ILLectric Skateboard

Joseph Ravichandran, Kanad Sarkar, Rachel Fu, Taiqing Ling

Kanads2, Jpr3, Yunhanf2, taiqing2

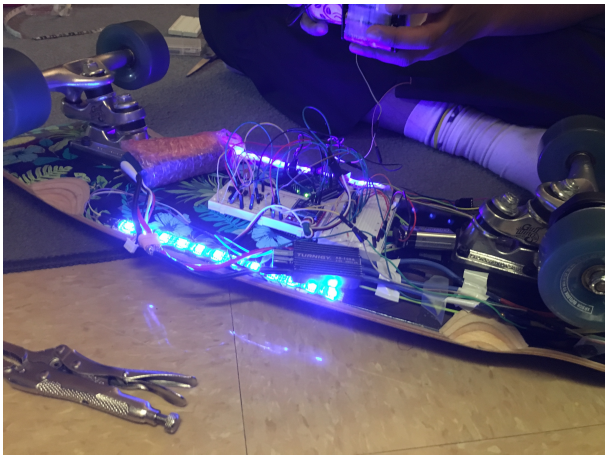
Introduction

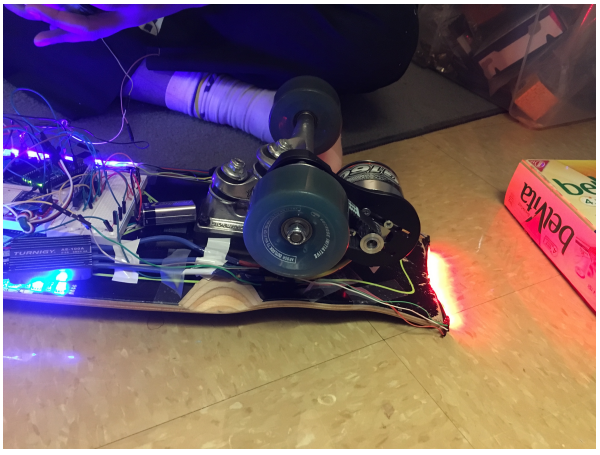
Statement of Purpose

An integral part of society, especially that of a college student in UIUC, is trying to find a viable source of transportation that is fast, energy efficient, compact and affordable. One of the more popular methods amongst our students is biking, but owning a bike means that the user will need to take time out of their life to constantly maintain their bike, as well as find a whole storable space for their bike when not in use. Our solution to this transportation problem is the Illectric Skateboard, an electric skateboard that is more affordable than current electric skateboards already on the market. This board allows its user to go from class to class without worry of where you should store it, while at the same time competes with bikes in regards to both speed and control. The remote control should allow for control with close to zero latency, and be connected to lights on the skateboard so that this mode of transportation is better suited for on the road traffic.

