ECE 3 Final Report

Betto A. Cerrillos (004965272), Jorge E. Pineda (204971366)
Department of Electrical and Computer Engineering
University of California, Los Angeles

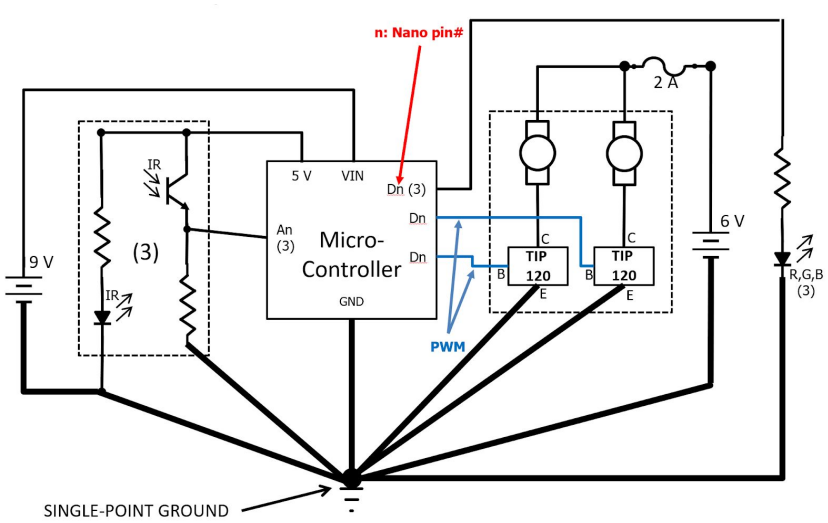
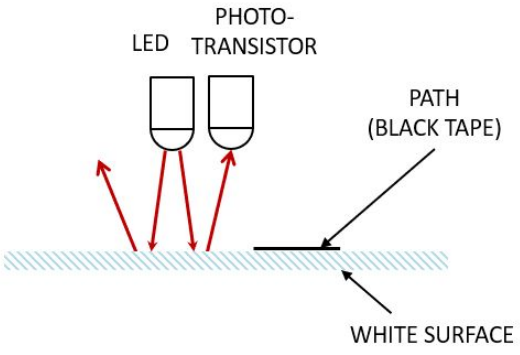


Diagram of the car circuit, provided by our professor.

Introduction and Background

Our project consisted of assembling the circuits and parts in order to build a car that was able to navigate through a taped track consisting of curved and straight sections. The car's design included three phototransistors to sense the track underneath it, motors attached to wheels to actually move the car along the track, and an Arduino Nano microcontroller that acts as the "brain" of the car in that it uses the sensor information to properly navigate the car along the track, and finally a red, blue, and green LED at the top of the car that indicates its current movement (left, straight, right).

