Jaekwan Ahn, Jeffery Yang
Professor Briggs, Dr. Xin Li
EE3
17 June 2017

Building a Line-Following Race Car

Introduction and Background:

The project goal is to build a car that can follow a black line on both straight and circular tracks. In order to accomplish this task, we had to wire up a programmable motor system along with a two-photodiode sensor system. Our design fit under the LMGT category. The motor system controlled the movement of the wheels, and the photodiodes detected the car's position with respect to the black line. Some groups used an additional Hall sensor to detect magnets placed on the track, but we decided not to implement this sensor to forgo additional complications.

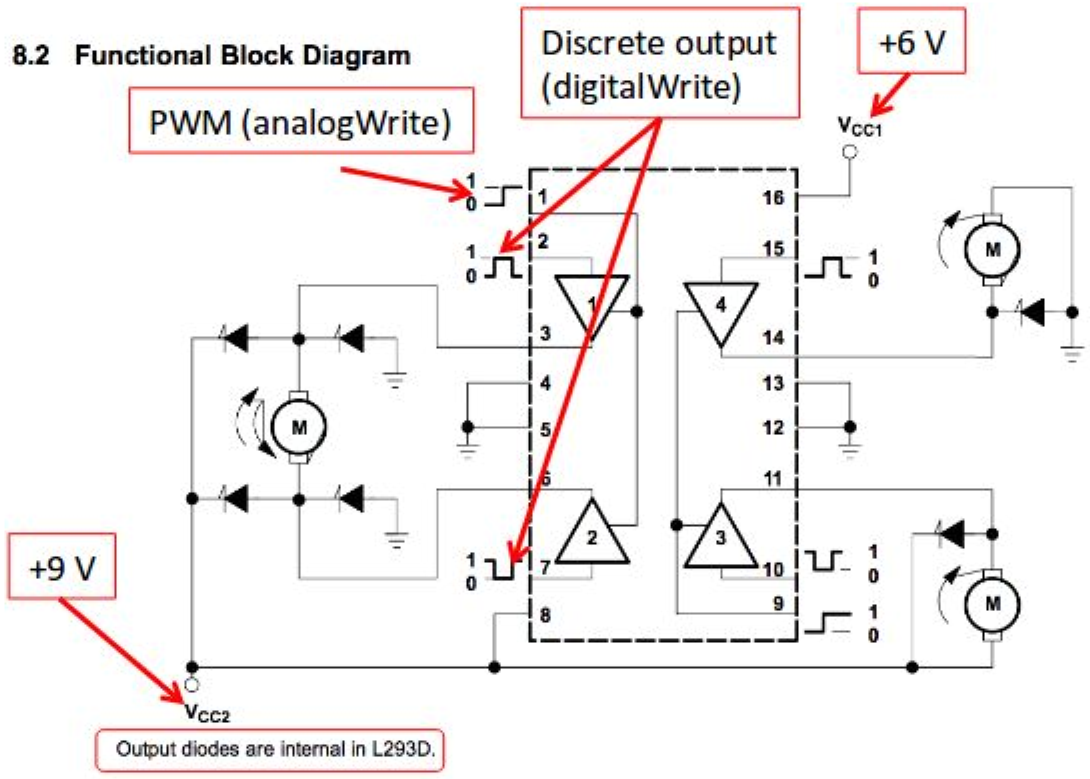
Our design used the following components:

- Chassis and wheels (1): The chassis or body of the car allowed us to put two wheels on the side and one wheel in the back. The two side wheels are controlled by motors, and the back wheel is there to stabilize the car.
- Motors (2): The motors can operate at different speeds, thus determining whether the car travels forward, left, or right. The motors have two wire connections, and a potential difference across the two connections causes the motors to spin.
- H-Bridge (1): The H-Bridge allows us to control the motors via an Arduino microcontroller. The H-Bridge allows for the Arduino to control the speed of the motors via PWM (Pulse Width Modulation). Without the H-Bridge, it can hard to control how fast the motors spin. The two wire connections from the motors are connected to the H-Bridge.
  NOTE: The Arduino is directly connected with the H-Bridge, not the motors.
- Photodiodes and IR LEDs (2 each): The IR LEDs emit light signals, and the photodiodes detect the strength of the surrounding light. When the photodiode is over the line, surrounding light is absorbed by the black line, which causes the photodiode to detect less light. Thus, the photodiode reading would decrease when it is over the black line. We have two photodiode and IR LED circuits wired up on the left and right side of the car, which allows us to position the black line in between the two circuits. We can use the photodiode readings to tell the car which direction it should proceed.
- Arduino Microcontroller (1): The Arduino is the central component of the car. It interacts with both the H-Bridge and the photodiode-IR LED circuits. It takes input from the photodiode readings and determines what speeds the motors should run at for the car to stay on the black line. NOTE: The H-Bridge circuit with the motors is separate from the photodiode-IR LED circuits, but both have connections to pins on the Arduino.
- 9V Battery (1): To power the circuit.

Testing Methodology:

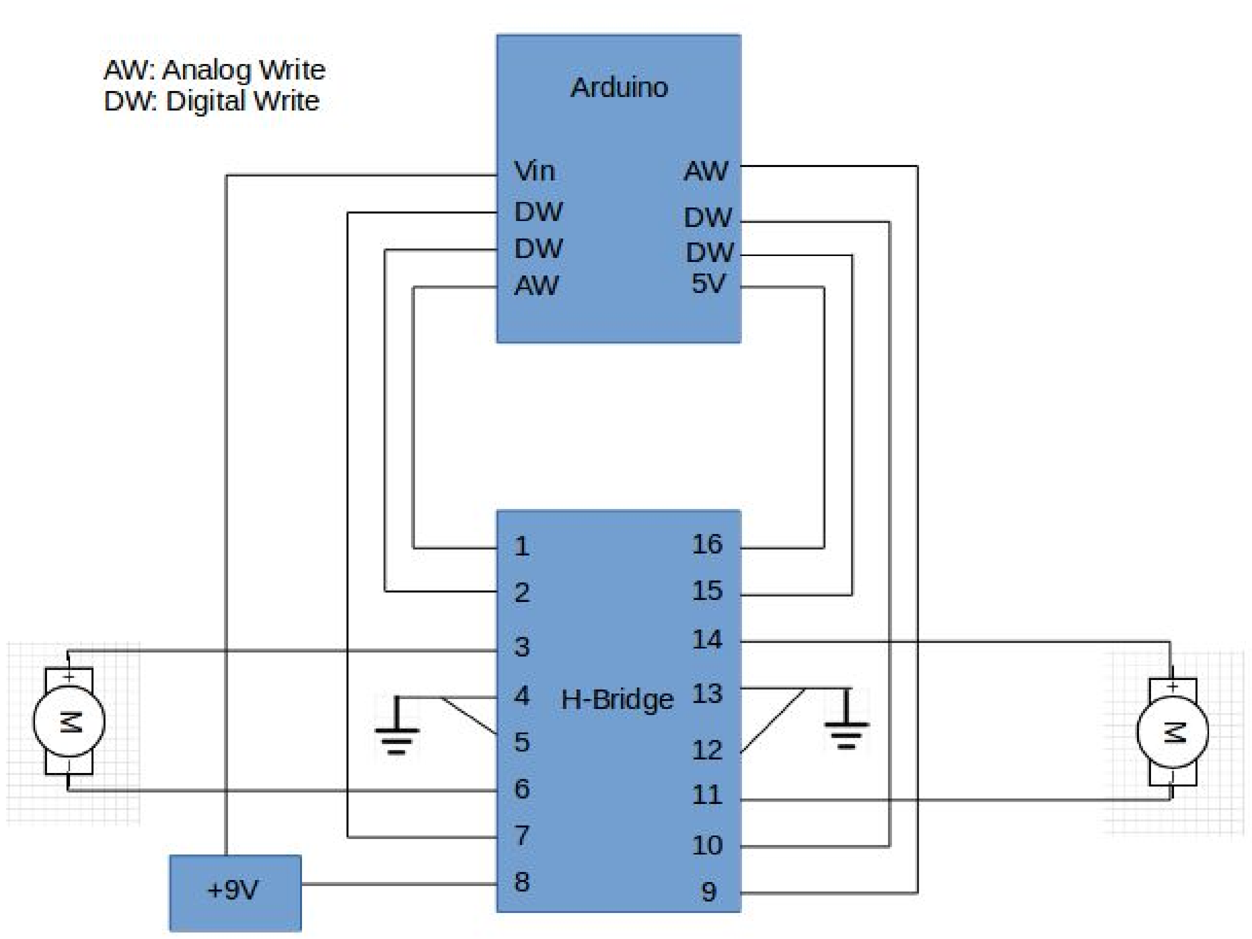
<u>How We Designed The Test</u>

The instructors of our EE3 class did a wonderful job providing us with schematics to help us construct a working design. After we finished assembling the chassis and wheels, we went straight to developing the H-Bridge motor circuit. The functional block diagram below enabled us to develop a working circuit that allowed the Arduino to control the speeds of both motors:


8.2 Functional Block Diagram
Discrete output (digitalWrite)
+6 V
PWM (analogWrite)
V_{CC1}
+9 V
V_{CC2}
Output diodes are internal in L293D.

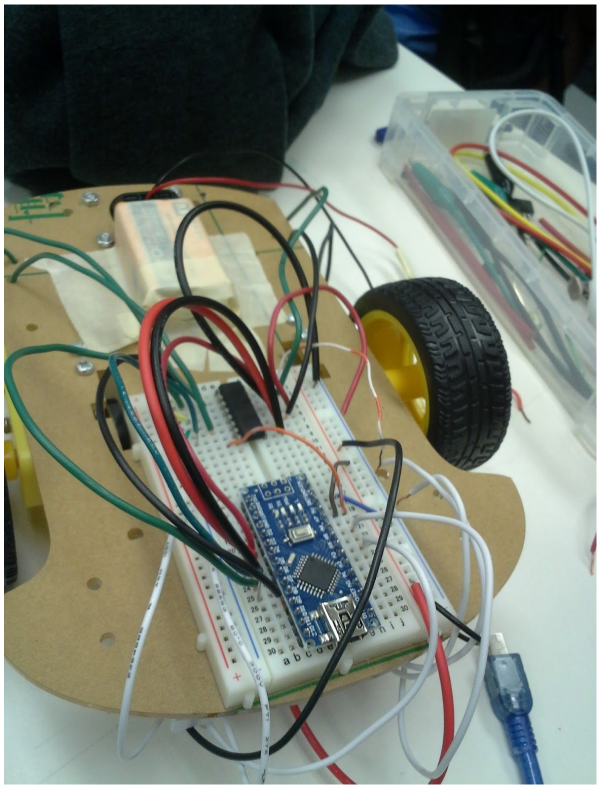
Credit: EE3 Instructors

However, we had to modify the circuit slightly after one of the groups blew up their 6V battery. We changed from using a separate 6V battery to powering the H-Bridge with 5 volts from the Arduino. Below is a schematic of the Arduino-Motor-H-Bridge circuit. Note that all of our grounds were connected on a breadboard. The 9V battery is powering both the H-Bridge and the Arduino. The Arduino GND pin and the negative end of the battery also have to be connected to the ground on the breadboard.