Features

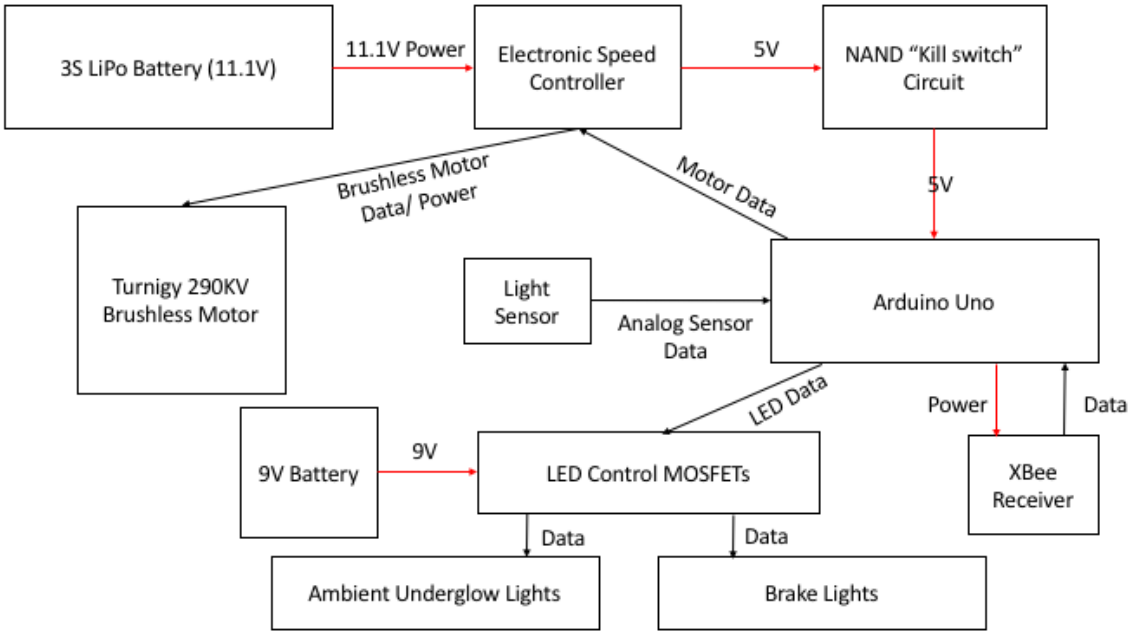
Some features of the board include a remote control that connects to the skateboard via a XBEE radio, which allows for direct connection using radio frequency signals with no latency. Another important feature is the floor lights, which are programmed to come on whenever the surrounding is dark. This is important so the rider is always able to potentially avoid potential cracks or hazards on the road. The skateboard is also equipped with turn and brake signals, which not many other electric skateboards have, yet it is essential when the user might be riding alongside cars and other big vehicles. The skateboard can go up to 30 mph, and has a continuous run time of two hours, which is enough to last for one school day. The Illectric Skateboard also have double trucks on the wheels, which allows for better turning and control.





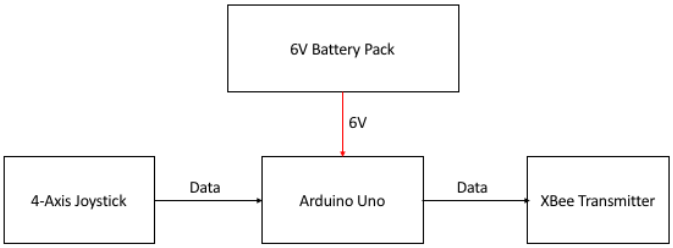
System Overview

On-Board System:



On the skateboard itself, most of the circuitry is controlled through an Arduino Uno. There are 2 batteries, 1 to power the Arduino and motor, and another to power the lights. Data is received from the transmitter through the XBee Receiver, and is then converted into a form readable by the ESC in the Arduino. The ESC controls the motor, and regulates the power of the LiPo battery to 5V, which is run through the NAND killswitch circuit and then into the Arduino VIN port.

Remote Control System:



The remote control is relatively minimalistic, using only 1 sensor and 1 output (the light from the XBee transmnitter). The Arduino converts input from the joystick into an encoding, which is then sent to the receiver XBee to control the skateboard.

Design Details

On-Board System:

Electronic Speed Controller: The purpose of the ESC is twofold. First, it regulates the voltage from the LiPo battery to a steady 5V that powers the NAND kill switch circuit, and then the Arduino. The second purpose is to take data back in from the Arduino and power the motor. Interestingly enough, the values accepted by the ESC can be easily written through the Servo.h library for Arduino. While the motor is not a servo, we can control it as one.

Turnigy 290KV Brushless Motor: A brushless motor is controlled using 3 wires instead of the 2 wire control that DC motors use. The brushless motor uses 3 wires because it needs to activate different magnets at different points. A brush motor uses brushes along the armature to control the switching of magnets that allows the rotor to turn, whereas the brushless motor requires an ESC to control this switching.

The NAND “Kill switch” circuit allows us to instantly disconnect the Arduino from power safely at any given moment. This circuit was instrumental during the testing and development of the board.

Truth Table:

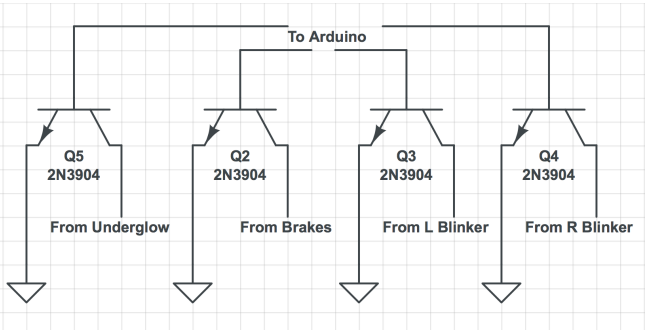
Input 1	Button	Output of NAND
0	0	1
0	1	1
1	0	1
1	1	0

The “Input 1” value is held high as long as the circuit is powered, so ignore cases when Input 1 = 0.

