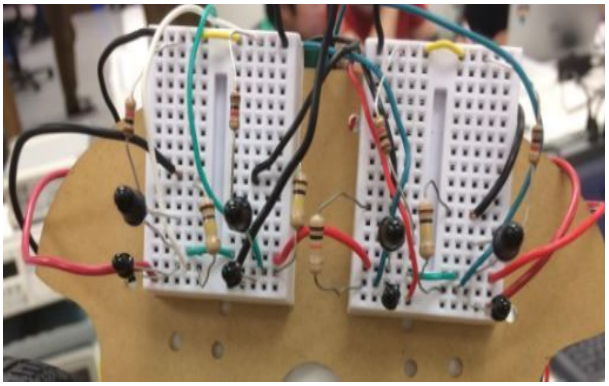
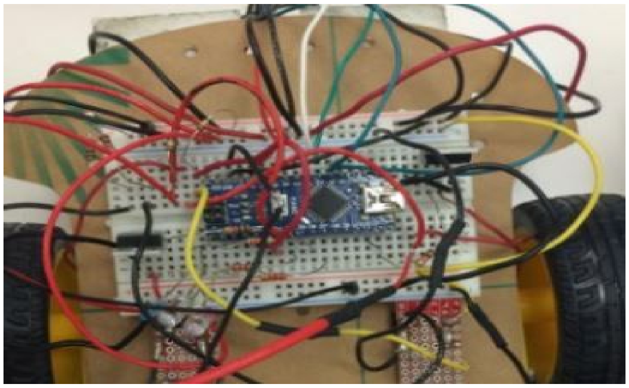
EE3 Final Report

Dennis Zang ID: 704766877
Eric Zhang ID: 104781338

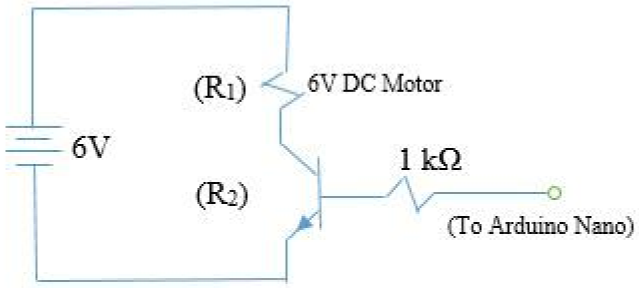
Introduction

In summary, the goal of this project is to design and build a rover that can navigate itself and follow a black-line track. By using some of the basic concepts we had learned in class, from circuit basics to using transistors, we are to make use of an Arduino Nano 3.X and program it with a C-like language in order to control the circuit that runs the entire car. The primary components that were provided include the Arduino Nano microcontroller, a breadboard, 2 6V DC motors, 2 TIP120 transistors, LEDs (infrared and visible blue, green, and red), 1K and 100K resistors, and the EE-SX1042 microsensors (LED and phototransistor pair) (optional). Given these and only these parts, we decided to implement a simple finite state machine that changes its wheel speeds to specific values to accommodate turning and straightening its path. Specifically, we implemented a row of 4 LED-phototransistor "line-readers" in a row, and depending on which reader(s) detected a black line, the car would turn so that it would be centered on the path (with the turn's sharpness depending on which sensor was triggered).



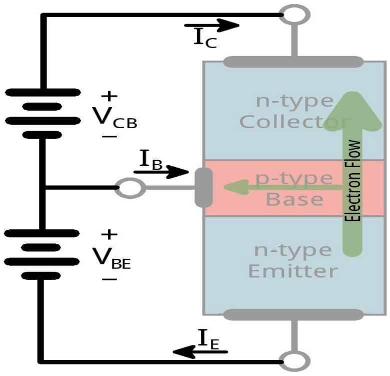


A great part of the project is based solely on the use of transistors and PWM signals. For the case of the 6V DC motors, we want to control the wheel speeds so that the car would be able to move and turn at specific speeds and angles. In our case, the circuit for the wheels is something like this: an external 6V battery source whose cathode is connected to one of the motor's terminals and the other connected to the Arduino Nano's (common) ground, the other motor terminal entering the TIP120 transistor's collector, with the emitter connected to the ground and the base connected to one of the Nano's digital pins by a 1K resistor. The circuit is shown below.



