Bluetooth Controlled Car

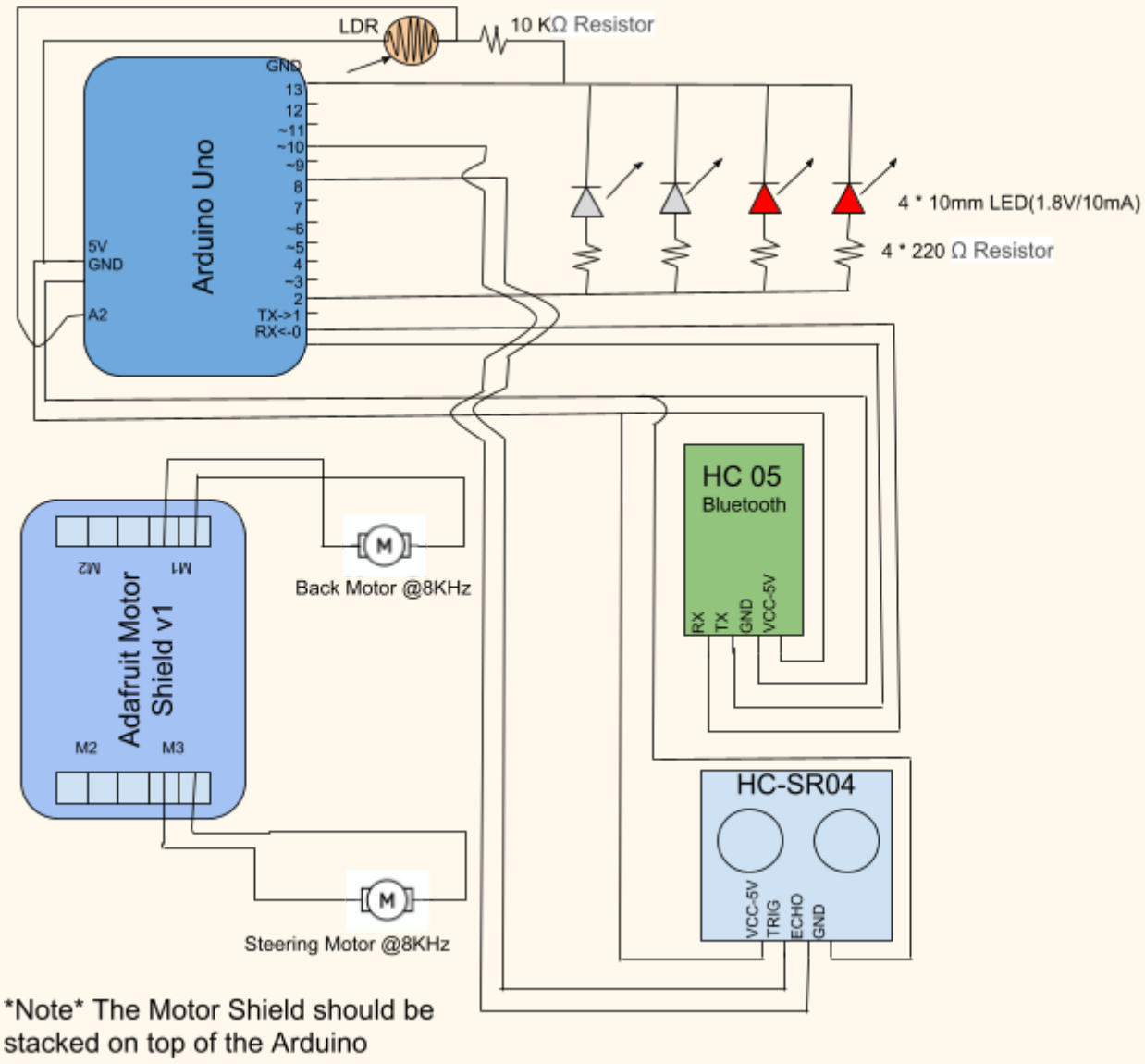
By Allen Chen & Yunas Magsi

INTRODUCTION

The purpose for our experiment was to develop a Remote Control Car (RC Car). It couldn't just be a regular RC Car. This RC Car mainly is so attractive that if you were a kid and saw that at the store, you would beg your parents to get it for you. It will have the bells and whistles. The bells and whistles include car lights, smart braking; so the car doesn't get damaged that easily, and the best of all, it's all controlled by your very own phone. As the user, you are in control of your vehicle at all times. The app has a fairly simple layout but offers more button for more precise choices.