Functional Table:

Input	Output
Button Up	+5V
Button Down	GND

LED Controller MOSFETs:



This is an array of 4 MOSFET NPN transistors to control the brightness of various lights. The transistors are controlled using pulse-width modulation signals from the Arduino. The four signals being controlled are underglow, brake lights (both sides), left blinker, and right blinker.

The LEDs feature 4 pins: +12V, R, G, and B. We directly connected the 9V battery to the +12V power line (9V is sufficient to power these LEDs for our purposes), and then connected the ground of the LEDs to the transistors. When the Arduino turns the transistors on, the ground pins from the lights are connected to ground, and the lights turn on. The circuit all shares a common ground.

Motor speed calculations:

The wheel can turn at a theoretical max RPM of 9657 RPM.

$$Motor\ RPM = 290 * Volts$$

$$Motor\ RPM = 290 * 11.1 = 3219\ RPM$$

$$Wheel\ RPM = \frac{Pulley\ 1}{Pulley\ 2} * Motor\ RPM = \frac{36}{12} * 3219 = 9657\ RPM$$

Controller: The controller consists of an Arduino Uno, an Adafruit Joystick, and a XBEE radio. The code for the Arduino Uno is Code 1. The XBEE radio communicates with a baud rate of 9600, which the rate of the XBEE receiver on the skateboard. The joystick is programmed to have the board speed up or slow down a constant amount when moved up or down, when the joystick is pulled down, the brake lights also turn on. The turning signals can also be turned on by toggling the joystick left or right. When nothing is toggled on the joystick, the skateboard will maintain its speed and all of the lights will be turned off.

Results

Our final deliverable is a fully functional electric skateboard and wireless controller. The skateboard features a 290 kV motor and a 100 Amp electronic speed controller, allowing for drive speeds of up to approximately 25 miles per hour. The skateboard itself was built from a combination of parts of several skateboards, selected individually to create a board optimized for fast travel. As a result, the board is able to overcome initial load-amps and accelerate the user from a complete stop. This feature is not found on many commercially available longboards, setting our product apart.

Electronically, the board features 4 individual LED zones. It features underglow which automatically is enabled in dark environments, allowing the user to avoid cracks easier. The board also features brake lights that automatically engage whenever the user begins to slow down, and a turn signal that can be activated using the remote. The board also features a “kill switch” on the bottom that controls power to the Arduino, which in turn gives a signal to the ESC to turn the motor. The kill switch is controlled via a NAND gate with 1 pin held high. The second pin is controlled by a button. When the button is pressed, power is cut from the Arduino and the motor will immediately stop.

Using the skateboard is a very pleasant experience. Despite its appearance, the remote is very comfortable to hold in the hand and functions very well. Accelerating has been fine-tuned in software to allow a comfortable pace of starting and stopping without feeling like falling. The skateboard went through multiple iterations of testing before its final design. We tested it in several different environments that the user is likely to use the skateboard in. The first test was the ‘bump’ test, where the board was ridden at a fast speed (approximately 75% of the maximum speed) over a bump in the floor. At first, this test caused the board to momentarily disconnect from power due to the jarring on the ESC. We were able to fix this issue by solidifying the connection between the Arduino and the ESC. The second test was the outdoor speed test. During this stage, the prototype electronics were on top of the board instead of mounted to the bottom. During the first run of this test, after hitting a large crack the electronics were ejected out of the container box and fell into the street. After repairing this issue, the board was able to perform successfully outside. During this test we reached speeds of approximately 20 miles per hour. Our third test was the sensor test. We rode the board in and out of dark and bright areas, and the underglow lights responded accordingly.

The skateboard can be ridden for about 45 minutes without requiring a recharge, although during that time speeds will noticeably decrease until the battery is dead.

