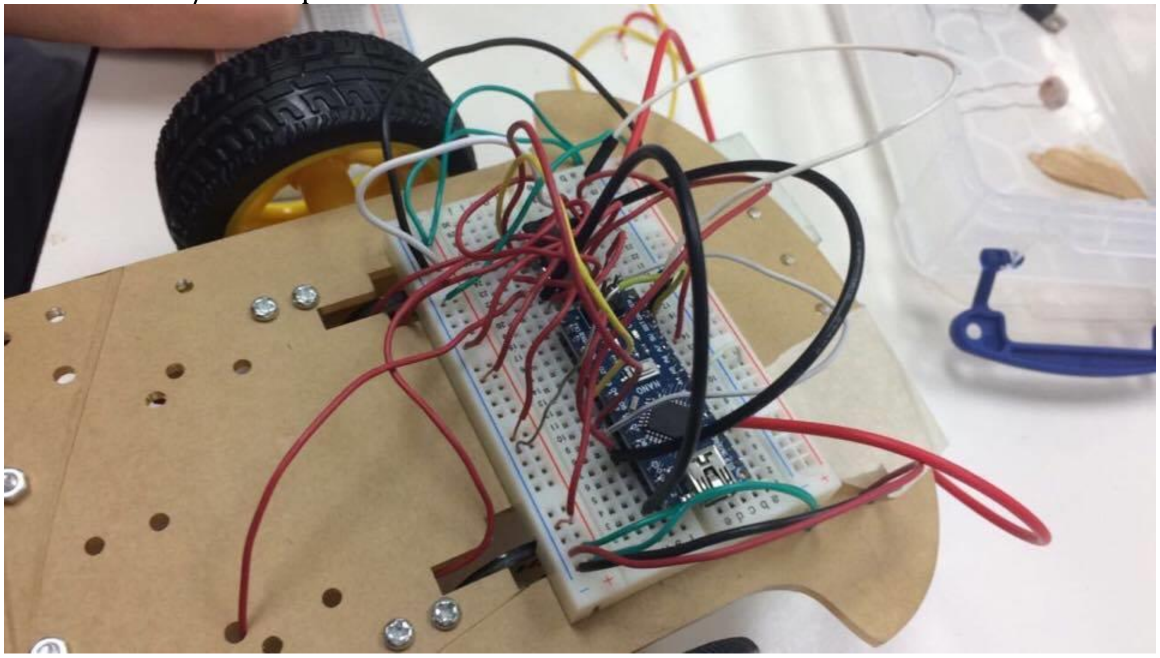
EE3 Final Report Justin Chang, Fong Chi Ho

1) Introduction and Background:

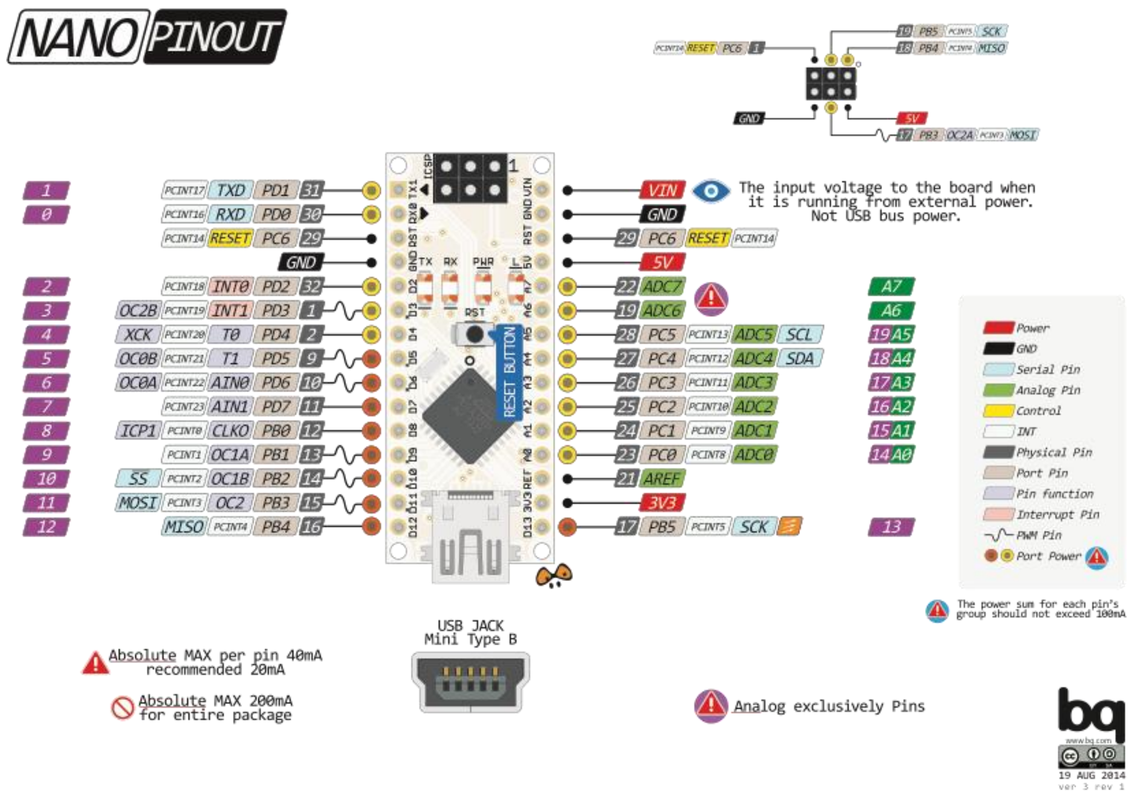
The goal for this project was to have a solid grasp and opportunity to apply basic concepts of Electrical Engineering and Computer Science learned in prior and current coursework. Essentially, the details of this project can be broken down to a simple, "How Well can Your Car Follow the Electric Tape?" question. We were given components such as (Infrared)IR LEDs, (Infrared) IR Sensors, regular LEDs, a Halls Effect Sensor(Magnetic Detecting Sensor), 2 DC Motors, a few breadboards to place the components (assuming resistors and wires were also given), a 9V Battery, and controller known as the Arduino Nano 3.0 (a small microcontroller), and chip known as the H bridge (L293D chip, used to send and receive signals to power Motors and Arduino). The microcontroller (Arduino Nano 3.0) could be programmed by uploading code (C language) from an Arduino IDE **[2],** that controlled the car. Since our objective in this project was to find and follow an electric tape track(one track was a straight line, while the other was circular), we had to learn and interpret data collected from our components usefully and use the microcontroller to accurately and consistently control the cart, when all the hardware is properly setup. The design selected to handle this project was inspired from the OPS IEEE 2016-2017 capstone schematic **[3]** and several consideration/class tips.