Schematic: Process

To set your very own RC Car up, the following schematic shows the layout from top to bottom:



DIY Your Own RC Car

Before even getting into the steps to building the ultimate RC Car, we need to gather the materials we need for the project. The list of materials required are:

Before beginning the DIY, here is the list of electronic parts and tools required to build your very own RC Car:

- RC Car with 2 DC motor

- Arduino UNO
- Adafruit Motor Shield V1(cheaper on ebay)
- 6 pack AA battery holder (with batteries)
- 8 pack AA holder (with batteries)
- Mini Breadboard
- Jumper Cables
- Bluetooth module (HC-05)
- Smart phone
- LED's (2 white, 2 red - All being 10mm, along with having resistors for them)
- Tools: Wire Stripper, Soldering Iron, Electrical Tape, etc.
- Power line breadboard
- 10K resistor, 4*220 resistor
- Ultrasonic Sensor
- Light Dependent Resistor
- Bunch of Wires
- Android phone that can load your app

The very first step for building the car is to gather all parts required are present. A good amount of testing is done on each sensor is working well before fully installed in the RC. This is to ensure that you have all your pieces in working conditions and will also save time down the road for bigger defaults.

With all that, we can now start building the car by following these straightforward steps:

1. Testing process with Arduino, motors(servo's), bluetooth chip, etc

We started off by practicing our skills and seeing how to motor control work, arduino, and how bluetooth chip is functioned. This part is slightly a step backward for us to reach a step forward. We looked at other people's code on various different task, like controlling an RGB led light using bluetooth control. Also testing how to drive motor's and the basic commands for it. This part was just to test out and a like a warm up to coding.

2. Researching process

With all the parts on the way and some having arrived. We went ahead and gather a whole bunch of research on the parts. For example, how they work, the pin layout, some libraries that are needed, the available pins that we can use on the motor shield. Also some considerations were taken into place for where to place the LED's, LDR, and the turbo sticking out of the bonnet(ultrasonic sensor). Along with that we had to do a bit of

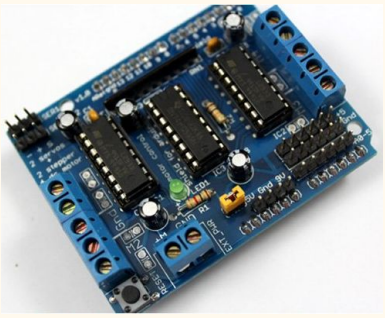
drilling for the placement of the these parts. This research process wen on for a while. Till the other parts arrived we used what we had. Luckily the motor shield, the car itself and the bluetooth were the first available resources we had. This worked out perfectly since we could get to work on the base foundation of the RC Car itself.

3. Gathering Materials

Originally we went ahead with the Adafruit Motor Shield V1. The V1 **SUCKED!** The was no headers for it, it took up so many pins of our uno, the shield was smoking, some motor drives got burnt out. Some materials got damaged in the process. Like the ultrasonic sensor had a cone inside the transmitter which just came off and the distance we were getting was very inaccurate. We ended up borrowing another ultrasonic from Mr. Cordiner. Without Mr. Cordiner we wouldn't have the smart feature braking. We also had to do rationing since we had 10mm led which weren't an available resource and had to get them from the store. Gathering the parts was a fairly simple task. All it required was to make sure we had the parts at the end of the day and they weren't misplaced. Overall gathering the material caused some anxiety because we didn't know if the parts would arrive on time. For example, the V2 Motor Shield arrived the day of when we did the demo. All in all the parts we had weren't the best but we managed to work with it and get it to work at the end.