Problems and Challenges

A significant problem plaguing our design is cable management. While the circuit functions flawlessly with the board upside down, in a real world trial wires falling out or components becoming disconnected is quite common. To attempt to solve this problem, we created a 3D printed enclosure to put the electronics in (See Appendix Drawing 1), but this only made the problem worse as it would put pressure on the wires, causing them to come loose. Another significant challenge for this project was programming the ESC. The ESC requires a very specific signal, generated by a very specific transceiver device. We are using an Arduino instead of a radio transmitter, so we had to figure out how to program the ESC without a joystick to control it. The joystick we are using does not work for programming the ESC, since it can only detect up or down, and therefore requires an Arduino to parse the signal and format it for the ESC. We were able to program the ESC through careful time calculations and trial and error. After it was calibrated to read our signal, we wrote code to accelerate the output value of the Arduino gradually based on the remote. As a result, the joystick doesn't function as a typical joystick does in a RC device, but instead will change the value of the motor over time. This is because varying degrees of input are not available on this joystick. These challenges were successfully overcome in the end.

Another major challenge of our design was the mechanical side. Mounting the motor proved to be exceedingly difficult. Our first idea was to mount the motor directly to the board, but this would result in the timing belt being pulled and pushed violently as the user turns the skateboard, which could cause it to snap. Our second idea was to mount the motor directly to the axle with a permanent offset so that the motor's pulley gear and the wheel's pulley gear are always in line, no matter how much the user turns the board. To actually do this was quite challenging. We ordered a motor mount that never showed up, so during the last week we ordered one from Amazon with 2 day shipping. Unfortunately this motor mount did not fit the axle, so Joseph went to his uncle's shop over the weekend and machined the part until it fit the axle. This process was quite difficult because it involved a significant amount of trial and error, and ended up requiring drilling directly through the solid skateboard trucks. In the end, this method proved to work, and the motor mount is so strong that removing it would be essentially impossible.

Future Plans

In the future we want to 3D print a more comfortable controller to enclose the electronics in. Our original idea was to use a Wii remote nunchuck to enclose the electronics, but sadly the Arduino and Xbee radio were too large to fit into this enclosure. Instead, we decided to use a coffee cup to put it all together, which worked quite well, but could be better. We also want to implement a smaller LiPo battery on the controller, and add a MicroUSB port for charging the controller. Additionally, we want to replace the Arduino Uno with an Arduino Nano, and solder the rest of the circuit directly to it. Lastly, we want to acquire a more accurate joystick which allows for greater precision when controlling the board.

On the skateboard side, we want to design a PCB that includes the MOSFETs for the LEDs, the Xbee radio, and the NAND gate for disabling the board. We want to double the battery storage capacity by adding a second LiPo battery in parallel with the current one to double our run time (our run time at the moment is approximately 45 minutes). We want to enclose both batteries with the circuitry, and add a charging port and an on/ off switch instead of requiring the user to plug the battery in. We also want to waterproof the circuitry using caulk along the edge of the enclosure. Lastly, we want to find a more secure solution for the LEDs, as at the moment the pins frequently become disconnected. This solution could involve soldering or purchasing LEDs with better connections, as our first attempt to solder to these LEDs failed within a few hours. We also want to look into embedding the LEDs into the board itself instead of just sticking them on the outside. Lastly, we want to look into the possibility of adding a second motor for more power.

References

[1]"How to make an electric skateboard", *How to make an electric skateboard*, 2017. [Online]. Available: http://www.howtomakeanelectricskateboard.com. [Accessed: 28- Sep- 2017].