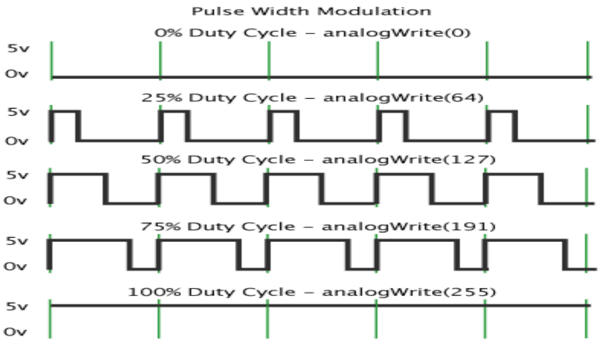
The general layout of the motor-transistor circuit.

The main idea here is to control the amount of current that enters the DC motor. To use transistors, one must know the main idea of what makes the transistors work. The transistor is a semiconductor circuit that essentially acts like a switch between its collector and emitter terminals, which is controlled through its base terminal. Basically, in an NPN transistor, to allow for current to pass through from the collector to the emitter, a higher voltage must be passed from the base to the emitter to allows electrons to pass from electron-rich regions (N-type junction) to electron poor regions (P-type junction).



(source: https://cdn.sparkfun.com/assets/learn_tutorials/1/9/3/transistor-active-electrons.png)

In our case, we would control how much current flows through the transistor by controlling the voltage between the base and the emitter using pulse-wave modulation (PWM). For the sake of the project, the Arduino Nano provides the use of square waves, and to provide a voltage between 0V-5V, we code the Arduino Nano to output (analogWrite) the (5V) output voltage at a specific frequency to get specific voltages to allow for specific amounts of current through the TIP120 transistor. Specifically, in our project, we use "analogWrite" on the Arduino Nano to output PWMs of specific duty cycles (0 for 0% duty cycle, 255 for 100% duty cycle) for specific average DC voltages.



(Note)
In the case of a square wave, we can calculate the average voltage by multiplying the amplitude by the duty cycle.
In this example, the average voltages would be:
0V
1.25V
2.5V
3.75V
5V

(source: https://www.arduino.cc/en/Tutorial/PWM)

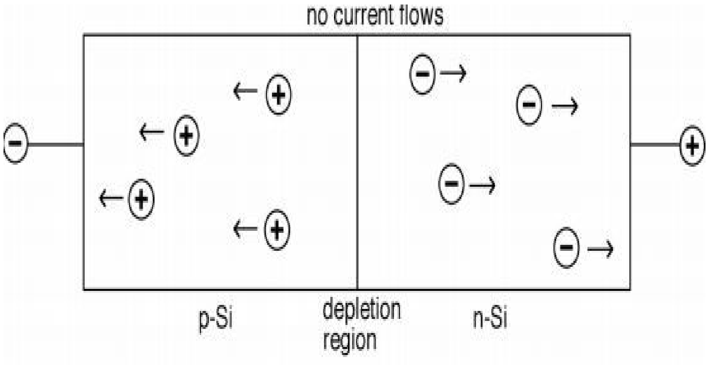
The main idea of the motor-transistor circuit is to control the actual amount of voltage that goes across the DC motor. If were were to put this in terms of the voltage divider equation: (credits to http://www.electronics-tutorials.ws/amplifier/emitter-resistance.html)

$$V_{out} = V_{in}(\frac{R1}{R1 + R2})$$

We can see that by controlling one of the resistor's resistance in the circuit, we can control the voltage across the other. In this case, looking back at the diagram for our motor-transistor circuit above, suppose that R_1 is the DC motor and R_2 is the transistor. By inputting voltages into R_2, we can decrease its resistance (which is considered to be much greater than that of the DC motor) and control the amount of voltage (or current) across R_1.

Another related topic needed for this project is the knowledge of diodes, specifically LEDs (light emitting diodes). A diode can be described as a device that only allows electrical currents to pass through one direction, and for given voltages, changes the effective resistance across the diode. Below, we can see how a diode is implemented and how it functions for specific voltages.

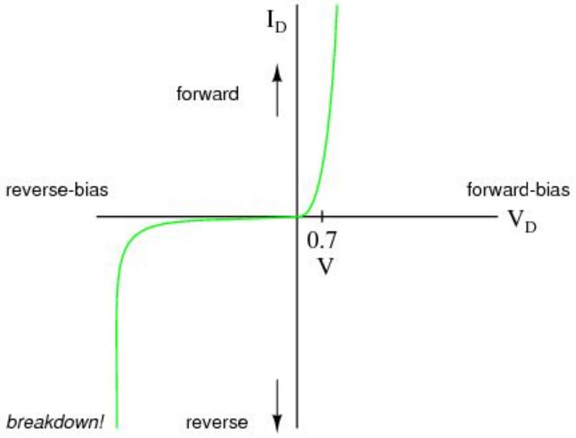
(a) Diode implementation



(Note)
Here, the P-type material is "lacking" in electrons, while the N-type material has surplus electrons. So, the positive charge carrier would move from P to N, while the negative charge carriers would move from N to P.

