected");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());

//start the UDP connection
Serial.println("Starting UDP");
udp.begin(udplocalPort);
Serial.print("Local Port:");
Serial.println(udp.localPort());

//start the UDP connection for the health
Serial.println("Starting UDP Health");
udpHealth.begin(2390);
Serial.print("Local Port Health:");
Serial.println(udpHealth.localPort());

//Motor 1
//Set up the pinmode for motor left direction pin
pinMode(D1, OUTPUT);
//set up pinmode for speed of first  motor
pinMode(D2, OUTPUT);
//set the direction of the first motor
digitalWrite(D1, LOW);

//Motor 2
//Set up the pinmode for motor  direction pin
pinMode(D7, OUTPUT);
//set up pinmode for speed of second  motor
pinMode(D8, OUTPUT);
//set the direction for the second motor
digitalWrite(D7, LOW);

//initialize the pinMode for the servo
myservo.attach(D0);

//initialize the pinMode for the team color light
pinMode(D5, OUTPUT);

//intialize the pinMode for sending health to Teensy to display
pinMode(D6, OUTPUT);

//initialize the pinMode for digital read of red or blue light
pinMode(D3, INPUT);

//initialize the pinMode for sending digital signal if servo moving or not
pinMode(D4, OUTPUT);
```

```
  myservo.write(180);

}

//runs over and over
void loop() {
  int ADC1; //set to ADC1 value from the UDP
  int ADC2; //set to ADC2 value from the UDP

  //get the ADC values from the controller
  String ADCReads = udpRead();

  //find the length of the String ADCReads.
  //Note that this is because ADC values can range from 0 to 4 digits. In our code,
  //an A shows that the ADC value is less than 4 digits
  int indexA = ADCReads.indexOf('A');

  //instantiate variables that we'll parse from the ADC reads
  //this is an ADC value from ADCReads
  int ADCvalue;
  //motorValue tells us 'x' for the first motor and 'y' for the second motor.
  char motorValue;

  //get the ADC values and which motor it corresponds
  //in the case that the ADC value is four chars, there will be no 'A' in our ADCReads string
  if (indexA == -1){
    //We then take the substring and convert it to an int.
    ADCvalue = ADCReads.substring(0,3).toInt();

    //the motorValue ('x' or 'y' is always the fourth char in ADCReads from the controller.
    motorValue = (char) ADCReads[3];

  } else{
    //in the case that the ADC value is less than four chars, there will be an 'A' in the ADCReads
    //And so we find the substring of the string to find the ADC value and convert it to an int.
    ADCvalue = ADCReads.substring(0,indexA).toInt();

    //the motorValue ('x' or 'y' is always the fourth char in ADCReads from the controller.
    motorValue = (char) ADCReads[3];
  }

  //get the values for first and second motor
  if (motorValue == 'x') {//first motor
    //set the motor1ADC value to actuate as the ADCValue parsed from ADCReads.
    motor1ADC = ADCvalue;

  } else if (motorValue = 'y'){//second motor
    //set the motor2ADC value to actuate as the ADCValue parsed from ADCReads.
    motor2ADC = ADCvalue;
  }


  //get whether melee is being implemented
  if (ADCReads[4] == 'm'){
    melee = true;
```

```
  meleeCounter = meleeCounter + 1;
} else if (ADCReads[4] == 'n'){
  meleeCounter = 0;
  melee = false;
}

//get the team color
teamColor = (char) ADCReads[5];

//get the health signal
String healthReads = udpReadHealth();
//String healthReads = "RnnR1R2R3R4BnnB1B2B3B4"
//String healthReads = "Rnn50505050Bnn50505050";
if (healthReads.length() > 0){
  String healthForTeam = "";
  if (digitalRead(D3) == LOW){
    healthForTeam = healthReads.substring(0,11);
  } else{
    healthForTeam = healthReads.substring(11,22);
  }

  //get the health dependent on the vehicle number
  health = healthForTeam.substring(3,5).toInt();
}

//send the health signal to the Teensy
//convert the health to a 0 to 3.3V signal
double maxHealth = 99.0;
int healthToSendTeensy = (int) (1023.0 * (health / maxHealth));
analogWrite(D6, healthToSendTeensy);


//make the robot go
//only go if the health is greater than zero.
if (health > 0){
  //set the max ADC value from the controller
  int maxADCValue = 418;

  //create a multiplier since our pots on the controller only go from 0 to 400.
  //since we want the full resolution of 0 to 1000, we multiply all the ADC values by 2.3
  //before actuating.
  double multiplier = 2*1023.0/(maxADCValue);

  //to create the range of zero in the middle
  int range = 20;