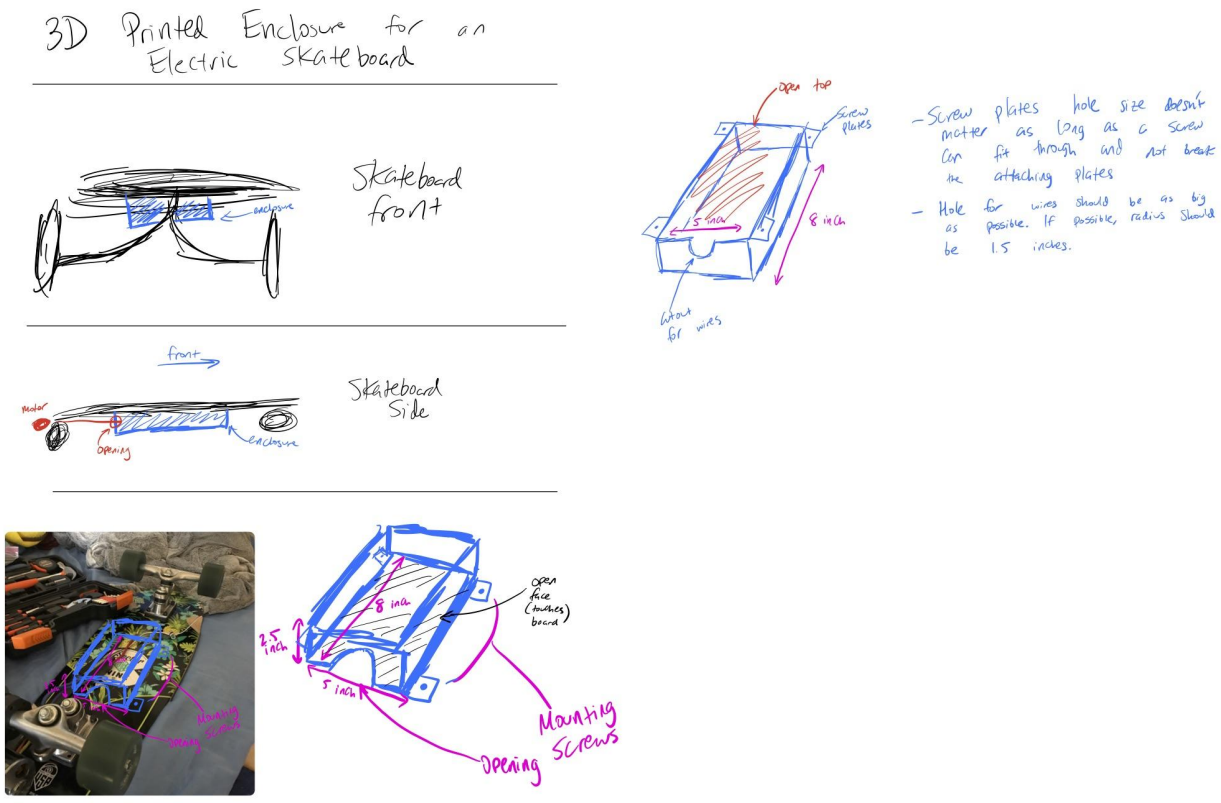
[2]V. Pomogaev, "Electric Skateboard V2.0: Smartphone Controlled", *Instructables.com*, 2017. [Online]. Available: http://www.instructables.com/id/Electric-Skateboard-v20/. [Accessed: 26- Oct- 2017].

[3]S. Lumetta, *ECE 120: Introduction to Computing Course Notes*. Champaign: University of Illinois at Urbana-Champaign, 2017.

[4]"How to Use XBee Modules As Transmitter & Receiver - Arduino Tutorial", *Instructables.com*, 2017. [Online]. Available: http://www.instructables.com/id/How-to-Use-XBee-Modules-As-Transmitter-Receiver-Ar/. [Accessed: 19- Dec- 2017].

Appendix

Drawing 1: Schematics for the 3D Printed Enclosure.



Code 1: The Remote Control Program

```
int IUP = A0;
int ILR = A1;
int sw_pin = 9;

void setup() {
  // put your setup code here, to run once:
  pinMode(IUP, INPUT);
  pinMode(ILR, INPUT);
  pinMode(sw_pin, INPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
```

```
  int vertical = analogRead(IUP);
  int horizontal = analogRead(ILR);
  //Serial.println(vertical);
  //Serial.println(digitalRead(sw_pin));
  if (digitalRead(sw_pin) == 1)
  {
    if (vertical > 400 && vertical < 700 && horizontal > 400 && horizontal < 700)
    {
      Serial.println("<0>");
    }
    else if (vertical < 400)
    {
      Serial.println("<1>");
    }
    else if (vertical > 700)
    {
      Serial.println("<2>");
    }
    else if (vertical > 400 && vertical < 700 && horizontal > 700)
    {
      Serial.println("<3>");
    }
    else if (vertical > 400 && vertical < 700 && horizontal < 400)
    {
      Serial.println("<4>");
    }
  }
  else if (digitalRead(sw_pin) == 0)
  {
    Serial.println("<5>");
    digitalWrite(sw_pin, HIGH);
  }
  delay (250);
}
```