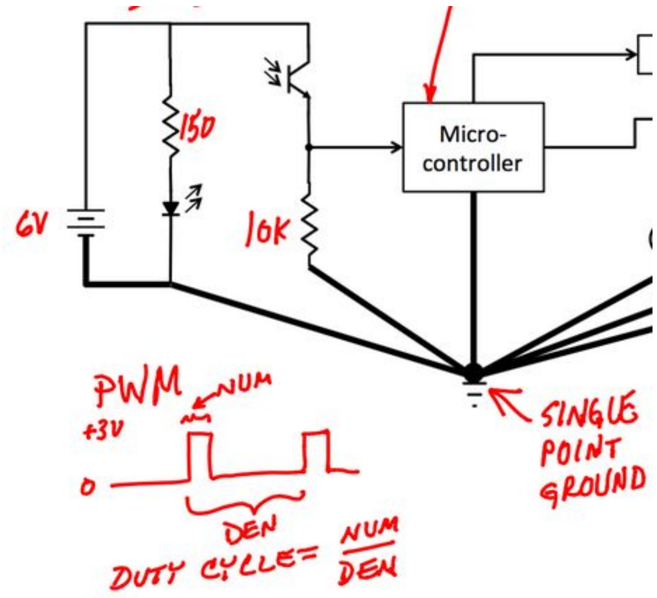
Credit: Jeffery Yang

After we constructed the above circuit, we uploaded a quick program to make sure the motors would turn and stop when commanded by the Arduino. For the digitalWrite commands, we had to make sure that one pin was set to HIGH while the other was set to LOW [1].Then, we set the analogWrite command to output the desired speed to the motor. If we wanted to stop the motor, we just have to set the analogWrite parameter to 0.



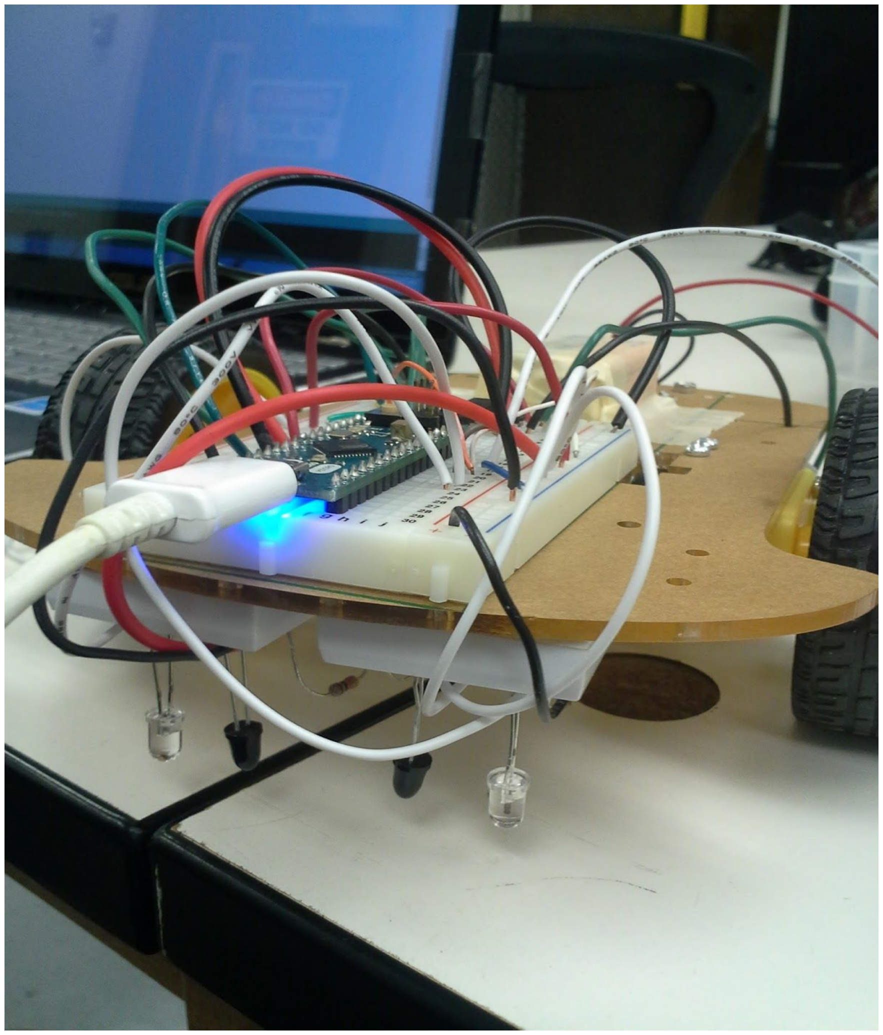
Arduino-Motor-H-Bridge Circuit

After we ensured that the motors were working fine, we proceeded to create two photodiode and IR LED circuits. These two circuits were much simpler to wire up; the main complication was making sure that the anode/cathodes and collector/emitters were in the right place.



