Imagine RIT NXP Cup Car

Jacob Meyerson
Department of Computer Engineering
Rochester Institute of Technology
77 Lomb Memorial Drive
Rochester, NY 14623

Charlie Poliwoda
Department of Computer Engineering
Rochester Institute of Technology
77 Lomb Memorial Drive
Rochester, NY 14623

***Abstract*—The goal of this project was to develop a program to drive a car autonomously around a track in as little time as possible. This was done by reading pixels from a line scan camera, processing the data, using a servo motor to steer and driving two DC motors in the rear. The project was a success as the car was able to speed around the track in record time on the first of three attempt.**

***Index Terms*—Autonomous Car, NXP Cup, Proportional Integral Derivative Controller, Line Scan Camera, Data Processing**

I. Introduction

With the implementation of autonomous cars becoming more of a reality, it is important that future engineers get exposure to hands on learning behind some of the concepts that are applied to such a complex problem. That is exactly what the NXP Cup is all about. It gives the students an opportunity to break the problem down and understand what it takes to create an optimized autonomous vehicle.

The complexity of the problem invites a multitude of different approaches to each aspect. This was seen with other cars striving in some areas and failing in others. One example of this was the output line data from the line scan camera. Some teams decided to use the derivatives to process the data to find edges while others looked for threshold values. Each car had slight variations in how their motors ran and caused teams to come up with unique values for finding the optimal speeds of their cars.

After hashing out ideas on a white board, discussions with others, and countless trials and errors, optimizations to different aspects of the race were found and implemented. For the camera, line data was first filtered using a five point averaging function. This eliminated some noise and allowed for a more accurate representation of the track. A dynamic threshold for a binary representation was then used. The lighting in the room of the track was not consistent and having a static threshold would cause inaccurate and unpleasant movement of the car.

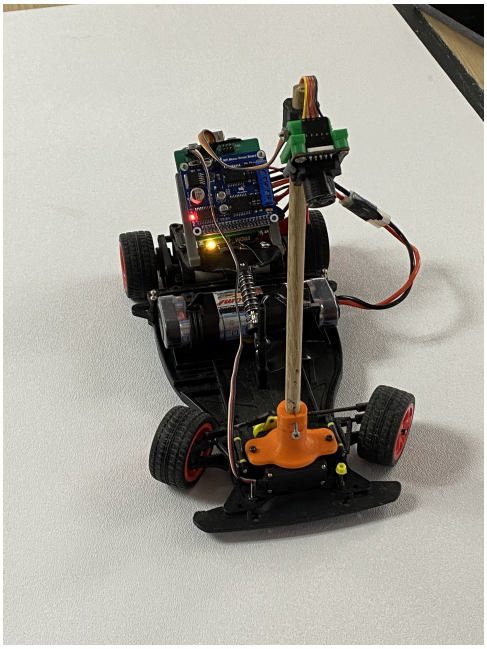

Fig. 1. NXP Cup Car

A binary representation of the data allowed for easy edge detection. With the edges detected, a midpoint was able to be determined for the car and different states were implemented to tell the car if and how much to turn. A PID (Proportional-Integral-Derivative) controller was implemented to cause a smoother driving experience based on track position. Several different modes were then developed to drive the car at different speeds around the track.

The rest of the paper is organized as follows. After the introduction, the next section covers some background information about the project. This includes information about previous car implementations, details about components such as the camera and the motors, the concept of steering, and how the race was approached. Section 3 covers the more specific approach to implementing the camera, motors, steering, and the race strategy. Section 4 lays out the results of the car and the race as well as touches upon some possible improvements. Finally, Section 5 concludes the paper.

II. Background

A. Previous Car Implementations

Naturally, there is no one angle to approach the race with. Depending on how the team conducted their research and implemented their theories, different strategies come about. One example of this would be the line processing methods to find the edges and midpoint of the track. A team competing at Imagine RIT in 2017 used derivatives when finding the center of the track [1]. A derivative function was used to determine when the line data has come across some sort of edge.