  //send the sinals to the motor if and only if  both motors have ADCs from controller
  if (motor1ADC > 0 && motor2ADC > 0){
    //first motor
    int toWriteFirst = 0;
    //figure out direction of motor
    if (motor1ADC > (int) maxADCValue/2){
      //motor direction low
      digitalWrite(D1, LOW);
      //the value of toWriteFirst = the ADC from the controller - 1/2 the max
      toWriteFirst = motor1ADC - maxADCValue/2;
```

```
  } else{
    //switch the motor direction to go other way
    digitalWrite(D1, HIGH);
    toWriteFirst = abs(motor1ADC-maxADCValue/2);
  }
  //create the ranges of zero.
  if (toWriteFirst < range){
    toWriteFirst = 0;
  }
  //set the value = to 1/2 of the controller value * the multiplier
  toWriteFirst = (int) toWriteFirst*multiplier;
  //write the scaled adc value to the first motor.
  analogWrite(D2,toWriteFirst);
  Serial.println(toWriteFirst);

  //second motor
  int toWriteSecond = 0;
  //figure out direction of motor
  if (motor2ADC > (int) maxADCValue/2){
    //motor direction stays low
    digitalWrite(D7, LOW);
    //the value of toWriteFirst = the ADC from the controller - 1/2 the max
    toWriteSecond = motor2ADC - maxADCValue/2;
    //create the ranges of zero.
  } else{
    //switch the motor direction to go other way
    digitalWrite(D7, HIGH);
    toWriteSecond = abs(motor2ADC-maxADCValue/2);
  }
  if (toWriteSecond < range){
    toWriteSecond = 0;
  }
  //set the value = to 1/2 of the controller value * the multiplier
  toWriteSecond = (int) toWriteSecond*multiplier;
  //write the scaled adc value to the second motor.
  analogWrite(D8,toWriteSecond);
}

//display the light of the team color
//may have to switch high and low depending on which color gets inverter
if (teamColor == 'r'){
  digitalWrite(D5, LOW);
} else if (teamColor == 'b'){
  digitalWrite(D5, HIGH);
}

// Tell digital pin servo is NOT moving
digitalWrite(D4, LOW);
//do the meleeing and display that you are meleeing
if (melee){
  //check if time between startTime and endTime > coolingPeriod
  coolDownPassed = ((millis() - startTime) >= coolingPeriod);
  //if the cooldown period has passed
  if (coolDownPassed && meleeCounter < 2){
    // Tell digital pin servo is moving
    digitalWrite(D4, HIGH);
```

```
      //reset the start time for the next iteration
      startTime = millis();
      //make the melee arm go
      myservo.write(70);
      //make the servo stay at max point for a bit
      delay(2000);
      //come back to initial point
      myservo.write(180);
      //make the servo stay at max point for a bit
      delay(500);
      melee = false;
     }

   }

  }
}


/**
 *  Read the UDP
 */
String udpRead(){
  String packetFromOther = "";
  int c = udp.parsePacket();
  if (c){
    udp.read(udpBuffer, UPD_PACKET_SIZE);
    for (int i = 0; i < UPD_PACKET_SIZE; i++){
      packetFromOther += (char) udpBuffer[i];
    }
    Serial.print("Receiving! ");
    Serial.println(packetFromOther);
    Serial.println(".");
  }
  return packetFromOther;
}


String udpReadHealth(){
  String health = "";
  int c = udpHealth.parsePacket();
  if (c){
    udpHealth.read(udpBufferHealth, UPD_PACKET_SIZE_HEALTH);
    for (int i = 0; i < UPD_PACKET_SIZE_HEALTH; i++){
      health += (char) udpBufferHealth[i];
    }
    Serial.print("Receiving Health! ");
    Serial.println(health);
    Serial.println(".");
  }
  return health;
}
```

3 – Teensy 1 on robot

```
// Code for Teensy with phototransistor on it with corresponding health LED

#include "teensy_general.h"
#include "t_usb.h"

volatile int count = 0;
volatile int frequency = 0;
int a = 0;
int b = 0;

int main(void) {

  // Enable global interrupts
  sei();

  // Initialize pins
  clear(DDRF,0);      // Phototransistor
  clear(DDRF,4);      // Receives digital signal from first LED for alive or dead
  clear(DDRB,1);   // Digital read for if servo is moving
  set(DDRF,6);        // Receiving Health
  set(DDRB,4);        // Pin for team light

  // Initialize USB connection
  m_usb_init();

  ////////////////////////////////// Timer 3 //////////////////////////////////////

  // Set clock divide
  teensy_clockdivide(0);

  // Turn on clock source to prescaler of 64
  clear(TCCR3B,CS32);
  set(TCCR3B,CS31);
  set(TCCR3B,CS30);

  // Timer mode - Mode 4 (Normal)
  clear(TCCR3B,WGM33);
  set(TCCR3B,WGM32);
  clear(TCCR3A,WGM31);
  clear(TCCR3A,WGM30);