Code 2: The Skateboard Program

```
#include <Servo.h>

Servo motor;

//Constants
const int LIGHT_PIN = A1; // Pin connected to voltage divider output
const long tickTime = 300;

bool leftTurn = false;
bool rightTurn = false;
bool floodlights = false;
bool braking = false;

bool leftTurnOn = false;
```

```
bool rightTurnOn = false;

unsigned long prevTickL = 0;
unsigned long prevTickR = 0;


// Measure the voltage at 5V and the actual resistance of your
// 47k resistor, and enter them below:
const float VCC = 4.98; // Measured voltage of Ardunio 5V line
const float R_DIV = 4660.0; // Measured resistance of 3.3k resistor

// Set this to the minimum resistance require to turn an LED on:
const float DARK_THRESHOLD = 10000.0;
int motorSpeed = 0;
int rightBlinker = 7;
int leftBlinker = 8;
int floorBlinker = 6;
int brakeBlinker = 5;
//Variables
bool started= false;//True: Message is strated
bool ended   = false;//True: Message is finished
char incomingByte ; //Variable to store the incoming byte
char msg[3];       //Message - array from 0 to 2 (3 values - PWM - e.g. 240)
byte index;              //Index of array

bool isON = false;

int timer = 0;

void setup() {

      //Start the serial communication
      Serial.begin(9600); //Baud rate must be the same as is on xBee module
  motor.attach(9);
      pinMode(rightBlinker, OUTPUT);
  pinMode(leftBlinker, OUTPUT);
  pinMode(floorBlinker, OUTPUT);
  pinMode(LIGHT_PIN, INPUT);
  pinMode(floorBlinker, OUTPUT);
  pinMode(brakeBlinker, OUTPUT);
  SetSpeed(0);
  delay(6000);
  SetSpeed(0);
}

void loop()
{
      if (leftTurn) {
    if (millis() - prevTickL > tickTime) {
      digitalWrite (leftBlinker, leftTurnOn);
      leftTurnOn = !leftTurnOn;
      prevTickL = millis();
```

```
  }
}
else {
  digitalWrite (leftBlinker, LOW);
}

if (rightTurn) {
  if (millis() - prevTickR > tickTime) {
    digitalWrite (rightBlinker, rightTurnOn);
    rightTurnOn = !rightTurnOn;
    prevTickR = millis();
  }
}
else {
  digitalWrite (rightBlinker, LOW);
}

if (floodlights) digitalWrite (floorBlinker, HIGH);
else digitalWrite (floorBlinker, LOW);

if (braking) digitalWrite (brakeBlinker, HIGH);
else digitalWrite (brakeBlinker, LOW);


    int lightADC = analogRead(LIGHT_PIN);
if (lightADC > 0)
{
  // Use the ADC reading to calculate voltage and resistance
  float lightV = lightADC * VCC / 1023.0;
  float lightR = R_DIV * (VCC / lightV - 1.0);
  Serial.println("Voltage: " + String(lightV) + " V");
  Serial.println("Resistance: " + String(lightR) + " ohms");

  // If resistance of photocell is greater than the dark
  // threshold setting, turn the LED on.
  if (lightR >= DARK_THRESHOLD)
    floodlights = true;
  else
    floodlights = false;

  Serial.println();
  delay(50);
}
while (Serial.available()>0){
    //Read the incoming byte
  incomingByte = Serial.read();
  //Start the message when the '<' symbol is received
  if(incomingByte == '<')
  {
   started = true;
   index = 0;
   msg[index] = '\0'; // Throw away any incomplete packet
```

```
  }
  //End the message when the '>' symbol is received
  else if(incomingByte == '>')
  {
    ended = true;
    break; // Done reading - exit from while loop!
  }
  //Read the message!
  else
  {
    if(index < 1) // Make sure there is room
    {
      msg[index] = incomingByte; // Add char to array
      index++;
      msg[index] = '\0'; // Add NULL to end
    }
  }
}

if(started && ended)
{
  int value = atoi(msg);

  if (value == 0) //set speed
  {
     rightTurn = false;
     leftTurn = false;
     braking = false;
     delay(150);
  }
  else if (value == 1)
  {
   //digitalWrite(motorPinFast, HIGH);
     IncreaseSpeed(40);
     delay(150);
     rightTurn = false;
     leftTurn = false;
     braking = false;
  }
  else if (value == 2)
  {
     //digitalWrite(motorPinSlow, HIGH);
     DecreaseSpeed(150);
     braking = true;
     rightTurn = false;
     leftTurn = false;
     delay(150);
  }
  else if (value == 3)
  {
   rightTurn = true;
   braking = false;
```

```
    leftTurn = false;
    delay(150);
   }
   else if (value == 4)
   {
    leftTurn = true;
    rightTurn = false;
    braking = false;
    delay(150);
   }


 }

   //Serial.println(value); //Only for debugging
   index = 0;
   msg[index] = '\0';
   started = false;
   ended = false;

}



void SetSpeed (int sp)
{
  motor.write(map (sp, 0, 1000, 700, 2000));
}

void IncreaseSpeed(int x)
{
  motorSpeed += x;
  if (motorSpeed < 1000)
  {
    SetSpeed(motorSpeed);
  }
  else
  {
    SetSpeed(1000);
    motorSpeed = 1000;
  }
}

void DecreaseSpeed(int x)
{
  motorSpeed -= x;
  if (motorSpeed > 0)
  {
    SetSpeed(motorSpeed);
  }
  else
  {
```

```
    SetSpeed(0);
    motorSpeed = 0;
  }
}
```