4. Plugging in the Motor Shield to the Arduino

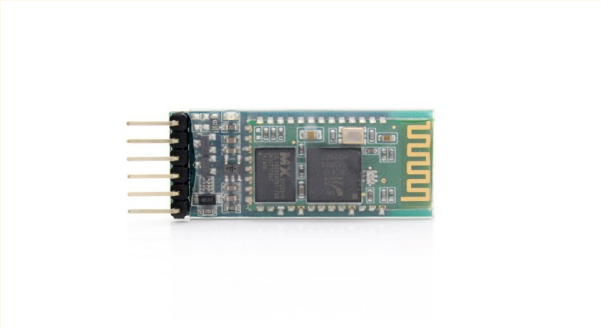
Plugging the Motor Shield into Arduino is a fairly simple task, but some things to watch out for are making sure the pins on the Shield fit right into the Arduino. Some gaps can cause the Shield to not function properly. For the power source, we need 2 power sources that are greater than 5 Volts. One power source is wired to the Motor Shield to power the motor and the other power source is powered to the Arduino. This will give the car a more steady voltage. Along with that we need to wire in the bluetooth chip. To do this we plug the bluetooth chip into 5V and GND. Last but not least we need to connect by wiring the Bluetooth Chip RX to the TX and Arduino TX to the RX .



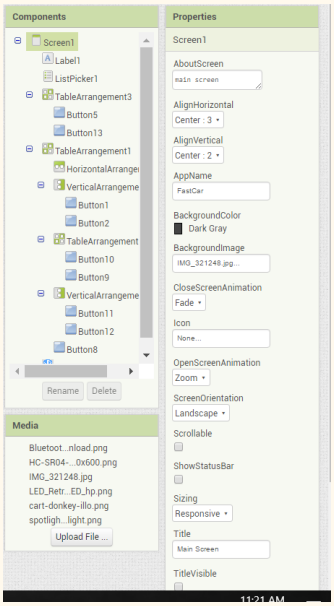
5. Testing the Motor Shield

To begin testing the Motor Shield, we have to start off by connecting the DC Motors to the Shield. The best part for the Adafruit motor shield is that it has 4 DC Motor ports.

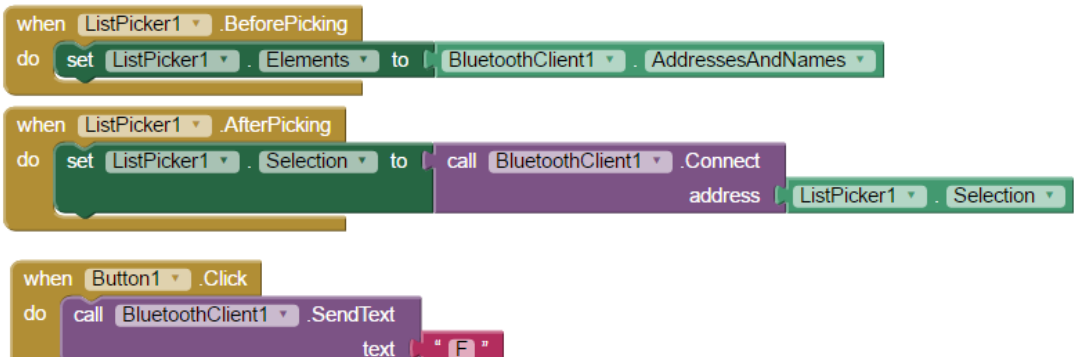
That means that if we jam or fry a port we have 2 more left over. To test the Motor Shield we can wire the DC Motor to the port 1, 2, 3, or 4. Now is the part to declare the frequency of the Motor Shield along with it's location, pin on the Shield, and the set speed for motor. With this information we can set the steering motor to run faster than our rear motor itself. This was all experimental and with that car just has commands that are placed in a loop and it follows it over and over again. The Motor Shield only works with 3 trigger commands to control the motor. These are release, forward, backward. Even though the names apply forward and backward, it doesn't necessarily have to follow that order. These command tell the car which way to send the current, which drives the motor in different directions.



6. Building the Android App



The part that we assumed would be the hardest to accomplish was the Android. Since we originally planned on using Android Studio to develop the app. Which was going to be a challenge that was going to be hard to accomplish in the time period we had. Instead we found an Android development package called MIT app Inventor 2. This program made everything as simple as possible. From drag and drop to auto list arrangement for list. Everything on the app builder site was simple and fairly easy for a first timer to learn and pick up on. The bluetooth app sends only 1 key char. The arduino receives the char and does whatever task was declared for the char received from the phone. The overall app components is CSS and the blocks are HTML . In the below figure is an example of the components and blocks that we used in our code:

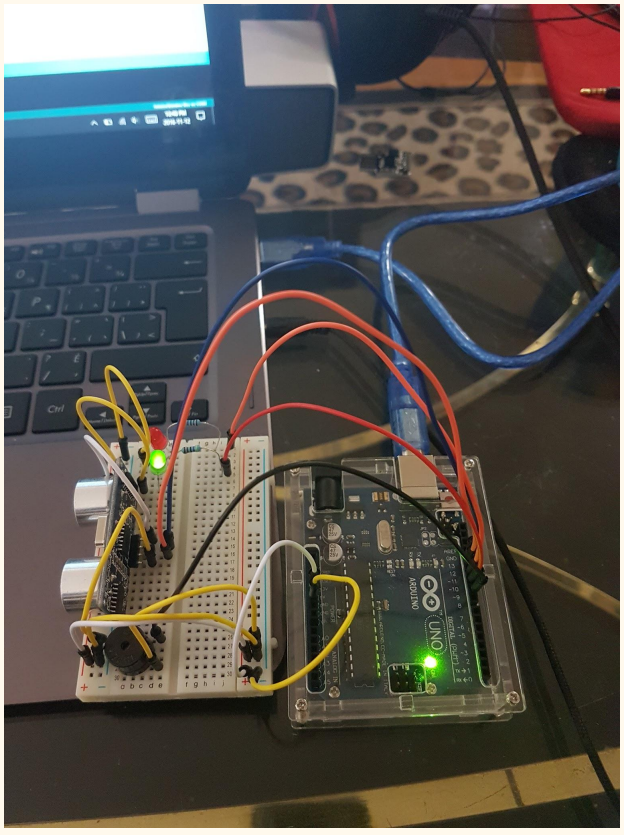


7. Incorporating the bluetooth with the Shield

The incorporation was fairly simple, we assigned the arduino to receive char letter, and we called it F, B,L,R,W,E. These are the char that the arduino receives from the phone VIA bluetooth. With the command the arduino receives, it acts according to it. For example, if the arduino receives the char E by texting the char gets picked up by the bluetooth module. The bluetooth module then receives it and puts the data to the TX pin of the module. This then travels to the Arduino's RX pin. The arduino then uses this value and compares it with the if statements that have the code for it. As a result following through with what the if statement instructs the arduino to do. We originally plugged the arduino directly from male to female wire, this cause a loss of connection every few times since the wires kept coming lose. To solve this issue we place the the bluetooth module on a mini breadboard. This left it to stay in place for the duration of the time.

8. Incorporating of LDR, LED and Ultrasonic Sensor

The incorporation by far was the most challenging part of all. One of the challenges we faced was having one part of the RC working where the other part would not function. After we accomplished our bluetooth controls to work we wanted to move on to extra features. This part is the above and beyond features that some of the modern cars have.. Features like automatic lighting and smart braking. These features are a lot of work to integrate to work together, specially when it is integrated with the motor shield. Being that the motor shield didn't have any headers but luckily our arduino, having hole beside the headers for the same pin connection made the task possible to do. We ended up soldering the wire through the holes for the pins we needed. We started of by making separate code for the lighting system and smart braking to learn the fundamentals of how it works. After

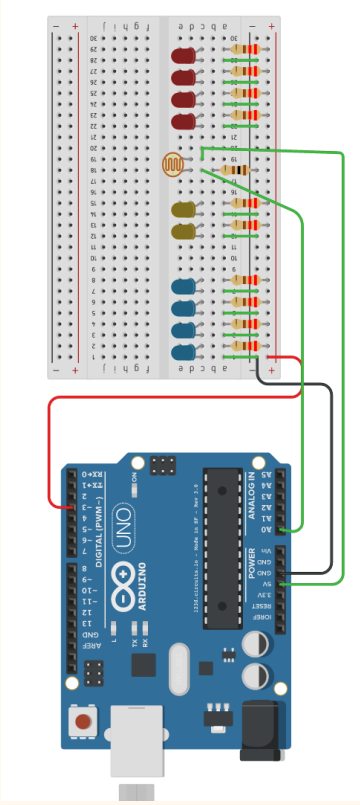


calculating the results and the averages for the LDR's resistance and finding good resistance to turn the LED's on for the auto braking to kick in. We began integrating them both. At this step. the code had to be change several times to ensure that there isn't interference between any of the features. The following code are the previously made codes to test the individual parts.