(Arduino, PID Description, "Basic Theory")

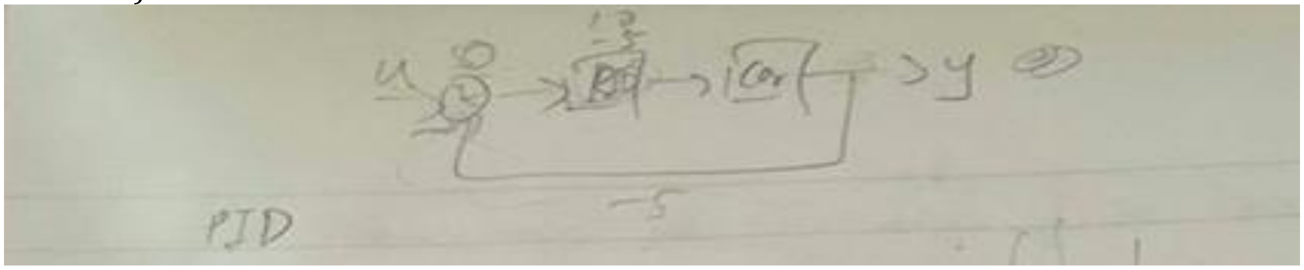
(Arduino) Before finding out what values to write to the motors, or choosing how to handle turns, there are 2 main concepts about the Arduino that needs to be understood. First, Arduinos have an analog read/write or digital read/write function, which means in the setup function of the Arduino, there can either be a component that is used to input or output, and only reads or writes to HIGH (on) or LOW(off) if in digital read/write, or if in analog (writes from 0-255, reads from 0-

1023) which would require connecting components in some form, usually by wire or resistors to a digital pin or analog pin based on the Nano pinout below **[1]**. However, whereas digital pins always work with digital read/write functions, connecting to the analog pins would mean reading an analog value (and not necessarily write, which leads to the second main concept about the Arduino). An interesting functionality of the Arduino known as PWM (Pulse Width Modulation) helps write to the analog values for certain components


NANO PINOUT
The input voltage to the board when it is running from external power. Not USB bus power.
RESET BUTTON
USB JACK Mini Type B
Absolute MAX per pin 40mA recommended 20mA
Absolute MAX 200mA for entire package
Analog exclusively Pins
The power sum for each pin's group should not exceed 100mA
Power
GND
Serial Pin
Analog Pin
Control
INT
Physical Pin
Port Pin
Pin function
Interrupt Pin
PWM Pin
Port Power
bq
19 AUG 2014
ver 3 rev 1

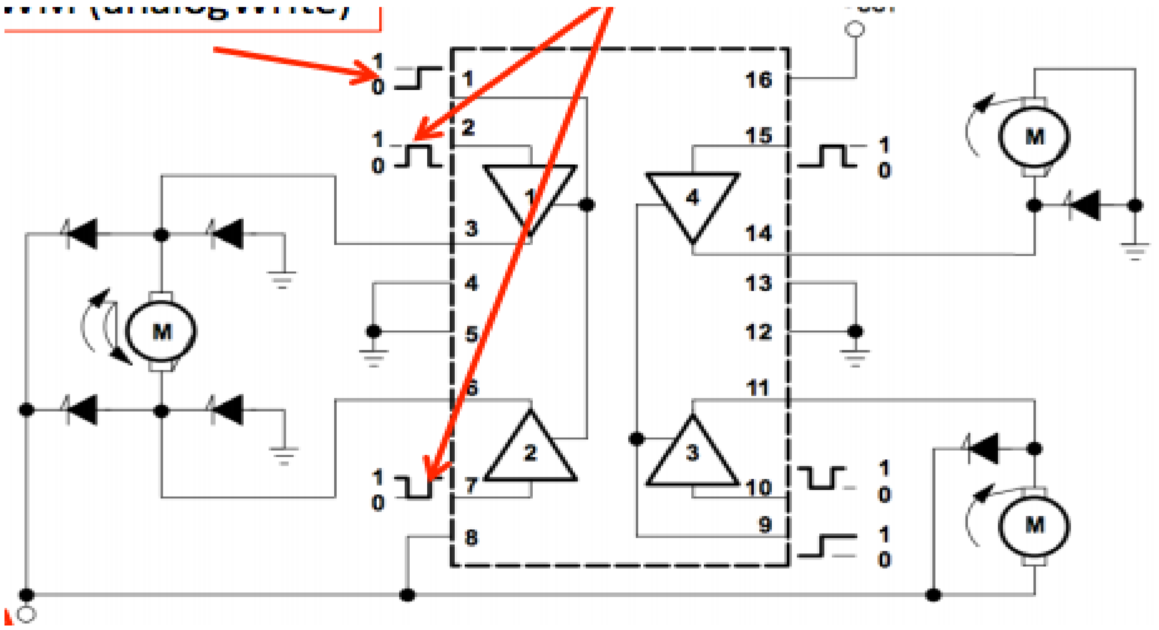
As shown in the diagram above**[1]**, certain arduino pins on the digital side such as D3,D5,D10 etc can write analog values (0-255) to control power output. For example, it could be used to control how bright an LED turns, or how strong a sensor displays a value, or even how fast we can control the motor using the H Bridge. As long as these values were well defined in the Arduino, then motors can be controlled through analog write and PWM, and sensors can give the car an idea where it is. After setting up the connections to the sensors, motors, battery, and H-Bridge, code can be uploaded from the Arduino IDE and uploaded to the Arduino which will serve as a controller for the car when supplied power from the 9V battery.