B. NXP Line Scan Camera

The NXP Line Scan Camera is able to detect light and dark colors. The lighter the image, the closer to V_{CC} the output reaches. The darker the image, the closer to GND the output reaches. This allowed for car to determine where it was on the track, since the track pieces had white backgrounds with black edges. In addition, the carpet on which the track was placed was also black, so the car could determine if it was fully off the track and could stop without running into a wall or table.

There are five connections made from the K64 to the camera. There are power and ground lines, where the K64 provides 5 V and a reference ground to the camera, a clock (CLK) line and system integration (SI) line, which are both inputs to the camera, and an Analog Out (AO) line, which is the camera's output. The CLK and SI signals were connected to GPIO input pins on the K64, and the AO signal was connected to one of the K64's Analog to Digital Converters (ADC). The CLK for the camera was run at The figure below shows the timing diagram for the line scan camera, obtained from a forum on NXP's website[1].

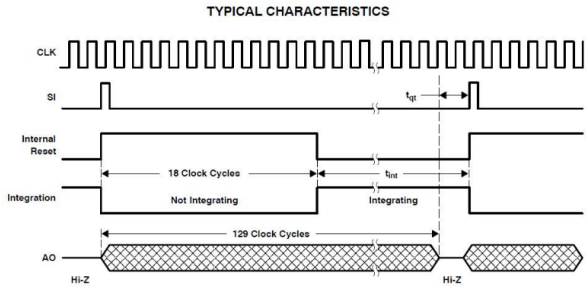

Fig. 2. Line Scan Camera Timing Diagram[1]

When the SI signal gets pulsed high, the camera begins to collect data and takes up 18 clock cycles before it starts to integrate the data and store it in the output array. The limiting factor of the camera was the integration time. The lower the integration time, the faster the data could be read in, and the more samples per second could processed. According to NXP's website about the line scan camera, there is a minimum integration time[2] of 33.75 μsec, and a maximum time[2] of 100 msec. If an integration time of greater than 100 msec is exceeded, then the capacitors on the output of the integrator will saturate, and the data would be ruined[2].

The output of the camera was a CMOS linear sensory array of 128 pixels. As the camera detected turns in the track, the bandwidth of high output got smaller and was shifted either left or right. Based on how far from the standard straight data the output was shifted, the degree of the turn was approximated. In addition, placing the camera at different angles would produce different results, since the car would be tracking different points in front of it. In general, the further the camera was pointed out, the sooner the car would see a turn, and begin to turn. However, there was a threshold where the car would see the turn too early, and then turn off the track.

C. DC & Servo Motors

The car was equipped with one servo motor and two DC motors. The DC motors were attached to the rear motors, and did not share the same axle, so each motor could independently spin. The DC motors were driven with an H bridge, so they could both be driven at independent speeds. This was key for differential steering when the car drove at higher speeds. Going around tight turns, the inside wheel was spun at a slow speed than the outside wheel, creating a tighter turn. At very high speeds, the car had the ability to drift and spin the back of the car around the end. To control the speed of the DC motor, the duty cycle of the PWM signal sent is modified. A 0% duty cycle does not spin the motor, and a 100% duty cycle PWM signal spins the motor at maximum speed.

The servo motor was mounted on the front of the car, and was responsible for turning the front wheels left and right to control the steering. The servo motor's position was controlled by sending a pulse-width modulation (PWM) signal with a specific duty cycle. This servo motor was not a continuous rotation servo, so there was a max turning value in both directions. Typically, the duty cycle range for a servo motor falls between 5% and 8% duty cycle.

D. Smooth Turning

Initially, the concept of "bang bang" turning was implemented, where the steering was limited to three options: full left, full right, or straight. With this implementation, the car was able to complete an oval track and a "figure eight" track. However, the path it took was very bouncy, meaning that during any straightaways, the car would be turning very hard left, then right, then left again as it bounced off of each side of the track. Since a lot of time was lost due to the

[1]Timing diagram taken from NXP's website at https://community.nxp.com/t5/University-Programs-Knowledge/Line-Scan-Camera-Use/ta-p/1105313

[2]Camera Limitations section taken from NXP's website at https://community.nxp.com/t5/University-Programs-Knowledge/Line-Scan-Camera-Use/ta-p/1105313

over steering, a smoother turning algorithm was implemented using a proportional–integral–derivative (PID) controller. The general equation for a PID controller is shown below in (1), where k_p, k_i, and k_d are all constants, $e(t)$ is a function of the error from the desired value, and $u(t)$ is the error adjustment factor.