(source: Dennis Briggs, https://ccle.ucla.edu/pluginfile.php/2059945/mod_resource/content/0/Semiconductor%20Lecture%2017F.pdf)

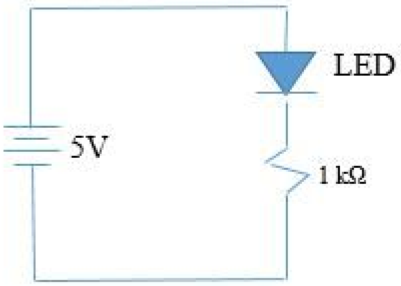
(b) Graph of voltage vs current



(Note)
Here is a graph for the voltage vs current across a diode. As we can see here, the current going through as the voltage is positive increases exponentially, while for a negative voltage the current is severely limited to a small value. Now, if we consider the inverse of the slope, ehich would be the resistance, we can see that for an increasing positive voltage, the resistance approaches 0 and for negative voltages, the resistance is extremely large (and approaches infinity).

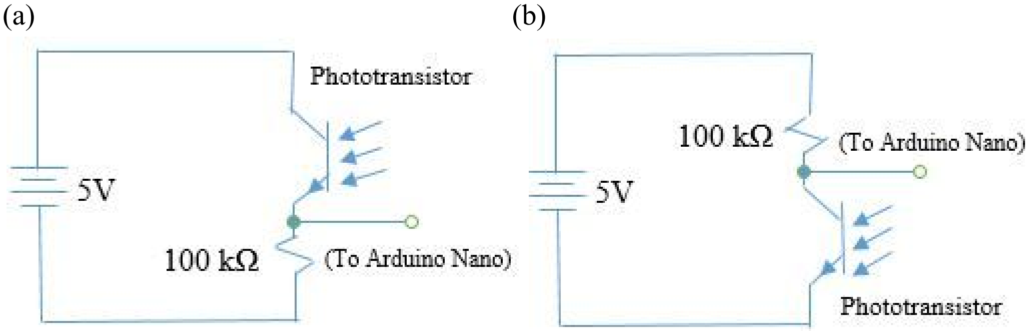
(source: https://www.allaboutcircuits.com/textbook/semiconductors/chpt-3/introduction-to-diodes-and-rectifiers/)

In (a), we see that a diode is made up of a P-type junction and an N-type junction, which only allows for charge carriers to pass in one direction efficiently. In (b), the main idea is that the resistance (inverse of the slope of the graph) decreases exponentially for positive voltage across the diode, and increases exponentially if it's negative. For the sake of the project, we see that in order to use the LEDs correctly, we need to both orient it correctly in the circuit and allow for only a specific amount of current through in order for it to function to emit light, for it essentially acts almost like a short circuit.



The general layout of each LED circuit.

For the infrared phototransistors, they work like the TIP120 transistors, except they take infrared light as input to control how much current passes through it. To use the phototransistors to read whenever the external lighting is light or dark, we need to construct the circuit in a particular way in order to have the Arduino Nano be able to read different input voltages for each condition. Below are two ways this can be done (mentioned in the week 3 lecture).



Emitter-follower circuit

Common-emitter circuit

In (a), the emitter-follower circuit, the input to the Arduino Nano depends on the resistance of the phototransistor placed before it on the circuit. When the reading is dark, the resistance is high and the voltage at the emitter junction would be close to zero (low), and when bright, the resistance is low and the voltage at the emitter junction would be close to 5V (high). In (b), the common-emitter circuit, the input to the Arduino Nano would be 5V (high) if the resistance across the phototransistor is high (due to dark readings) and 0V (low) is the resistance across the phototransistor is low (due to bright readings). For our project, we were recommend to use the former (the emitter-follower circuit), as it would not mean that there would be current flowing into the Arduino by the default dark readings in the common-emitter circuit.