(PID) Another key concept of an approach for determining how much a turn should be in a given direction comes with understanding the concept of PID, a controller that uses negative feedback to adjust motor speed to guide the cart back on the track. The main idea, simply put, is that if the car is, for a certain instance of time, having ideal readings, then suddenly has non-ideal readings, a formula must be applied to figure out where the car might be in the current situation, and must attempt to readjust accordingly. The primary concept of proportion (how far from the track), derivative (change in distance from the track) were considered in the writing of PID code. Understanding that proportion can tell if we are either on the left side or right side of the track is helpful, but useless if there is no derivative controller, which keeps track of the change in error and readjusts accordingly. The corresponding equation defines 2 constants :Kp,Kd, and checks the error from the "ideal" ir sensor readings each iteration or time delay before a future check by taking current readings and subtracting them from previous readings and summing up the total proportion, derivative, and integral errors in a nice formula of the form(pseudo code from OPS Slides **[3]**, and picture of block diagram of PID is from notebook):



```
(gives the error from the right sensor)
errorRight=ideal1-LED1V;
derivativeRight=(pErrorRight-errorRight)/dt;
pErrorRight=errorRight;
double CorrectionRight=KP*errorRight+KD*derivativeRight;
(gives the error from the left sensor)
errorLeft=ideal2-LED2V;
derivativeLeft=(pErrorLeft-errorLeft)/dt;
pErrorLeft=errorLeft;
double CorrectionLeft=KP*errorLeft+KD*derivativeLeft;
```

Additionally, depending on the motors, although the setups may look the same, internal resistance and other factors get in the way of successfully calibrating the proper ratio between the two motors. Ex: In our case, the left motor was weaker than the right motor. Understanding the H Bridge and how the discrete outputs (digital Write to motors) and internal output diodes for battery connections as well as other built in transistors help a lot in explaining why certain motor, Arduino, and voltage connections were necessary in the layout of the design. **(Below is the H Bridge schematic, referenced from Wk 6 Project Suggestions)**

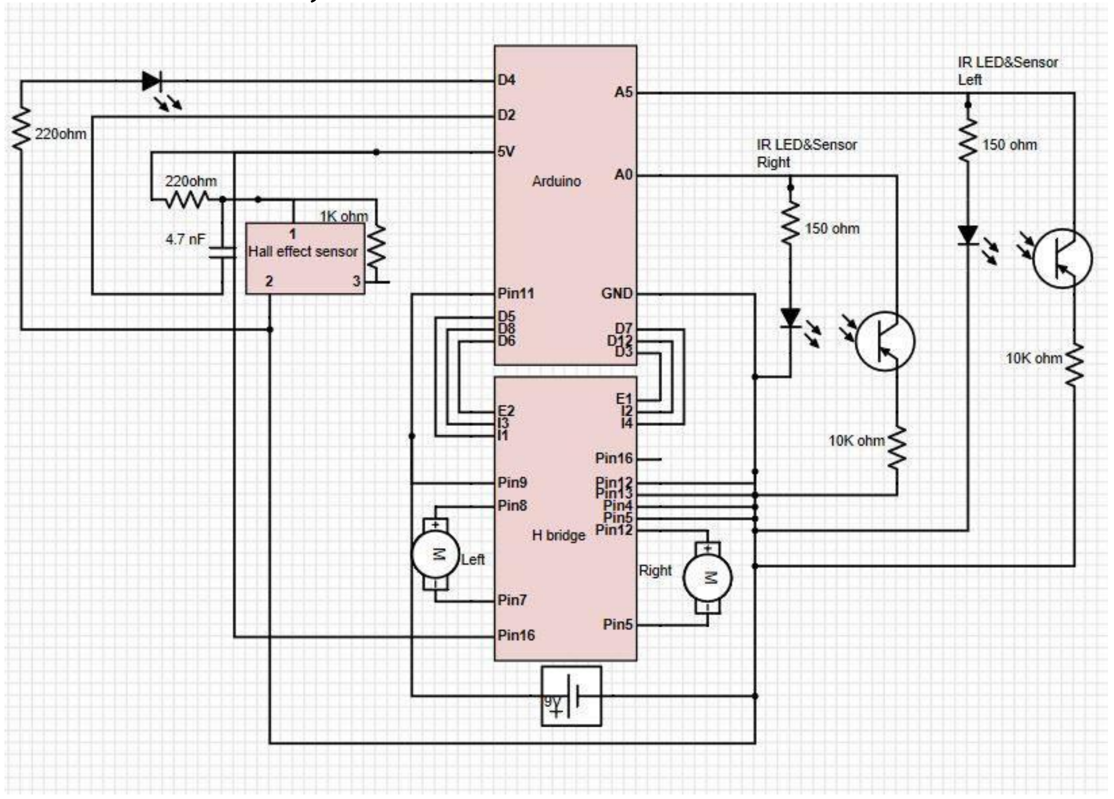


2) Testing Methodology:

Very briefly, the following components that will be listed were tested:

1. IR Sensors and IR Leds (callibrations)
2. 9V, 6V, 5V from Arduino and H Bridge (L239D) (motor control values and PWM)
3. Halls Effect Sensor, Regular LED (220 Ohms)(magnetic sensor range of response)
4. Bringing it all together (how sensors, both IR and halls effect work with Arduino and motors, **more will be discussed** in the Results/Discussion Section)

Circuit for whole Project:



1. IR Sensor and Led Callibration:
a) "How We Designed the Test": First, test setup was necessary for sensor callibrations, since different areas might give the IR Sensors different "ideal" values. To figure out where to calibrate, we tested the sensors on the actual line track, a simplified line track, and the regular lab tables, and ran the car over tape over some darker colored object to see how the IR Sensors reacted to each test.
b) "How We Conducted the Test": If both right and left (white) IR Recievers would change values, depending on what is detected, and if the Arduino was connected to the computer (assume code is uploaded), then the Serial Monitor would be able to display changes every execution of the loop. This meant before doing anything though, the IR Leds and Receivers had to be connected to the Arduino through analog pins. The Receivers would be defined as input that reads from analogRead, and average 50 readings in order to determine its ideal value. Next, we had to place the car on a simplified line track (8x11 plain white sheet of paper with electric tape down the middle of the paper) and observed the IR sensor readings when both sensors were on the paper, and when the right sensor was on the paper, and when the left sensor was on the paper. Next, we took the car to the actual track with computer connected to the Arduino, and observed any potential changes to the IR readings, and wrote down our observations.

**The code used for verifying this is below…

```
//setup
int nSamples=50;
for(int n=0;n<nSamples;n++)
f+=analogRead(IRLED1);
f/=nSamples;
//loop
int IR1=analogRead(IRLED1);  //right sensor.
int LED1V=IR1;
int IR2=analogRead(IRLED2);
int LED2V=IR2;
Serial.print("IR1 is ");
Serial.println(LED1V);
Serial.print("IR2 is ");
Serial.println(LED2V);
```

c) "How We Analyzed the Test Data": On the simplified line track, the IR Sensors when both sensors were not on the tape, (our code doesn't do any fancy mapping on the analogRead values, Ex: from 0-1023->255-0 so these are raw readings) the Right Sensor showed a range of values from 975-982, and the Left Sensor showed a range of values from 958-963) On the actual track, the values were a little bit more consistent towards 980 for the Right Sensor and 960 on the Left Sensor, so we decided to define those values as our "Ideal" values on the track. When it detected tape (fully on top of the tape), the values could drop as low as 200 for both sensors, and anywhere around 400-high 600s low 700s would mean in some form, somewhat over the track. Using analogRead is more helpful than digital because this gives a more precise value on how we should address the turns in the code, and gives more detail/information to the PID code to use in motor control. In short, if these values were mapped to motor control, it would look something like this in proportion controller's point of view, if the **LEFT SENSOR was like this**:

IR Sensor Values:	950+	400-700	200
Right Motor	Right Speed	-CorrectionLeft	-CorrectionLeft(More)
Left Motor	Left Speed	+CorrectionLeft	+CorrectionLeft(More)

And vice versa for the other sensor. However, there should be other factors to consider when considering these turns, above is a rough idea as to what certain IR Sensors could mean for the car.

d) "How We Interpreted the Data": Assuming that all the correct connections are correct and that the sensors aren't too close together on the bottom (are well separated and stand up straight when placed on the track, because they CAN in fact move a lot and mess up readings left and right), then the sensors are actually relatively consistent and can be really helpful in working with the motors for the required project at hand. The other work needed to figure out motor control and PWM values as well as how to handle turns/PID will be discussed shortly after this section. In short, the values we were getting were pretty helpful and provided a rough mapping for how we could be handling the turns in the car. We would go as straight as possible until an IR Sensor reaches a value less than 700 (since values >700 are assumed to be relatively ideal) and as a result, execute PID or some form of correction adjustment based on the sensor that is on the electric tape.

2. H Bridge and Motor PWM Configurations:

a) "How We Designed the Test": We recognized that the motor at first, had the possibility of only going "HIGH" or max speed, if the analogWrite pin isn't correctly defined, as well as that there might be other hardware problems with setting up the connections for the H Bridge to Arduino (refer to above schematic). So as a result, after doing some research on PWM and referencing class notes, we decided to test different sets of code to configure for the motor, a "straight line", "turn right" , "turn left", and speed adjust code after delays. If these conditions were met for the code written, then the hardware and test should be verified and ready to be combined with the values from step 1 above, and the PID code to provide the correct adjustments to the car.

b) "How We Conducted the Test": To track the car going in either a straight line, making small turns to the right/left (by changing the direction of how the wheel turned, which probably wasn't the best idea and I will get to that later in this report), we tried to set the analogRead enable pin motor values (the 1st, upper left, and bottom right of the H Bridge) to some arbitrary small value that would get the car moving when plugged into the 9V battery (assuming all the hardware is setup correctly). Afterwards, we also decided after a delay of 1000 of going straight, we would try to increase speed by a set value for the car to make sure we could properly adjust the speed with code. If the motor was too fast, we knew we would have to lower the values, and if nothing changes, that would mean we weren't actually using PWM correctly and that the digital pin the motors are connected to were probably hooked up to the pins that only write to digitalWrite output.