$$u(t) = k_p e(t) + k_i \int e(t)\,dt + k_d \frac{de}{dt} \tag{1}$$

Because the error was being calculated in real time, the $e(t)$ function in (1) was not a continuous function, and instead was replaced by saving the previous two error values. The full PID equation used in the code is shown below in (2).

$$u(t) = k_p * error + k_i \frac{old\_error1 + old\_error2}{2.0} + k_d(error - 2old\_error1 + old\_error2) \tag{2}$$

In (2), each constant is responsible for a different aspect of the steering controller. The k_p value controls how hard the car needs to turn, the k_i value controls how much offset needs to be added or subtracted to reach the steady state desired amount, and the k_d value controls how soon the car needs to turn. These values were then calibrated for the car at different motor speeds to determine their optimal values for the smoothest possible steering. Values too high resulted in excessive turning, while values too small resulted in under steering, causing the car to fly off the track on turns.

E. General Race Strategy

In order for an optimal time around the track, a few criteria were kept in mind when developing the code. The car was to accelerate on straightaways as they would be good opportunities for increasing the speed. At higher speeds, the car would need to turn sharper and sooner. This was done through camera angle, PID control, and reducing inner wheel speed. With the track being randomized, it was also important to include several different modes. This would ensure that the car would have the necessary tools to tackle any configuration it faced.

III. Proposed Method

A. NXP Line Scan Camera

In order to first test the camera, the output was probed when it was pointed at the center of the white track. The oscilloscope capture is shown below in Fig. 3.

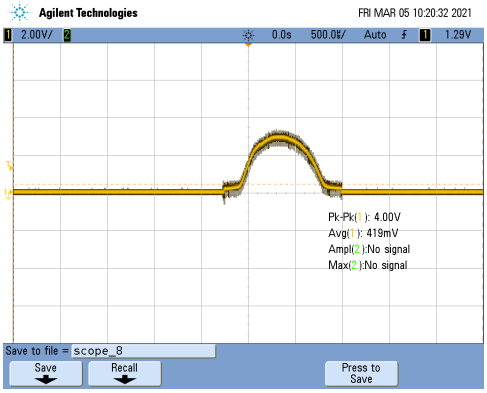

Fig. 3. Oscilloscope Capture of the Camera Facing the Track

The output shown in Fig. 3 confirmed that the camera outputted high values when the light intensity was high. To test the detection of a black edge of the track, the camera was pointed at a black stripe on a white background. That oscilloscope capture is shown below in Fig. 4.

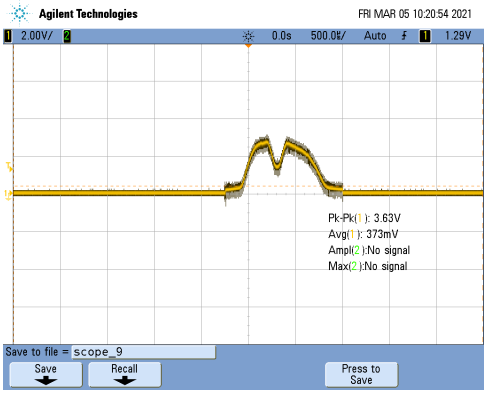

Fig. 4. Oscilloscope Capture of the Camera with a Black Stripe on a White Background

It was expected that the middle of the pulse would dip down, since it should have detected a black background in between the white on both sides. After confirming these two oscilloscope captures were accurate, the camera was mounted on the car and connected to the K64 on the car.

In order to ensure the data from the camera was accurate, the raw camera data was fed through a smoothing filter. The smoothing filter was a five point average, and the pseudo code for its implementation is shown below.

Listing 1. Five Point Averaging Function

```
for(i = 0; i < 128; i++) {
  if(i > 1 && i < 126){
    smoothline[i] = (line_data[i] +
        line_data[i+1] + line_data[i+2] +
        line_data[i-1] + line_data[i-2])/5;
  }
}
```

Each element in the smoothed data was calculated by averaging the raw data at that index with the two data points directly left and directly right in the raw data array. After smoothing the data, it was converted into a binary array in order to perform edge detection. The threshold used was dynamically calculated by searching the raw data for the maximum value read, and then taking 80% of that. Any value equal to or above that was interpreted as a binary '1', and all other values were interpreted as a binary '0'. The binary array code is shown below.