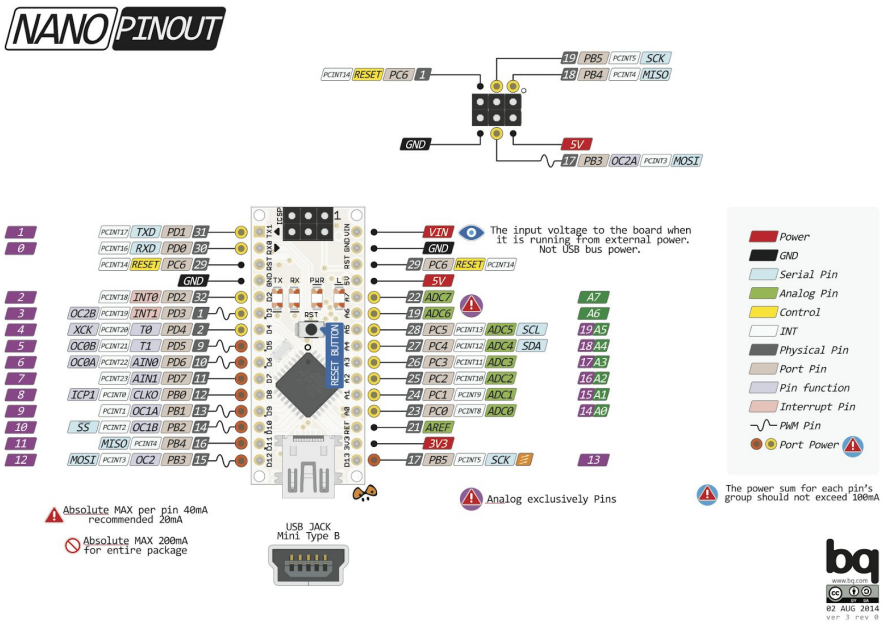


In order to accomplish this, some background was needed in understanding how these phototransistors and motors worked. So as to track the path of movement, each of our three phototransistors was paired up with nearby IR LED that would illuminate the path below. The path was marked by black tape, while the paper it was on was white. This meant that if a sensor was atop the white paper, more of the light would be reflected off and detected by its respective phototransistor, allowing more current to pass from the collector to the emitter of the phototransistor, and thereby allowing more current through the wire the phototransistor is attached on as well (the diagram on the left was provided by our professor).

Furthermore, while connecting the circuit, we had to be careful of which legs of the phototransistors and IR LED went on what end since they have their own polarity. In our case, we used SFH4545 IR LEDs and TEFT4300 phototransistors, which both have their short legs as positive ends, or anodes.

For each motor to work, they first needed their own battery sources, separate from the one powering our microcontroller. However, the motors would only be activated whenever current flowed through as the implementation of transistors would enable us to control the motors' behavior. In order to see how we accomplished this, we must understand first what a transistor does. Transistors have three parts: a base, collector, and emitter. Whenever a high voltage relative to the emitter is applied to the base, current is allowed to flow from the collector to the emitter [1]. Therefore in our design, for each motor, the base was connected to the microcontroller as it dictates when to turn the motor on and off, while the collector is connected to the motor and the emitter to GND.

Finally, the Arduino microcontroller is the main hub for retrieving sensor data from the phototransistors and controlling the motors accordingly, while also turning the LEDs on and off to show which direction the car is headed. We program the Arduino's functionality by uploading our code, written on Arduino software, to the device. The software is unique in that C language is used (which is generally used by most microprocessor programming) with a special programming library built for Arduino circuits. This library allows us to read input from the circuit, as well as send output, which we will use to control the car!



[2] Pinouts for the Arduino Nano

Testing Methodology

After assembling the circuit, we had to verify that the parts themselves worked. First, we focused on verifying that the sensors worked.

IR Sensing and Calibration

Our three pairs of sensors were positioned underneath along the left, middle, and right side of the cart. Each sensor has its own sensitivity, due to nuances in their manufacturing and their exact orientation and positioning on the car. Therefore, we had to run tests in order to calibrate what sensor values we should use as cutoffs in order to tell the cart when to go left, right, and straight.

For this calibration process, we read in the left, middle, and right sensor values through the A3, A5, and A7 analog pins of the Arduino, respectively, and displayed the values on the software's console in order (this meant that these values would appear in the console inside the Arduino IDE while we were still connected to the car). The code excerpt shown below briefly displays our implementation of the two functions we used during the aforementioned procedure.

```
.....
// Read in the values from the respective ports defined as constants
// LEFT_IN, MIDDLE_IN, RIGHT_IN = A3, A5, A7 respectively
svR = analogRead(RIGHT_IN);
svM = analogRead(MIDDLE_IN);
svL = analogRead(LEFT_IN);
// Print out values
Serial.println(svL);
Serial.println(svM);
Serial.println(svR);
...
```

Having uploaded the code, we placed the car above a white piece of paper (to emulate the white paper for the actual track) and viewed the sensor values as it ran. Keep in mind, that our AnalogRead function maps input voltage to an integer range from 0 to 1023 [3]. For simplicity, our analysis rounds these observed values to the nearest multiple of 10. When located above the paper, we observed values that ranged from about 250 on the left sensor, 280 on the middle sensor, 220 on the right . We also observed readings while located over the white table and found them to be similar, meaning that the values were probably a good indicator of what the values would be, when the car is located on the actual track.