**The code used for verifying this is below…

```
//setup
   pinMode(I1, OUTPUT);
   pinMode(I2, OUTPUT);
   pinMode(I3, OUTPUT);
   pinMode(I4, OUTPUT);
//loop
analogWrite(E1, 78); //e1 corresponds to right motor. Right motor faster than Left
Motor if both were at the same value, so value to even out.
   analogWrite(E2, 99);
   digitalWrite(I1, LOW); //I 1 moves it back, woops flipped
   digitalWrite(I2, HIGH); //I2 moves it forward
   digitalWrite(I3, HIGH); //I 3 moves it forward
   digitalWrite(I4, LOW); //I4 moves it back
delay(1000);
   analogWrite(E1,78+40); //increase right motor speed, turn left
delay(1000);  //etc
```

c) "How We Analyzed the Test Data": Based on our first attempt, the car was at full speed and didn't change in speed based on each delay, so we knew we didn't apply PWM correctly (actually had the code define the motor to D2, which is NOT A PWM pin that allows analogWrite and only allowed digitalWrite for the enable pins). So, we had to fix that and saw later on that values for the motors such as 100 and up was too fast and values around 70-90 were more ideal and reliable for the car (without taking into consideration voltages supplied from 9V yet, which plays a huge role in this project). Later on, we also noticed that during turns (which seemed to execute decently well in practice), drew a lot of power because our turns relied on using the Arduino to digitalWrite the motor to go BACKWARDS on the wheel that needed to slow down, making the turns much stronger and in some ways, easier to slow down (though later on proves to be detrimental because of the power drawn and inconsistency). Eventually, we also noticed that "straight", "turn left/right" were only consistent given a certain range of voltages, preferably from 8.2V-8.8V and by the end of the project (refer to spreadsheet below in section 3).

d) "How We interpreted the Data": Although the data wasn't as consistent as we liked, we were able to have at least some control over how motor movement worked in general, and wanted to try bringing it together with the IR sensor code to see if it could properly detect the line and respond correctly in theory at the least. This will be explained in more detail, and described with more models/video as well later on in this report, but in general, we confirmed PWM and motor control.

3. Halls Effect Sensor and Regular LED:

a) "How We Designed the Test": This required more digging on our own parts because lectures didn't discuss this portion as extensively (which was expected since this was more of an extra credit effort). Nevertheless, there was a brief description of how this were to be handled, and we decided to first verify the schematic provided online and above on a breadboard. Next, test if it would work attached to the car, verifying through serial monitor then using a LED to signal the result of detecting a magnet.

b) “How We Conducted the Test”: Before conducting the actual test, we had to find some explanation on how the sensor worked, by referencing a tutorial **[4].** Then, we had to connect the LED to a resistor, and then to a digital pin for writing as a digitalWrite output, and the halls effect sensor which read from digitalRead as input, and is also connected to a capacitor and ground/digital pin on the Arduino. This schematic was first verified on breadboard and tested with a magnet, and on the track later when we taped it onto the car, in between the motors. We also needed to figure out the relative distance the sensor needed to be in order to register the magnet. The LED ended up floating like an “Antennae” due to suspending it with long wires, and the sensor placed under the car was held in place by the wires. Next, we had a magnet piece go through the wires, and checked the serial monitor for indications of “magnet being sensed”, and afterwards tried it on the track (which had magnets underneath the black tape) and looked for the same indications, as well as noting down if we missed any magnets or if any magnets were out of sensing range.

**The code used for verifying this is below…

```
//setup
        pinMode(HallEffectSensor, INPUT);
        pinMode(LED1, OUTPUT);
//loop
        digitalWrite(LED,LOW);
        int HES=digitalRead(HallEffectSensor); //if detect sensor, reads 1.
        Serial.println("HALLS EFFECT SENSOR MAGNET VALUE");
        Serial.println(HES); //1 for sensing value,0 for no magnet.
        if(HES==1) //sensed a magnet!
        {
                Serial.println("MAGNET SENSED!");
                digitalWrite(LED,HIGH);
        }
```

c) “How We Analyzed the Test Data”: The sensor had around (y-axis) 2.5 cm range (not sure if this was how strong the sensor should have been), but it was consistently working at that range. Anything too far wouldn’t accurately register with the sensor, and sometimes, if the car is off track, then it wouldn’t work.

d) “How We interpreted the Data”: The magnet sensor and LED didn’t really give us as much trouble calibrating or matching v.s. the motors and sensor. However, the sensor is heavily reliant on the car staying on track, and would fail to do its part if it doesn’t successfully follow it.

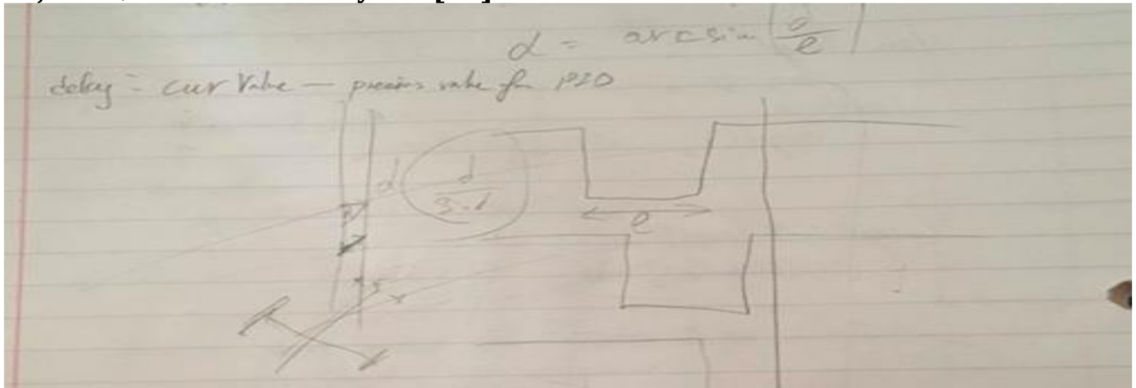
4. “Bringing it all Together”:

a) “How We Designed the Test”: In order to properly wrap up the other 3 main tests we had to do, we had to know where to implement and structure our commands/controls for the car. Figuring out for example, when to be going straight, or doing PID, and how to control the motor speed was something that had to be

done through trial and error on the track. We already knew where to turn the motor when a sensor starts straying away from ideal values, as well as how to handle turns, so the only real challenge was to see how it worked on the real track.
b) "How We Conducted the Test": The first step was to upload some code that combined IR readings and Motor movement, and observe how the car reacted when falling off track. The main concept to analyze here was if the car was steering to the left, and falling off track towards the left, the car must turn back on the track by turning right or have a positive steering (if define error to left as negative steering) and vice versa. After we knew for sure that worked, we could start tweaking motor values to control how much turning was required to ensure that it could eventually be as ideal and on the track as possible. In essence, the motor speeds, if correctly adjusted, should always reflect ideal values in which the sensors have the tape in between the car (longest, and core part of the project). After calibrating those values correctly, we would then move on to upload code that allows the car to signal that a magnet has been detected with the halls effect sensor and LED to signal that. Whenever something didn't work, we would adjust values and constants.
c) "How We Analyzed the Test Data": d) "How We interpreted the Data": For these sections, it would be easier to discuss in part 3 of the report where there will be a table describing, with notes, the motor control, Voltage supply, and record on tracks in terms of distance travelled, as well as method of approach (changed here and there over time).

Methods of consideration were: (Quick notes)

1. Calculating the distance the head of one side of the car is away from the tape, and the relative distance the side of the sensor that is off that tape, and finding the angle in between and adjusting by making a turn as a function of the angle needed to be adjusted, recommended by TA. **[1a]**


$\alpha = \arcsin\left(\frac{d}{e}\right)$
delay = cur Value — previous value for PID
α
$\frac{d}{3-1}$
e

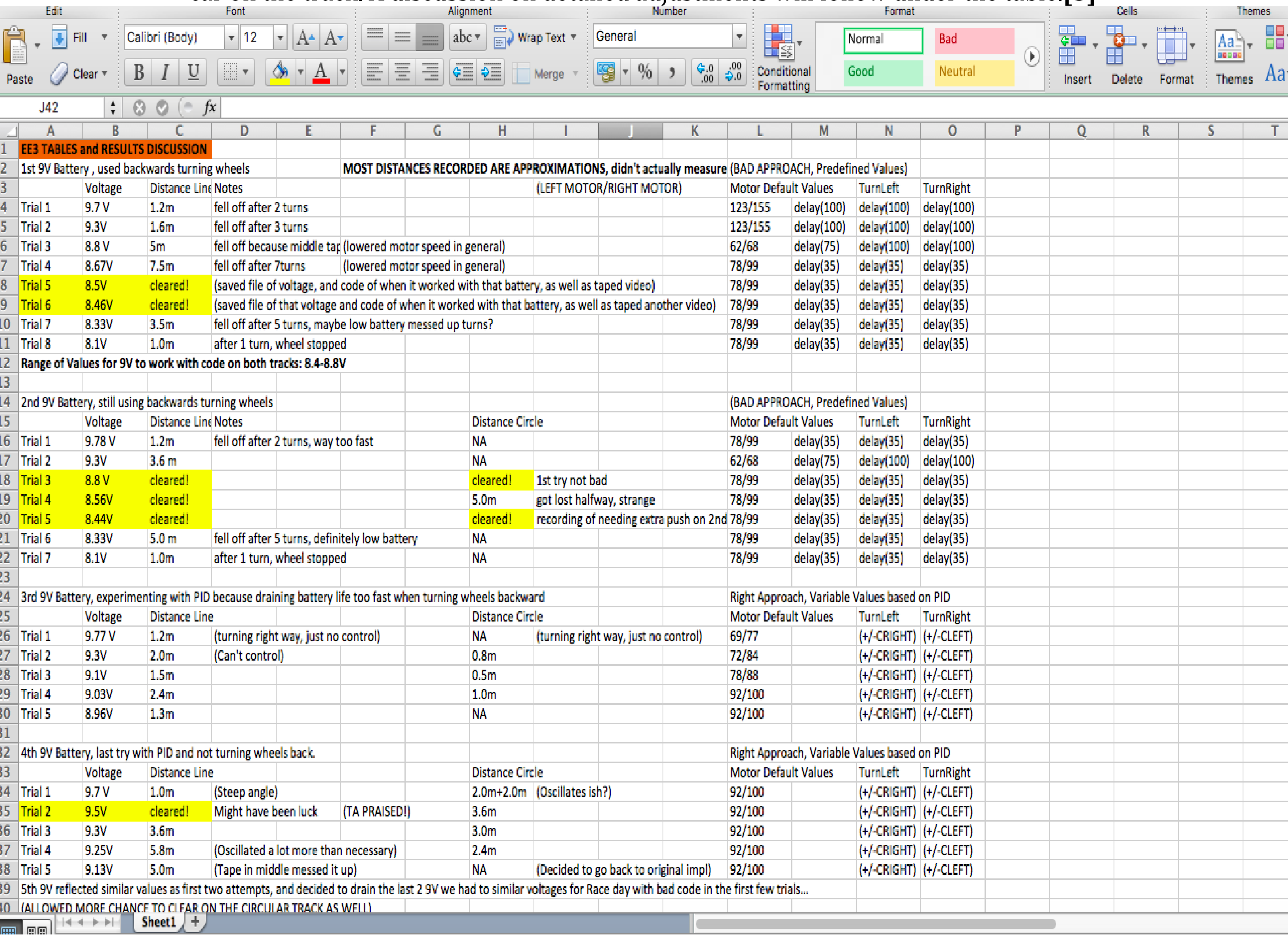
2. Using PID, calculating the value of how proportion and derivative of total error every few milliseconds we do a calculation, relative to the ideal position of the car, and mapping a value to add or subtract to the motors. Refer to Introduction Section for picture and theory behind PID (couldn't get this working, can check results discussion section for details and spreadsheet in that section).