Credit: EE3 Instructors (Xin Li probably created this)

As you can see from the schematic, it’s important not to flip the anode and cathode of the IR LED. The anode connects to the 150 Ohm resistor, and the cathode connects to ground [2]. Similarly, we had to make sure that for the phototransistor, the collector was connected to power and the emitter to the Arduino and 10 kOhm resistor [3]. We were able to accomplish this using data sheets. Again, the incident where a group’s 6V battery exploded caused us to change the power source of the sensors to the 5V output from the Arduino. The single point ground is also important, because we had to make sure the two ground lines along the breadboard were connected. The following picture is the best image we took of the sensor circuit.


Sensor Circuits

Notice that there are wires coming from below the chassis out onto the ground lines of the main breadboard. The same goes for power. You can also vaguely see the resistors connected to the IR LEDs (in black). We positioned the phototransistors on the outside so that the car had a wider range to stay on the black line. If we positioned the phototransistors too close together the car could easily veer off the black line and not know how to return to the line.

Before we tried to combine the sensor system with the motor system, we tested each system separately. We first tried to make the motors go straight and record the PWM parameters required for each motor. We also tried to see what values would cause the car to turn left and right. This was more of a guess and check process simply because each motor spins differently for the same PWM value. Once we configured the motors to travel in a basically straight line, we started testing the sensors. We first placed the car on the line with the two phototransistors equidistant from the black line, and recorded the readings of both transistors. Then, we moved the car so that the left transistor was on the line and recorded the readings from both transistors. The same was done with the right transistor on the line.

How We Interpreted the Data

NOTE: Quantitative data will be presented in the next section. The following is a discussion of our thought process from a more qualitative standpoint.

Once we performed calibration tests with the motors and the sensors, we used our results from the sensor tests to dictate the motors' behavior. There are only three possible cases for the car's movement: forward, to the left, or to the right. If the sensors detected that the car was on the line, we would have the car move straight. However, when the car was slightly away from the line, we had to use both sensors' values to determine whether to turn left or right. If the left sensor's value decreased while the right sensor's value remained high, this mean that the left sensor is closer to the black line and light near the sensor is being absorbed by the black line. Thus, the car should make a left turn, meaning that the right wheel has to move faster than the left wheel. Similarly, if the right sensor's value decreased while the left sensor's value remained high, the car should make a right turn with the left wheel moving faster than the right wheel.

To take our thought process one step further, Jae introduced the concept of flagging our car when it was running on the track. For example, if our car veered off the track completely, we needed a way to bring it back to the track. By setting the flag value to 1, that would mean the car has veered off to the left. Thus, we would need a right turn to get it back on track. By setting the flag value to 2, that would mean the car has veered off to the right. Thus, we would need a left turn to get it back on track. To integrate this concept into our program, we needed to know when to set the flag value to 1 and when to set it to 2. It would make sense to set the flag value to 1 when the car makes a left turn (right wheel moves faster). This way, if the car turns too far to the left, it can use a right turn to get it back on track. It would also make sense to the flag value to 2 when the car makes a right turn (left wheel moves faster). If the car turns too far to the right, it can use a left turn to get it back on track.

With our thought process and data from the sensor calibration, we were able to program the car to follow a black line on both straight and circular paths.

Results and Discussion:

Action	Left Motor (Analog Writes)	Right Motor (Analog Writes)
Straight	255	255
Left Turn	85	255
Right Turn	255	85

These are “analogWrite” values depending on actions(Straight, Left Turn, Right Turn).