Initial

Auto Lighting	Smart Braking

int ldr = A2; //pin for ldr
int ldrValue = 0; //initial reading of ldr
int led = 3; //pin location to turn on the lights when it's dark for the ldr
 	
void setup() {
  pinMode(led, OUTPUT);
  Serial.begin(9600);
}
 	
void loop() {
  ldrValue = analogRead(ldr); //reads the ldr value and changes it from the original 0 we set it as
  Serial.println(ldrValue); //prints the ldr value on the serial monitor
  delay(500); //gets the reading after every half a second
  if (ldrValue <=100){
  analogWrite (led, 255); //if the value is this range or less turn on the led
  }
  else {
  analogWrite (led, 0); //otherwise keep the led off
  }
}
	/* This code is to check the if anything is in the way of the ultra sonic sensor, if it is the buzzer will beep on, off, and the led will remain on*/
	#define trigPin 9 //transmitter pin
	#define echoPin 11 //receiver pin
	#define redLed 10 //if the distance is less than 20cm
	#define led2 3 //if more than 10cm
	#define buzz 9 //pin for the buzzer to beep
	
	void setup() {
	  Serial.begin (9600);
	  pinMode(trigPin, OUTPUT);
	  pinMode(echoPin, INPUT);
	  pinMode(redLed, OUTPUT);
	  pinMode(led2, OUTPUT);
	}
	
	void loop() {
	  long duration, distance; //calling the distance for cm
	  digitalWrite(trigPin, LOW); // sends a short burst
	  delayMicroseconds(2); // followed by a small delay
	  digitalWrite(trigPin, HIGH); //sends a larger burst
	  delayMicroseconds(10); // with another longer delay
	  digitalWrite(trigPin, LOW); //turns of the transmitter
	  duration = pulseIn(echoPin, HIGH); //waits and calculates the time it takes to receive it
	  distance = (duration/2) / 29.1; //the math for breaking the time to distance
	  Serial.println(distance); //prints the distance
	  if (distance < 20) { // This is where the red LED is on happens along with the beeping buzzer
	  tone(buzz, 200);
	  digitalWrite(buzz, HIGH);
	  delay(25);
	  noTone(buzz);



```
    digitalWrite(buzz, LOW);
    delay(25);
   digitalWrite(redLed,HIGH); // When the Red
condition is met, the Green LED should turn off
  digitalWrite(led2,LOW);

}
  else { //red LED off and green LED is on
   // noTone(3);
    digitalWrite(redLed,LOW);
    digitalWrite(led2,HIGH);
  }
  if (distance >= 200 || distance <= 0){ //if the distance is
really far it avoids it
    Serial.println("Out of range");
  }
  else {
    Serial.print(distance); //prints the distance if the
object isn't out of range
    Serial.println(" cm");
  }
  delay(100);
}
```

REFLECTIVE QUESTIONS

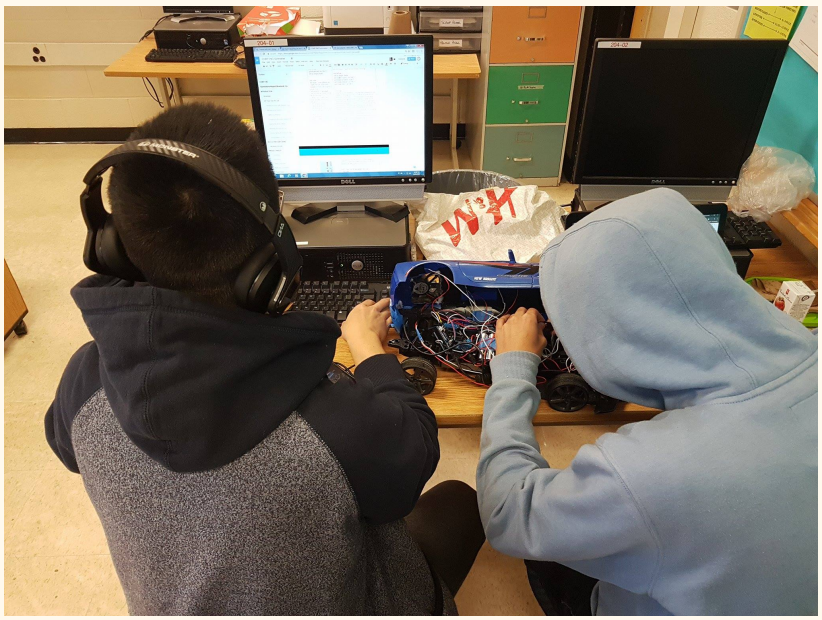
Problem One: The bluetooth just wasn't working properly!