Some other Considerations we had for the project in general were:

Using a 6V different battery supply (taken out because too many groups were short circuiting), shrouding sensors, a 3rd IR Led Sensor and Receiver, proper PID, and motor control (keeping wheels moving one direction) were considered in this project, though we decided to go with a design that first worked for us in a few early test runs (which ended up failing to clear both tracks come race day), and abandoned the other considerations.

3) Results and Discussion:

Below is a screenshot of the Data Table collected from all relevant test trials of the car on the track. A discussion on detailed adjustments will follow under the table.[5]



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	EE3 TABLES and RESULTS DISCUSSION														
2	1st 9V Battery , used backwards turning wheels					**MOST DISTANCES RECORDED ARE APPROXIMATIONS, didn't actually measure**						(BAD APPROACH, Predefined Values)			
3		Voltage	Distance Lin	Notes					(LEFT MOTOR/RIGHT MOTOR)			Motor Default Values		TurnLeft	TurnRight
4	Trial 1	9.7 V	1.2m	fell off after 2 turns								123/155	delay(100)	delay(100)	delay(100)
5	Trial 2	9.3V	1.6m	fell off after 3 turns								123/155	delay(100)	delay(100)	delay(100)
6	Trial 3	8.8 V	5m	fell off because middle tap	(lowered motor speed in general)							62/68	delay(75)	delay(100)	delay(100)
7	Trial 4	8.67V	7.5m	fell off after 7turns		(lowered motor speed in general)						78/99	delay(35)	delay(35)	delay(35)
8	Trial 5	8.5V	cleared!	(saved file of voltage, and code of when it worked with that battery, as well as taped video)								78/99	delay(35)	delay(35)	delay(35)
9	Trial 6	8.46V	cleared!	(saved file of that voltage and code of when it worked with that battery, as well as taped another video)								78/99	delay(35)	delay(35)	delay(35)
10	Trial 7	8.33V	3.5m	fell off after 5 turns, maybe low battery messed up turns?								78/99	delay(35)	delay(35)	delay(35)
11	Trial 8	8.1V	1.0m	after 1 turn, wheel stopped								78/99	delay(35)	delay(35)	delay(35)
12	**Range of Values for 9V to work with code on both tracks: 8.4-8.8V**														
13															
14	2nd 9V Battery, still using backwards turning wheels											(BAD APPROACH, Predefined Values)			
15		Voltage	Distance Lin	Notes				Distance Circle				Motor Default Values		TurnLeft	TurnRight
16	Trial 1	9.78 V	1.2m	fell off after 2 turns, way too fast				NA				78/99	delay(35)	delay(35)	delay(35)
17	Trial 2	9.3V	3.6 m					NA				62/68	delay(75)	delay(100)	delay(100)
18	Trial 3	8.8 V	cleared!					cleared!	1st try not bad			78/99	delay(35)	delay(35)	delay(35)
19	Trial 4	8.56V	cleared!					5.0m	got lost halfway, strange			78/99	delay(35)	delay(35)	delay(35)
20	Trial 5	8.44V	cleared!					cleared!	recording of needing extra push on 2nd			78/99	delay(35)	delay(35)	delay(35)
21	Trial 6	8.33V	5.0 m	fell off after 5 turns, definitely low battery				NA				78/99	delay(35)	delay(35)	delay(35)
22	Trial 7	8.1V	1.0m	after 1 turn, wheel stopped				NA				78/99	delay(35)	delay(35)	delay(35)
23															
24	3rd 9V Battery, experimenting with PID because draining battery life too fast when turning wheels backward											Right Approach, Variable Values based on PID			
25		Voltage	Distance Line					Distance Circle				Motor Default Values		TurnLeft	TurnRight
26	Trial 1	9.77 V	1.2m	(turning right way, just no control)				NA	(turning right way, just no control)			69/77		(+/-CRIGHT)	(+/-CLEFT)
27	Trial 2	9.3V	2.0m	(Can't control)				0.8m				72/84		(+/-CRIGHT)	(+/-CLEFT)
28	Trial 3	9.1V	1.5m					0.5m				78/88		(+/-CRIGHT)	(+/-CLEFT)
29	Trial 4	9.03V	2.4m					1.0m				92/100		(+/-CRIGHT)	(+/-CLEFT)
30	Trial 5	8.96V	1.3m					NA				92/100		(+/-CRIGHT)	(+/-CLEFT)
31															
32	4th 9V Battery, last try with PID and not turning wheels back.											Right Approach, Variable Values based on PID			
33		Voltage	Distance Line					Distance Circle				Motor Default Values		TurnLeft	TurnRight
34	Trial 1	9.7 V	1.0m	(Steep angle)				2.0m+2.0m	(Oscillates ish?)			92/100		(+/-CRIGHT)	(+/-CLEFT)
35	Trial 2	9.5V	cleared!	Might have been luck		(TA PRAISED!)		3.6m				92/100		(+/-CRIGHT)	(+/-CLEFT)
36	Trial 3	9.3V	3.6m					3.0m				92/100		(+/-CRIGHT)	(+/-CLEFT)
37	Trial 4	9.25V	5.8m	(Oscillated a lot more than necessary)				2.4m				92/100		(+/-CRIGHT)	(+/-CLEFT)
38	Trial 5	9.13V	5.0m	(Tape in middle messed it up)				NA	(Decided to go back to original impl)			92/100		(+/-CRIGHT)	(+/-CLEFT)
39	5th 9V reflected similar values as first two attempts, and decided to drain the last 2 9V we had to similar voltages for Race day with bad code in the first few trials...														
40	(ALLOWED MORE CHANCE TO CLEAR ON THE CIRCULAR TRACK AS WELL)														