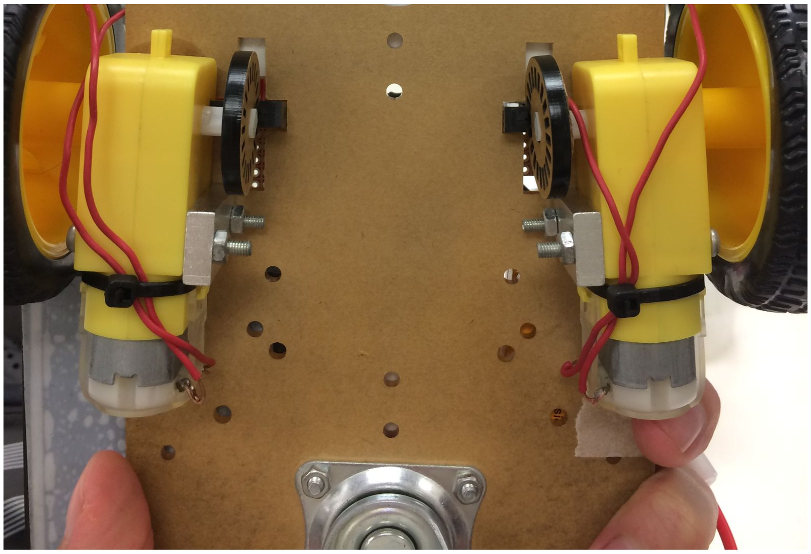
Putting all of these together, the project can be simplified to handling all of these circuit properly with the Arduino Nano. In this project, we programmed the Arduino Nano using a C-like language to input and output voltages to different circuits For reading voltages from the phototransistor circuits, we used "analogRead" to read values from 0-1023, "analogWrite" to output PWMs at specific duty cycles for specific voltages, and "digitalWrite" to output 5V to the required LEDs (note that there are on specific digital pins for analogWrite on the Arduino

Nano). The code basically tells the Arduino Nano to “analogWrite” and “digitalWrite” for given conditionals from “analogRead”.

Testing Methodology

How We Designed the Test

For the test the wheel speed sensing subsystem, we first needed to understand the mechanics of the photointerrupter we were provided. A photo interrupter (the EE-SX1042) has an LED on one leg and a phototransistor on the other. When there is just empty space between the legs, the interrupter will send a high value to the arduino, but if the gap is blocked the value will be significantly lower. We can use the difference in these values to measure wheel speed by having a disc with 20 spokes and gaps that spins with the wheel of the car. The photointerrupter will read when a spoke passes it or a gap does. We used these functions to test the wheel speed by adjusting the PWM of the wheels and monitoring how the photointerrupters read that change in speed in both wheels. In the image below, you can see how the photointerrupters are positioned with the disc in order to read wheel speed.



How We Conducted the Test

To conduct the test, we needed a way to translate the measurements from the photointerrupter into useful data that we could use to measure wheel speed. We referred to the function attachInterrupt() **[1]**. This function allows us to measure the number of changing values that passes through a phototransistor while the program is running separately. The code below is designed to test different PWM values and measure the number of interrupts using the phototransistors to calculate the RPM. We collected data until the RPM reached zero.

Below is the code that we used to perform this test:

(idea to use “attachInterrupt()” and “detachInterrupt()” functions accredited to http://gammon.com.au/interrupts and https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/)

```
//Sensors            Motors                LEDs
//sensor1 = 2        motor_left = 5        left_led = 6
//sensor2 = 3        motor_right = 11      right_led = 7

//Wheel Speed (PWM)
int LSpeed = 145;
int RSpeed = 110;

//Interrupt Counters, Functions, and other Parameters
unsigned long Lcount = 0;
unsigned long Rcount = 0;

void Linterrupt() {
  Lcount++;
}

void Rinterrupt() {
  Rcount++;
}

void setup() {
  //Sensors
  pinMode(2,INPUT);
  pinMode(3,INPUT);
  //Wheels
  pinMode(5,OUTPUT);
  pinMode(11,OUTPUT);
  //Serial Monitor
  Serial.begin(9600);
}
void loop() {
  //0 and 1 correspond to digital pins 2 and 3
  attachInterrupt(0, Linterrupt, CHANGE);
  attachInterrupt(1, Rinterrupt, CHANGE);
  analogWrite(5, LSpeed);
  analogWrite(11, RSpeed);
  unsigned long start = millis();
  unsigned long duration;
  //delay until at least half a second has passed
  while((duration = (millis()-start)) < 500);
  detachInterrupt(0);
  detachInterrupt(1);
  double Lrpm = ((double)(Lcount*60000))/(40*duration);
  double Rrpm = ((double)(Rcount*60000))/(40*duration);
  Serial.println(Lrpm, 1);
  Serial.println(Rrpm, 1);
  Serial.println('\n');
  …
  //(loop for blinking led and resetting PWM when stopped)
  …
  Lcount = 0;
  Rcount = 0;
  LSpeed -= 5;
  RSpeed -= 5;
}
```