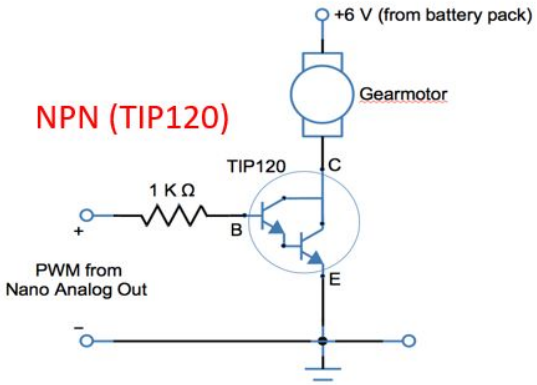
Afterwards, we underwent the same procedures except this time, we placed the sensors over a dark surface, which included dark lines of electrical tape (as the track uses dark tape which we believed was electrical). These gave us values of about 750 on the left sensor, 800 on the middle sensor, and 770 on the right sensor. Below is a summary of our data:

Sensor Placement	Left Sensor	Middle Sensor	Right Sensor
White Surface	250	280	220
Black Surface	750	800	770

It was interesting to see the deviation in values from one sensor to another, though they were all generally within the same ranges of about 200 - 250 atop white surfaces and 700-800 atop dark surfaces.

We now figured the best course of action was to implement a threshold value in our code, based off the sensor values, so as to properly detect when the path of the track is underneath each sensor. Then, the simplest action is to choose a value for the sensors, between the two ranges, as the difference in values when tested on white surfaces versus dark surfaces is large enough that choosing a single value in the middle as a threshold is enough to differentiate between the two. So in our code, we then decided upon a threshold value of 650. For our purposes, we decided to simply use this threshold to set a state for various booleans that dictate whether our car must go right, left, or keep heading straight.

Motor Testing



After connecting our motors as the image above shows, which was a diagram provided by our professor, we first had to make sure that the motors run, at all. So, we wrote some code to apply a voltage to the base of the transistor. An excerpt is shown below.

```
....
// Our output constants: LEFT_MOTOR 11, RIGHT_MOTOR 9
// Set their modes to output
  pinMode(RIGHT_MOTOR, OUTPUT);
  pinMode(LEFT_MOTOR, OUTPUT);
// Then send a signal to the motors (Value is between 0 and 255).
  analogWrite(RIGHT_MOTOR, 255);
  analogWrite(LEFT_MOTOR, 255);
...
```

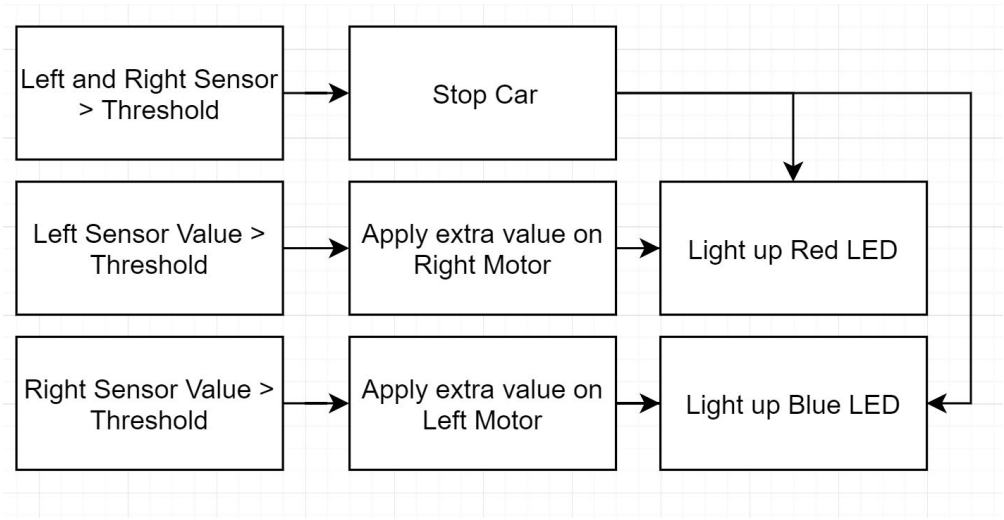
When we ran, the motors worked meaning that the circuit had been properly wired. We then tried to find the minimum value that would make these motors function. So we played around with the second parameter values of the analogWrite function on each motor until we observed reasonable results. We found that our minimum value would be around 130, and so we set both motors to that 130 value. We then placed the car on the ground and observed its movement when going straight. We noticed that it seemed to veer a little off a straight trajectory, and so attempted to adjust the wheels a number of times, as our trials on the table below indicate.