Listing 2. Binary Array Code with Dynamic Threshold

```
for(i = 0; i < 128; i++){
  binline[i] = smoothline[i] >
    (80.0/100.0 * max_value)
    ? 1 : 0;
```

From the binary data, the edges where the array goes from 0 to 1 and then down from 1 to 0 could be easily found, and the midpoint of the range of 1 values would indicate where the car currently was on the track. The dynamic threshold implementation was required in order for the car to ensure it could stay on the track in all corners of the room. Some corners were darker than others, so a hard coded track threshold was very unreliable. It was likely that the car would fly off the track and hit the wall thinking that the track continued straight when in reality it turned, or that the track turned when in reality it was a straightaway. In addition, the dynamic threshold allowed for the integration time of the camera to be reduced down to close to the minimum time of 33.75 µsec. The issue where the output of the camera won't reach the max of V_{CC} with a lower integration time was resolved, because the threshold only depended on the maximum of the raw data, and not reliant on the maximum being exactly 5 V.

B. DC & Servo Motors

Through trial and error, the minimum duty cycle of the PWM signal required for the DC motors to be able to move the car was 50%, and the motors were run at a frequency of 10 kHz. It was also discovered that the left rear motor did not spin as fast nor as smoothly as the right rear motor, so the car would drift to the left on straightaways. The motors were attempted to be calibrated to spin at different speeds and drive straight, but it was unable to be calibrated well enough to make a noticeable impact. In the end, the car was still able to drive straight by using the servo motor to align itself.

The servo motor was run at a 50 Hz frequency. The range of the servo went from a 4.9% duty cycle (full left) to an 8.3% duty cycle (full right). The midpoint of this range, 6.6% duty cycle, drove the car straight.

C. Track Centering and Tight Turning

In order to center the car on a straight track piece, the midpoint index of the high camera data was calculated and compared to the constant midpoint index. Through testing, the optimal midpoint index for the car was determined to be 65.5. Based on this centered value, the amount necessary to turn was determined by how far from 65.5 the adjusted midpoint of the camera data was. Using 65.5 as a centering index allowed the car to be just off center to the left, which made turning both left and right faster with the slower left motor. The car was able to hug the inside of the track turning both directions, which significantly improved its time around the track.

In the code, the threshold for the car to identify it was on a straightaway was plus or minus eight indices from the center index. Any adjusted midpoint in that range was determined to be a straightaway, and the car would be in the STRAIGHT drive state. From the edges of this range, if the adjusted midpoint fell within nine indices of either endpoint, the car was in the SLIGHT_TURN state, where it would spin the inside wheel at 90% of the outside motor speed. Any adjusted midpoint outside of the range of a slight turn was interpreted as a hard turn, and entered the HARD_TURN state. In this state, the inside motor would spin in the range of 5% to 25% of the outside motor speed, depending upon which speed mode the car was driving. The transition from the slight turn to the hard turn allowed the car to begin turning early, and then turn very sharply, and cut the corner of every turn. There was no need to brake on the turns because the car was able to see them early and begin to turn with enough time to take the inside line at full speed.

D. Smooth Turning

The PID control system from (2) was implemented in the code to calculate how much to adjust the raw servo position for smooth steering. To calculate the raw servo position without the full PID implementation was calculated by scaling how far the adjusted midpoint from the current camera data was from the center track index of 65.5. Then, that difference was scaled down by the k_p value and added to the center servo position to set the servo. By calculating the raw servo position in this manner, the sharper the turn, the larger the difference between the center track index and the adjusted midpoint index, so the further the motor would turn. To determine the error, this raw servo value was compared to the previous servo turning value, and the difference was taken. Then, to implement the full PID system, the $u(t)$ function from (2) was implemented, and that offset was added to the previous servo duty value. Before actually writing the calculated servo duty cycle to the motor, it was clipped at the lower and upper bounds, previously noted in Section 3.2. If the calculated value was above 8.3%, it was clipped to be 8.3%, and if the value as below 4.9%, it was clipped to be 4.9%, sine these were the physical limitations of the servo motor.