How We Analyzed the Test Data

In each trial, we changed the PWM for each wheel and observed how it affected the RPM for each of the wheels. Over the several trials, we also calculated the duty cycle to be consistent with the "analogWrite" value. However, we discovered that when we put the same analogWrite value (same voltage) into each of the wheels the left one had a lower RPM. We observed that this was the case for every trial where the PWM was the same.

Tested values

Right "analogWrite" Value	Duty Cycle %	Right RPM
135	52.94%	133
130	50.98%	126.5
125	49.02%	120
120	47.06%	113.5
115	45.10%	107
110	43.14%	101
105	41.18%	94.5
100	39.25%	88
95	37.24%	81.5
90	35.29%	75

85	33.33%	79
80	31.37%	62.5
75	29.41%	56
70	27.45%	49.5
65	25.49%	43
60	23.53%	36.5
55	21.57%	30
50	19.61%	23.5
45	17.65%	0

Left “analogWrite” Value	Duty Cycle %	Left RPM
135	52.94%	94.5
130	50.98%	89.5
125	49.02%	84.5
120	47.06%	79.5
115	45.10%	74.5
110	43.14%	69.5
105	41.18%	64.5
100	39.22%	59
95	37.25%	54.4
90	35.29%	49.5
85	33.33%	44.5
80	31.37%	39.5
75	29.41%	34
70	27.45%	29

65	25.49%	25.5
60	23.53%	12.5
55	21.57%	0

Actual PWM duty cycles used/considered for equivalent wheel speeds for the actual car

Left “analogWrite” Value	Duty Cycle %	Equivalent Voltage (Left)	Right “analogWrite” Value	Duty Cycle %	Equivalent Voltage (Right)	RPM
220	86.27%	5.18V	180	70.59%	4.23V	133.5
180	70.59%	4.23V	150	58.82%	3.53V	117
160	62.75%	3.77V	128	50.20%	3.01V	106.5
135	52.94%	3.18V	105	41.18%	2.47V	94.5
90	35.29%	2.12V	70	27.45%	1.65V	49.5