At first, the "BAD APPROACH" used was only using IR readings at a set range to define a turn either left or right by a set amount (marked by delay), and was a brute force method to work the car into staying on the line, hard coding the values to match, rather than using PID or relying on correction values. However, this ended up working on certain battery voltages (with a small push here and there when the car just stops), and pushing the wheels backwards to make the turns a little bit slower, though also draining the battery a lot quicker.

At our third 9V battery, since we thought we had a working version of the code, we attempted proper PID correction values, and stopped turning the wheels back when a correction needed to be made. In other words, this also conserved battery a lot better and wouldn't suddenly stop sometimes on the track and need pushing to properly operate. We started off by slowing down the motors, and tried to find a proper mapping for the Correction Left and Correction Right values (from using the above expression on theory of PID to compute these correction values) to help the motors by only changing the amount of analogRead value in the code based on the correction values. If the turnRight, correction left value was a certain value, then the left motor speed would increase by the value of correction left (ranged from 0-2600)/200 and the right motor speed would decrease by the value of correction left (also ranged from 0-2600)/200. This is because in a right turn, the left wheel must be faster than the right wheel. Similarly, if it would need to turnLeft, a correction value of similar range would be computed to help the right motor increase in speed (range from 0-2600)/200 and decrease the left motor speed. This is because in a left turn, the right wheel needs to be faster than the left wheel. However, we did most of this without properly considering some of the offsets due to stronger motors, and that as a result made most of our test trials bad, and difficult to resolve.

Eventually, we decided to retreat to our first few attempts where we relied on using a back turning wheel to slow down the car in a turn (which drains a lot of battery) and only capable of working for battery voltages from 8.4-8.8V). If the Voltage of the battery was > 8.8V , then It would fail due to speed being too fast, and if battery was <8.4V, it would stop in place too many times and be unable to finish the track). We also learned that there was internal battery resistance because when plugging the function generator's 8.4-8.8V was way too fast for the car, so that the "actual" range of Voltage we were drawing should have been somewhere closer to 7.2-7.5V instead to get it to clear the track.

Most of the motor values obtained were from trial and error attempts to find the "right" values for the car. For example, 78/99 was very good for our car in 8.4V-8.8V batteries, but not as good for values greater for the same reasons as listed above. At first, we tried 123/155 because we thought that speed was also part of the goal of the project, but realized it had no control, and tried to tone it down as much as possible, while having a car that was still able to clear the tracks. Delay 100 was too long and didn't give the car enough time to find the track while delay 35 was just enough so that it could be constantly taking new input values (though totally wrong approach to this project). Later on when PID was considered, these values were taken out and turned into mapping values as discussed above.

Below in the references, are links to the videos of when…
[6] Car moving in a straight line first time correctly
[7] Car moving again in a straight line

[8] Car kind of clearing circle
[9] Car needing a little bit more pushing to clear the circle.

4) Conclusion and Future Work:

All in all, our design didn't meet our goal as it was unable to clear the tracks on presentation day, due to the fact we were too reliant on bad turning code and unstable PID correction values. We learned a lot about relying on bad code and bad methods for turning (the problem with turning with pushing a wheel back is that it draws too much power and gets stuck on the track quite often if the voltage supply is really low, and friction from the track makes it hard for the car to go.) and that we should have at least tried to stick with an unstable PID code without turning wheels back when a turn was necessary. If there was more time, we would definitely want to work on solving the problem with PID, as well as properly implementing turns and be more patient with that code instead of trying to rely on very low circumstantial/ low chance of working design setups. In all honesty, we had a base of data, but didn't want to keep messing around with a project that already seemed to work on earlier attempts of clearing the track. This project had a lot of different approaches, and there were definitely more optimal designs/methods out there that we decided against simply for the reason of already having a working implementation in the beginning, which led to the ultimate downfall race day.

5) Illustration Credits:

[1a]TA Ablaikhan Akhazhanov

6) References:

[1] https://forum.arduino.cc/index.php?topic=147582.0
[2] https://www.arduino.cc/en/Main/Software
[3] https://docs.google.com/presentation/d/1E_UEbhAMPBCL9RscxnQi86TiOLn7g-Ay3j9OQgBnRXI/edit#slide=id.g20628ef536_0_83
[4] https://diyhacking.com/arduino-hall-effect-sensor-tutorial/
[5]https://docs.google.com/spreadsheets/d/1R_QLqnwsm6ZJhBcxKtCkCPWkDRt5bilfgA2Y39tFmUc/edit?usp=sharing
[6] https://www.youtube.com/watch?v=VumbSBMLCv8
[7] https://www.youtube.com/watch?v=nLZDP94sAiw
[8] https://www.youtube.com/watch?v=zosNfPxSdYw
[9] https://www.youtube.com/watch?v=H6qFi3fzNuU

7) Arduino Code: (submitted as a multiple file)