After studying the chip for a while, and learning about how the data is sent and received from our app to the arduino, we wrote some code following a bunch of different tutorials. After we uploaded our code, nothing worked. The basic code we had originally written basically the string of text (one word) the app would send to the bluetooth chip; basic commands like "FORWARD", and "BACKWARD". The bluetooth module would receive that text in ASCII encoding, and we wrote a snippet of code that would allow for us to combine each individual character into the string, stored as a "command" variable. There would then be conditional statements that would check for what command was written and program the 2 DC motors of the car to run a certain way

The problem was that the car simply wasn't moving, followed commands inconsistently, and the bluetooth kept on disconnecting. We had initially believed the problem was due to faulty wiring as for we used a mixture of tape/hot glue to secure the wires together to the arduino (Our board did not have headers so we had to connect the wires directly to the holes of the PCB of the arduino).
We believed this because the bluetooth would have better connectivity when the wires were bent a certain way. To fix this, we simply re-wired the bluetooth chip with solder. This, however, didn't solve the problem.

We then tested if the bluetooth chip was actually receiving the data from our app and sending that data to the arduino by tracking it on the Serial Monitor of the arduino (printing what command was received). This helped us solve another problem which was that the app was sending some commands that did not match up with that the code (conditional statements for running the motor) expected. We changed the statements to make sure that they matched up.

We continued to test the bluetooth chip to see if it was receiving/transmitting the data properly. From the LED light on the chip which indicated its state, we could tell that it usually disconnected after the 3rd command from our app. This was a really strange problem because it worked perfectly fine the first few commands before



disconnecting. After failing to find the cause of this problem, we decided to see if Mr. Cordiner had any insight. He recommended that we check the voltage of the battery pack to see if the bluetooth chip was constantly receiving enough power. After testing the power source with a voltmeter with our code running, we realized that the bluetooth chip disconnected usually when the voltage coming to the motor shield (which was what the BT chip was connected to for power) when the voltage was less than 5V. To fix this, we added additional battery packs to the motor shield itself rather than the arduino to assure that all components were receiving sufficient power. This fixed the problem and our car had its basic function working properly.

The main lesson here is that fixing one problem could always lead to two more problems; one should not assume but make sure the changes they are to make will positively affect the situation. In our case, we did some irrational things which could have hindered the progress of our project.

Problem Two: Organization, and planning for the worst

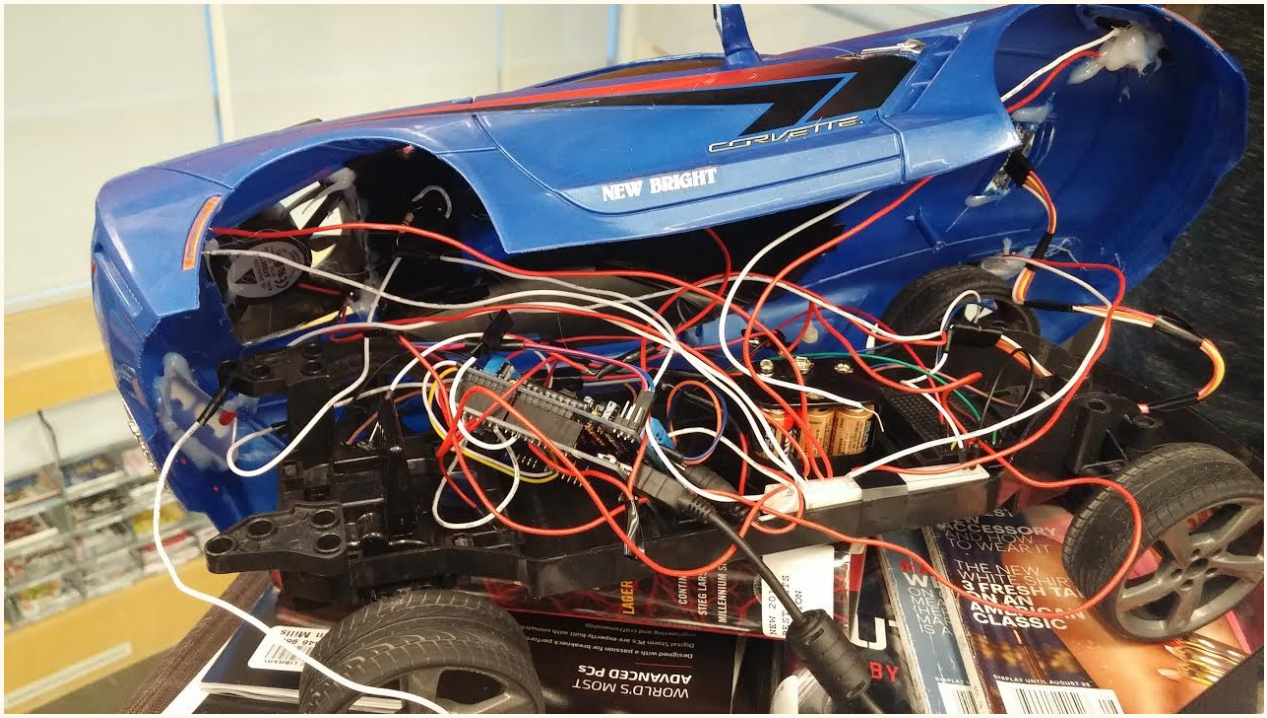
One of the other main problems we had in this project was trying to get everything to function properly at the same time. This is a general problem we encountered many times over the period of working on this project; and it truly hindered our progress.