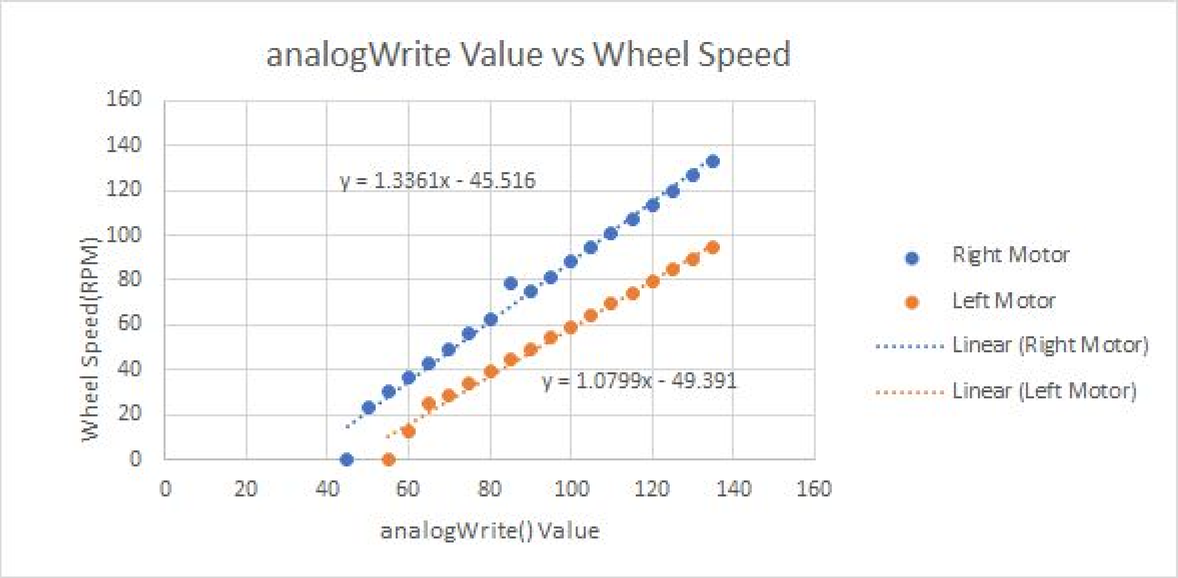
How We interpreted the Data

Using the data collected, we say the discrepancies between the two wheel motors. The left wheel motor needed more voltage in order to reach the same RPM as the right wheel motor. We adjusted the PWM duty cycle inputted into each of the wheels until equivalent speeds were reached. This is shown in the last data table, where we repeatedly guessed and experimented. With various duty cycles until equivalent wheel speeds were achieved.

Results and Discussion

Test Discussion

Graph below was created using data from the tables above.



From the graph above, we can see that as the value of analogWrite value increases, the RPM will increase for the corresponding wheel. The slopes of the right motor trendline and the left motor trend line are very similar. For the right motor trend line, the wheel speed increases by 1.3361 RPM for every increment in the analogWrite() value. For the left motor trend line, the wheel speed increases by 1.0799 for every increment in the analogWrite() value. However, the left motor trendline has a significantly lower y-intercept of -49.391 compared to the y-intercept of -45.516 of the right motor. This difference means that the left motor required a higher PWM to reach the same speed as the right motor.

Using this information from the graphs above, we can set the PWMs of each wheel so that they will have the same speeds and know what is the minimum PWM we can use for the actual car. We also gathered the information of how much PWM the wheel would stop at. The data also helped with the main objective of completing the course because we were able to set the PWM of each motor to get the most accurate turns as well as get the car to go straight with stability.

Race Day Results

On race day we were able to complete the track on the second try with a time of 19.5 seconds. On the first try our car went slightly off track when it was going straight. This slight deviation caused the car to go off course. The extra portion of our car performed correctly on the first try. We uploaded the code, and the wheels of our car started and slowed down until the wheels stopped. The LED flashed five times per second, and the wheels started up again. The only thing we forgot was to use pinMode() to set the LED pins to output, so our LEDs were dimmer than usual. However, this slight error required a simple two line code fix.

Conclusions and Future Work

Our project met most of our expectations. On race day we expected to complete the track without much difficulty within at most five tries because we had been profusely testing our car during the days before race day, and it had been working consistently on those days. We knew from our tests that if our car went off track the error probably came from an improper initialization with the sensors in the start or a slight error during the car's journey that would lead it off track. These errors happened around every three tries. Our extra credit portion worked exactly as planned with the wheels slowing down properly and the LEDs flashing on time.

During the process of creating this car, we learned a lot about circuitry, and how to implement code along with it to create a fully functioning device. Some of the learning came in class and some of the concepts we had to discover ourselves using online resources. However, the biggest takeaway was learning how to plan out a project and efficiently test a function that we had created. These parts were surprisingly the most time consuming.

If we had more time, we would have extended our project by making our car faster. We would increase the PWM in the wheels. However, this increase in speed would make the car very unstable as it went along the path. We would need to test it on the course several times and make adjustments in the code as we went along. It might even be helpful to reimplement our code to account for more error that the car might make as it went faster. It would also be interesting to implement a way for the car to find the black line of the course if it ever goes off track. We would write code that would let the car know it's off the track and to go backwards until it goes back on track. The car would also need to know how to readjust itself to keep going. To test this function, we would have to force the car to go off the black line in the code and to let it find its way back. There are definitely plenty of functions that can be added to the car.

References

“analogRead().” *Arduino Reference*, 15 Nov. 2017, https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

“analogWrite().” *Arduino Reference*, 15 Nov. 2017, https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/

“AttachInterrupt().” *Arduino Reference*, 15 Nov. 2017, www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/.

“digitalWrite().” *Arduino Reference*, 15 Nov. 2017, https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

"Emitter Resistance And The Transistor Emitter Resistor." *Basic Electronics Tutorials*. N. p., 2013. http://www.electronics-tutorials.ws/amplifier/emitter-resistance.html

"Gammon Forum : Electronics : Microprocessors : Interrupts." *Gammon.com.au*. N. p., 2017. Web. 8 Jan. 2012. http://gammon.com.au/interrupts

“NPN Epitaxial Darlington transistor” Fairchild Semiconductor Corporation. Nov. 2014. http://www.mouser.com/ds/2/149/TIP120-890130.pdf