Left Motor	Right Motor	Result
130	130	Veer to the right
130	200	Veer to the left
170	200	Move straightish, but veer slowly to the left
180	200	Move straight but will eventually veer off after a while to the left.
185	200	Move straight for a while but will eventually veer off after a while to the right.

According to our results, we need about a difference of 20 between the motors in order to attain movement in a straight path, although it does continue to veer off after a while. But this raised no concerns, as the car did move in a straight line long enough to the point that it would not steer off a straight track. These values of 180 and 200 served as our baseline for the values of the motors. Moving on to the car's turning functionalities, we first attempted to simply increase the value of one wheel while keeping the other at baseline (e.g. increase value on left motor when making right turns and increase value of right motor when making left turns). We decided adding a value like 50 to the appropriate motor would work, but could not really test it out until we actually put it on the track, which will be discussed in the Results and Discussion section (spoiler: we had to change our approach and completely stop one of the two wheels, rather than speeding up the other).

Combining Sensors, Motors, and LEDs

We then combined these three main components of the system by writing the program so that it reacts to the sensor information by moving in the direction of the sensor that surpasses its threshold value. If the sensor passes the threshold value, then the car will turn in that direction by applying the extra value, on top of the baseline, on the correct motor to make the turn, as displayed in further detail by the flowchart on the following page.



By default we made the car light up the green LED and move forward in a straight trajectory, unless one of the above conditions was met.

Results and Discussion

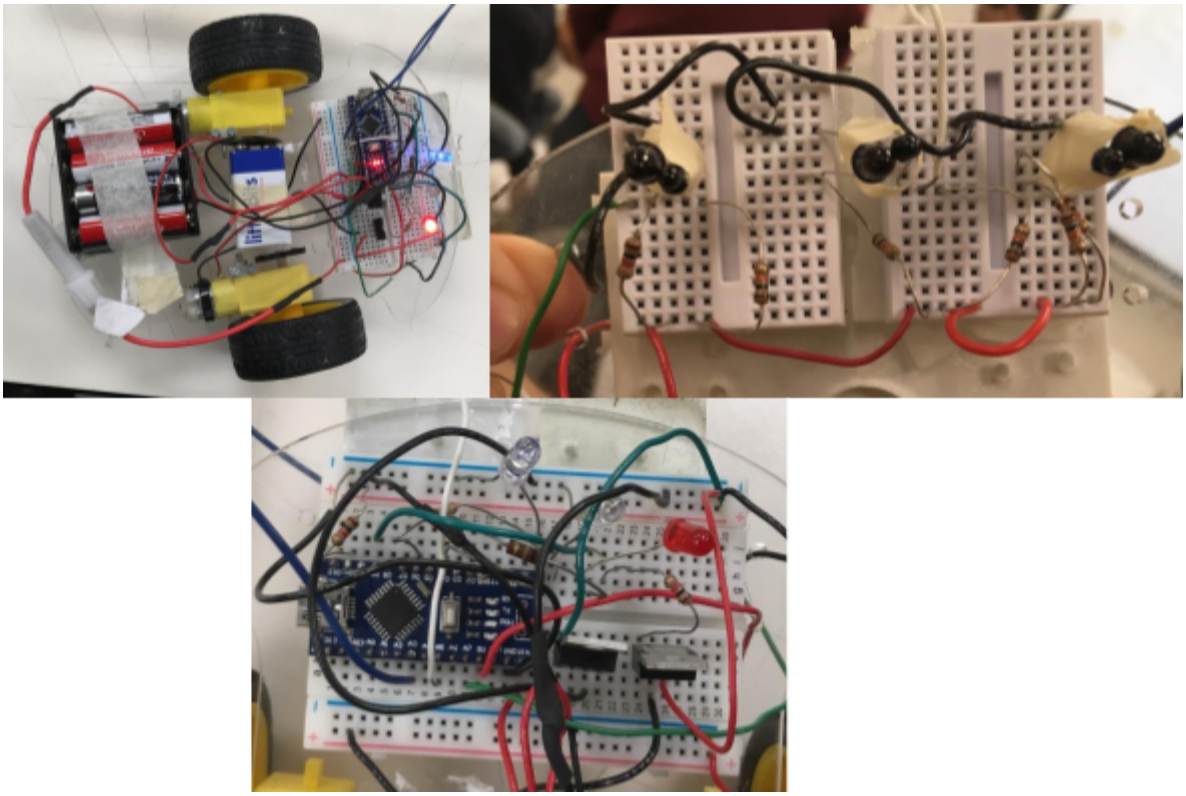
After programming and calibrating our car, we then placed it on the test track to observe how it did.