Hardware/Circuitry

Many things we expected to work did not work as planned. For example, when testing the ultrasonic sensor - it worked fine. But once we integrated the code segment within our code, we found that more problems had occurred. Many times, it was from the physical wiring of the whole circuit; many things were coming loose and we were having a lot of trouble organizing everything because we just did many things without extensive, careful planning.

The wiring was originally very unorganized, making fixing loose pins and tracing the circuit very difficult.

The wiring:



We solved this problem by re-wiring EVERYTHING. It took quite some time, but we made sure that we did some planning in order to wire the circuit in an effective, and clean way. We taped and colour-labeled certain wires to make sure that the wiring would not cause that much of a headache this time.

Software

We (obviously) encountered many problems with the code itself. Relevant to this problem reflection, the code itself was somewhat messy originally because we hadn't planned it out very well; every time we integrated a new feature of the smart car, we simply added the segment into the code. The code became unorganized. This forced many errors onto our program; a lot of unknown errors in which we had to research how to fix.

To fix this problem, we read the code line by line and organized it in a way that it was structured and readable. We also added more comments to make more sense of the code in case we forgot our initial logic when writing it.

Takeaway

The takeaway to both the software and hardware disorganization problem is that messy code and wiring can prove to be a very big problem. Next time, we will definitely do more planning on how to code and wire more efficiently with better organization.