  // Set frequency to 10 Hz
  OCR3A = 25000;

  // Set Interrupt portion
  set(TIMSK3,OCIE3A);

  ////////////////////////////////////////////////////////////////////////////////

  ////////////////////////////////// ADC Portion //////////////////////////////////

  // Set voltage reference to Vcc
  clear(ADMUX,REFS1);
```

```
 set(ADMUX,REFS0);

 // Set ADC Prescaler to 128 --> clock speed should be 125 kHz
 set(ADCSRA,ADPS2);
 set(ADCSRA,ADPS1);
 set(ADCSRA,ADPS0);

 // Disable digital inputs
 set(DIDR0,ADC0D);

 // Set trigger to free running mode
 set(ADCSRA,ADATE);

 // Set ESP (F0 ADC0)
 clear(ADCSRB,MUX5);
 clear(ADMUX,MUX2);
 clear(ADMUX,MUX1);
 clear(ADMUX,MUX0);

 // Enable ADC subsystem
 set(ADCSRA,ADEN);

 // Begin conversion
 set(ADCSRA,ADSC);

 ////////////////////////////////////////////////////////////////////////////

 for(;;){

   if(bit_is_set(PINF,4) != 0) { set(PORTB,4); }
   else { clear(PORTB,4); }

   a = ADC;

   // Wait .05 ms or 20 kHz
   teensy_wait(.05);

   b = ADC;

   // If rising edge
           if ((b-a) > 5) {

     // If falling edge
     while((b-a) > -5) {
       a = ADC;
       teensy_wait(.05);
       b = ADC;
     }

     count = count + 1;
   }

 }

 return 0;   /* never reached */
```

```
}

// Interrupt every .1 seconds (when OCR3A=25000)
ISR(TIMER3_COMPA_vect){

  frequency = 10*count;

  if (frequency > 120 && frequency < 330 && bit_is_set(PINB,1) == 0)
  {
            set(PORTF,6);
  }
  else
  {
            clear(PORTF,6);
  }

  m_usb_tx_uint(frequency);
  m_usb_tx_string("\n");

  count = 0;

}
```

4 – Teensy 2 on robot

```
// Code for Teensy with heatlh bar

#include "teensy_general.h"
#include "t_usb.h"

int main(void) {

   // Initialize pins
   clear(DDRD,6);            // ESP
   // Outputs
   set(DDRB,0);              // Health bar LEDs
   set(DDRF,1);
   set(DDRF,4);
   set(DDRF,5);
   set(DDRF,6);
   set(DDRF,7);
   set(DDRB,6);
   set(DDRB,5);
   set(DDRB,4);
   set(DDRD,7);

   // TESTER
   //set(DDRB,1);
   // TESTER

  // Initialize USB connection
  m_usb_init();

  ////////////////////////////////// ADC Portion //////////////////////////////////

  // Set voltage reference to Vcc
  clear(ADMUX,REFS1);
  set(ADMUX,REFS0);

  // Set ADC Prescaler to 128 --> clock speed should be 125 kHz
  set(ADCSRA,ADPS2);
  set(ADCSRA,ADPS1);
  set(ADCSRA,ADPS0);

  // Disable digital inputs
  set(DIDR2,ADC9D);

  // Set trigger to free running mode
  set(ADCSRA,ADATE);

  // Set ESP (D6 ADC9)
  set(ADCSRB,MUX5);
  clear(ADMUX,MUX2);
  clear(ADMUX,MUX1);
  set(ADMUX,MUX0);

  // Enable ADC subsystem
  set(ADCSRA,ADEN);
```

```
 // Begin conversion
 set(ADCSRA,ADSC);

 ////////////////////////////////////////////////////////////////////////////

 for(;;){

  // TESTER
  //set(PORTB,1);
  // TESTER

  int health = ADC;

  m_usb_tx_uint(health);
  m_usb_tx_string("\n");

  if (health >= 1) { set(PORTB,0); }
  else { clear(PORTB,0); }

  if (health >= 67) { set(PORTF,1); }
  else { clear(PORTF,1); }

  if (health >= 134) { set(PORTF,4); }
  else { clear(PORTF,4); }

  if (health >= 203) { set(PORTF,5); }
  else { clear(PORTF,5); }

  if (health >= 270) { set(PORTF,6); }
  else { clear(PORTF,6); }

  if (health >= 330) { set(PORTF,7); }
  else { clear(PORTF,7); }

  if (health >= 345) { set(PORTB,6); }
  else { clear(PORTB,6); }

  if (health >= 358) { set(PORTB,5); }
  else { clear(PORTB,5); }

  if (health >= 366) { set(PORTB,4); }
  else { clear(PORTB,4); }

  if (health >= 375) { set(PORTD,7); }
  else { clear(PORTD,7); }

 }

 return 0;   /* never reached */

}
```