When we first tried to get our cart to move around the track, it seemed to detect he track below it accurately. However, when it came to turning, the car was unable to complete the turn quickly enough and would veer off the track. At first, we hypothesized the problem to be that the extra value applied to one of the motors did not allow for a large enough difference in rotation speed when making turns, meaning that the turns are not sharp enough. So we first tried adjusting the difference in speed between motors, by increasing the speed of the wheel "dominating" the turn (meaning that for right turns we increased the speed of the left wheel and vice versa). However, this was not successful, so we then tried to increase the difference by instead lowering the speed of the wheel that previously stayed at baseline during the turn. This produced slightly better results, and it was at this point that we realized the reason that our previous method did not succeed when it came to turning could be attributed to the fact that our power supply did not supply enough power to power both motors fully, so we were limited as to how fast each motor could go given the limited supply.

We continued to play around with the values trying to get the car to follow the turn correctly until we found that our best solution was to completely turn off one of the motors, and get the other motor to a high speed value. this made the turns very sharp and slowed down the car but it gave a much more reliable manner of navigating the track's turns. And though, it did keep veering off during some test runs, it was producing much better results. To be safe, and avoid these occurrences where the car seemed to veer off, we lowered the base values for the wheels by 20 (making it 160 for the left motor and 180 for the right motor).

When we ran our car during race day, we were a little afraid at first because we were using different batteries to power the motors (we used 6V instead of our usual 4.8V). Despite the change, the car

completed the track in 39.45 seconds, on the first attempt without any issues. This was a huge success in terms of displaying the reliability of our end solution. However, there was definitely room for improvement as many of the other teams were able to beat our time. For example, we could have taken more time to change the motor values appropriately, to increase the car's speed when moving straight and across turns, but we chose to take a safer route, slowing down our motors as it was much more reliable. Reference [4] shows some videos of our car in action.



Conclusions and Future Work

Our design of the circuit and code was adequate enough to achieve the necessary goal of completing the track. Although we could make more improvements on it as discussed in the section before, this project taught us a lot about integrating circuit technology with microprocessor programming, which was made possible through the Arduino microcontroller that ran all the logical tasks, providing output to certain circuit components based on the input from other components. Furthermore, we learned how to use transistors in a circuit as a method to control other circuit parts, with our motors. Another lesson tied more to application of engineering in the real world was that was the methods of calibration we underwent for certain components, in which we had to run several tests to properly get them to function for our own purposes. If we had more time, we would have liked to integrate more complicated logic for turns to traverse the path faster and even more accurately. This might have included making more graphs and acquiring more data in order to create a rough linear equation for each wheel that could make the wheels increase in speed depending on the sensor values. Overall, our current design was satisfactory, but if it weren't for other classes and the limited time we had for the project, we could have definitely done more.

References (include Illustration Credits)

[1] https://drive.google.com/file/d/195okFO420RNY4pWGs7FTBVjMxMOW_r_V/view?usp=sharing

[2] https://components101.com/microcontrollers/arduino-nano

[3] https://www.arduino.cc/reference/en/language/functions/analog-io/analogread/

[4] https://drive.google.com/open?id=1zCPGFLyE54QrLwT128VTeQgmJEbhL5L0