The Final Piece, The code:

```
// Using NewPing for ultrasonic sensor
#include <NewPing.h>
// Using this for motorshield V1
#include <AFMotor.h>

// Intializing front and back DC motor
AF_DCMotor frontMotor(1, MOTOR12_8KHZ);
AF_DCMotor backMotor(4, MOTOR12_8KHZ);

// Setting up original pins and max distance for NewPing library
#define TRIGGER_PIN  9
#define ECHO_PIN     11
#define MAX_DISTANCE 200

// Initializing new NewPing object; that is the ultrasonic sensor
NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

// Set up analog pin for LDR and digital PWM pin for all car lights
int ldrPin = 2;
int ldrValue = 0;
int carLeds = 3;

// Set up sonar values to use for averaging out the ping_cm() method response
int sonarValue = 0;
int totalSonarValue = 0;

void setup() {
  // Initialize serial communication
  Serial.begin(9600);

  //Initialize front and back motors' speeds
  frontMotor.setSpeed(255);
  backMotor.setSpeed(255);
  // Initialize trig and echo pin for ultasonic
  pinMode(9, OUTPUT);
  pinMode(11, INPUT);

  // Initialize car LED lights
  pinMode(carLeds, OUTPUT);
  digitalWrite(3, LOW);
```

```
}

void loop() {
  char command = getCommand(); //command user from mobile app gives

  //value from 0 to 255, higher value means more light
  ldrValue = analogRead(ldrPin)/4;

  if (ldrValue < 80){ // if it is a certain amount of darkness,
    // Turn on lights
    digitalWrite(carLeds, HIGH);
    } else{
      // turn off lights
      digitalWrite(carLeds, LOW);
      }

  // averaging sonar value in cm, 10 tries
  while(totalSonarValue == 0){
  for (int counter = 0; counter < 10; counter++){
    totalSonarValue = totalSonarValue + sonar.ping_cm();
    //Serial.println(totalSonarValue);
  }
  }

  // find average cm sonar value
  sonarValue = totalSonarValue/10;
  Serial.println(sonarValue);
  if (sonarValue < 25){ // if it is under
  // Stop
      frontMotor.run (RELEASE);
      backMotor.run (RELEASE);
      digitalWrite(carLeds, HIGH);
    } else {
       // Of course, we could have used for a case and switch or if/ else if statements, but that wouldn't have affected performance much/at all.
     // Therefore, although this option is long, it's already basic enough that simplifying more would do more harm than good.
    // Move forward
    if (command =='F'){
      backMotor.run (FORWARD);
      frontMotor.run (RELEASE);
      //delay(500);
```

```
}
// Move back
if (command =='B'){
  backMotor.run (BACKWARD);
  frontMotor.run (RELEASE);
  //delay(500);
}
// Move forward left
if (command =='L'){
  frontMotor.run (FORWARD);
  backMotor.run (FORWARD);
  //delay(500);
}
// Move forward right
if (command =='R'){
  frontMotor.run (BACKWARD);
  backMotor.run (FORWARD);
  //delay(500);
}
// Stop
if (command =='S'){
  frontMotor.run (RELEASE);
  backMotor.run (RELEASE);
  digitalWrite(carLeds, HIGH);
  //delay(500);
}
// Move back right
if (command =='E'){
  frontMotor.run (BACKWARD);
  backMotor.run (BACKWARD);
  //delay(500);
}
 // Move back left
if (command =='W'){
   frontMotor.run (FORWARD);
   backMotor.run (BACKWARD);
  //delay(500);
}

}
// Reset sensor value from ultrasonic
totalSonarValue = 0;
```

```
    }

char getCommand(){
  // Function of this method is to get a command from the serial port - RX
  // In this case it is connected to the bluetooth TX.

    // check if there is information to be read, from the bluetooth sensor's Tx and Rx that is connected Rx and Tx (receive/transmit) of the arduino
  if(Serial.available() > 0){
    // command becomes what the bluetooth module recieves from the app; stored as a character
    char command = Serial.read();
    // Print command for testing purposes
    Serial.print(command);
    Serial.print("\n");
    return(command);
  }
}
```

Segment 1: getChar function

This command was used to that code could be reused easily; the function of the method was to simply receive 1 character from the RX port of the arduino (in which the bluetooth was sending to). After receiving that command, it would return it as a character. Furthermore, it would print out the command received which was very helpful when testing.

```
char getCommand(){
  // Function of this method is to get a command from the serial port - RX
  // In this case it is connected to the bluetooth TX.

    // check if there is information to be read, from the bluetooth sensor's Tx and Rx that is connected Rx and Tx (receive/transmit) of the arduino
  if(Serial.available() > 0){
    // command becomes what the bluetooth module recieves from the app; stored as a character
    char command = Serial.read();
    // Print command for testing purposes
    Serial.print(command);
    Serial.print("\n");
    return(command);
  }
}
```

Segment 2: UltraSonic sensor and averaging

The code itself is pretty self explanatory. We are using a for loop to take 10 values in and then average it out. We use a method within the NewPing library, which automatically returns the cm value when called. Then, we average it out by dividing it by 10. If the sonar value is under a certain amount, we automatically stop the car. Otherwise, it will go on with the rest of the code.

```
void loop() {
// averaging sonar value in cm, 10 tries
  while(totalSonarValue == 0){
  for (int counter = 0; counter < 10; counter++){
    totalSonarValue = totalSonarValue + sonar.ping_cm();
    //Serial.println(totalSonarValue);
  }
  }
  // find average cm sonar value
  sonarValue = totalSonarValue/10;
  Serial.println(sonarValue);
  if (sonarValue < 25){ // if it is under
  // Stop
      frontMotor.run (RELEASE);
      backMotor.run (RELEASE);
      digitalWrite(carLeds, HIGH);
    } else {
       // Of course, we could have used for a case and switch or if/ else if statements, but that wouldn't have affected performance much/at all.
     // Therefore, although this option is long, it's already basic enough that simplifying more would do more harm than good.
    // Move forward
}
```

The App Dowload Link:

The android app is avaliable at the following link: http://tinyurl.com/zkjdyd3