When we tested the sensors, there were four possible cases:

	Left Sensor (average)	Right Sensor (average)	Action that the car should make
On The Track	85	82	Go Straight
Heading Right (right sensor doesn’t recognize the black line while left sensor does)	40	95	Left Turn
Heading Left (left sensor doesn’t recognize the black line while right sensor does)	105	35	Right Turn
Off The Track	30	35	Right / Left Turn based on flag

Values are slightly changing so we made bound values for left and right sensors to determine whether it’s recognizing the black line or not.

	Left Sensor	Right Sensor
Bound	73	70

Therefore, when the left sensor reads less than 73, it means that the left sensor is on the black line. When the left sensor reads more than 73, it means that the left sensor is not on the black line. This applies the same with the right sensor, except with a bound value of 70.

Thus, based on the four possible cases above, we made our car make either left or right turns correspondingly.

Conclusions and Future Work:

At first, we focused on completion instead of speed. However, when we first tested our car, completing the course seemed impossible. The car didn't seem like it was detecting the black line. We double checked our calibrations and code several times, and still had no idea what the error was. However, after many experiments, we figured out our car wasn't able to make left turns properly. The problem was just a simple and stupid physical setup mistake. The left sensor was sort of covered by some black tape and wires so it didn't do what it supposed to do. We fixed it by removing objects causing interference to the left sensor and then it worked better; the car was able to complete both tracks.

Here are a couple videos we took when it first started working:

Straight Practice Track:
https://drive.google.com/file/d/0B4vQYsivFEivWkRPX0hrcUlXYkk/view?usp=sharing

Circular Practice Track:
https://drive.google.com/file/d/0B4vQYsivFEivVUNxeHdvSHE3WG8/view?usp=sharing

Now that we could complete the course, we tried to improve our speed. Since the maximum value we could put in "analogWrite" was 255, we replaced the previous values with max values(255) right away but it seemed like such a radical change that the car lost balance and veered off the track. Therefore, we came back to our original working program and started making incremental improvements up to the max values. While improving speed, we also needed to make our sensor threshold values more accurate. As the car speeds up, the sensors should detect our car's position more elaborately. Improving step by step, our car ended up with great records for both tracks.

Here are a couple videos we took of the car on the actual race track:

Le Mans Circular Track:
https://drive.google.com/open?id=0B4vQYsivFEivaHEtbkoyNFIzbE0

SoCal Dragway Straight Track:
https://drive.google.com/file/d/0B4vQYsivFEivODBaaV9CX1BHejQ/view?usp=sharing

Throughout the project, the biggest thing we learned is that errors are always caused by unexpected reasons (especially simple and stupid mistakes). For example, when we first hooked up the Arduino-Motor-H-Bridge circuit, our motors weren't turning because we didn't push the H-Bridge far enough into the breadboard. We also learned to improve our car step by step so that we can see where errors come from.

If we had more time, we would have liked to implement the Hall Sensor so that the car could detect magnets placed along the track. It would be cool to see whether the sensor would improve the speed of our car. In addition, we could have also implemented a shroud to shield the phototransistor from ambient light that is not from the IR LED. To test if the shroud was a good idea, we could run some calibration tests and see if it made it easier to detect when the car was away from the black line.

Furthermore, before taking this EE3 class, we just learned about electrical engineering conceptually and theoretically. Even though we understood circuits, energy, and many other things about the theoretical side of electrical engineering, we had no idea how to apply our knowledge into the real world. This was a great experience for us to learn how to apply our EE knowledge and gain the motivation to study more deeply in the electrical engineering field.

References:

1. H-Bridge Data Sheet: http://www.ti.com/lit/ds/symlink/l293.pdf
2. IR LED Data Sheet: https://cdn-shop.adafruit.com/datasheets/IR333_A_datasheet.pdf
3. Phototransistor Data Sheet: https://www.rapidonline.com/pdf/156408-da-02-en.pdf