Code 3: Motor Calibration

```
/* Calibrate
 *  Calibrates ESC by first setting PWM to max (255) and then setting it to 0 when
the user presses a button
 *  The ESC will go "beep beep" when the high range is detected, then you press
the button
*/

#include <Servo.h>

Servo myservo;  // create servo object to control a servo

void setup() {
  pinMode (13, OUTPUT);
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
  digitalWrite(13, HIGH); // Turn on indicator light to show we are calibrating
  myservo.write (2000); // Write high value
  delay (4000);  // Wait for ESC to recognize high value
  myservo.write (700); // Write low value
  digitalWrite (13, LOW); // Turn off indicator light, we are calibrated
}

void loop() {
}
```

Code 4: LED Controller Over Serial

```
bool leftTurn = false;
bool rightTurn = false;
bool floodlights = false;
bool braking = false;

bool leftTurnOn = false;
bool rightTurnOn = false;

unsigned long prevTickL = 0;
unsigned long prevTickR = 0;

const long tickTime = 300;

// Pins:
// 10: Left turn
// 11: Right turn
// 12: Underglow

void setup() {
  // put your setup code here, to run once:
  pinMode (9, OUTPUT);
```

```
  pinMode (10, OUTPUT);
  pinMode (11, OUTPUT);
  pinMode (12, OUTPUT);
  Serial.begin (9600);
}

void loop() {
  if (Serial.available() > 0) {
    char in = Serial.read();
    if (in == 'A') leftTurn = !leftTurn;
    if (in == 'B') rightTurn = !rightTurn;
    if (in == 'C') floodlights = !floodlights;
    if (in == 'D') braking = !braking;
  }

  if (leftTurn) {
    if (millis() - prevTickL > tickTime) {
      digitalWrite (10, leftTurnOn);
      leftTurnOn = !leftTurnOn;
      prevTickL = millis();
    }
  }
  else {
    digitalWrite (10, LOW);
  }

  if (rightTurn) {
    if (millis() - prevTickR > tickTime) {
      digitalWrite (11, rightTurnOn);
      rightTurnOn = !rightTurnOn;
      prevTickR = millis();
    }
  }
  else {
    digitalWrite (11, LOW);
  }

  if (floodlights) digitalWrite (12, HIGH);
  else digitalWrite (12, LOW);

  if (braking) digitalWrite (9, HIGH);
  else digitalWrite (9, LOW);
}
```

Code 5: Control Motor Over Serial

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int value = 700;

void setup() {
```

```
  pinMode (13, OUTPUT);
  myservo.attach(9);  // attaches the servo on pin 9 to the servo object
  digitalWrite(13, HIGH);
  myservo.write (700);
  delay (6000);
  myservo.write (700);
  digitalWrite (13, LOW);
  Serial.begin (9600);
}

void loop() {
  if (Serial.available()) {
    value = Serial.parseInt();
  }
  myservo.write (value);
}
```