E. Variable Speed

Being able to move as fast as possible while still remaining in control was a key to success. To do this, acceleration was implemented when the car was on a straightaway. This was

done by steadily increasing the duty cycle of the rear motors each time the car detected it was still going straight. This ensured that the motor speeds would eventually reach their desired limit while also not jumping the gun and losing traction due to too large of an increase.

F. Race Strategy

When it came down to race day, the car was ready to be optimized based on the given layout of the track. The first step of preparation was to develop at least three different modes, each safer than the last. Because there were three opportunities to make it around the track, higher speeds were to be attempted with the first two runs while the last run would be a safe option. Initially, more modes were to be implemented that would account for specific track layouts. Eventually, consistency was reached at three different speeds leading to just the three different modes being implemented. The specifics of each mode can be found in Table I.

TABLE I
CAR RACE MODES

Mode	1	2	3
Color	Purple	Yellow	Green
Max Duty Cycle %	100	95	85
Min Duty Cycle %	90	85	75
Hard Turn Speed % (Inside Wheel)	5	10	25
k_p	0.10	0.115	0.13
k_d	0.79	0.75	0.75

In order to implement the different modes, arrays were instantiated in the program holding the values at their respective indices. Both Switch 2 and Switch 3 on the K64 board were utilized to cycle though the different modes. A counter was incremented with each press of Switch 3 and the value was modded by three, the number of modes. The LED was also used to have a visual of which mode was currently selected. When Switch 2 was pressed, the loop was exited with the index value for each array as the result.

For the fast mode, the rear motors were driven at full speed, 100% cycle. The duty cycle of the inside wheel on the turn was also to be scaled to a near stop. This allowed for sharper turns. The k_p value was reduced to allow for smaller adjustments to steering at higher speeds. The k_d value was increased to have the car turn sooner. For the slow mode, the motor speeds were reduced and the inner wheel speed on turns was increased to take into account the slower turn. The k_p value was increased to align the car at the slower speeds while the k_d value was decreased because the car did not need to turn as soon. For the second mode, values were calibrated in between the two extremes to maximize speed while utilizing the smoother turning.

At higher speeds, the car was limited by its traction. To avoid this, 91% isopropyl alcohol was used to clean the dust and grime off of the wheels after each run. Reapplication was a necessity as the wheels would pick up enough dirt after each run to causing consecutive runs to be less and less reliable.

IV. RESULTS

A. Car Variable Tuning

Through calibration testing, the optimal k_p, k_i, and k_d values were determined, along with the ideal camera angle. The ideal camera angle for all speed modes was measured to be 36 degrees from the horizontal tangent line to the camera. To determine whether or not the values were correct, the car was judged by how tight and smooth it was able to take turns, as well as how straight it could drive through both straight and wavy track pieces. The table below shows the calibration values at different minimum and maximum speeds of the car.

TABLE II
CALIBRATION VALUES

Min Speed (Duty Cycle %)	Max Speed (Duty Cycle %)	k_p	k_i	k_d	Hard Turn Speed % (Inside Wheel)
50	70	0.20	0	0.52	60
60	80	0.15	0	0.65	60
75	85	0.13	0	0.75	25
85	95	0.115	0	0.75	10
90	100	0.10	0	0.79	5

The slight turn percentage for all trials stayed constant at 90% of the current motor speed, but the hard turn percentage varied as the speed of the car changed. The speed of the car caused a greater effect on how hard to cut the sharp turns, which is why that percentage had to be changed. In addition, as the motor speed increased, k_p had to be decreased and k_d had to be increased. The decreasing trend of k_p existed to attempt to minimize the "bounce" of the car on straight tracks, to prevent over steering by telling the car to not turn quite as hard as at lower speeds. The k_d value was then increased as a result of decreasing k_p, to tell the car to turn sooner, since it no longer was turning as hard.

B. Test Runs

To test the consistency of the car, various test courses were created. The first track tested is shown below. It was specifically designed to have a long straightaway go into an intersection and then into a sharp turn, which was expected to be the most difficult stretch of track that could appear during the race.

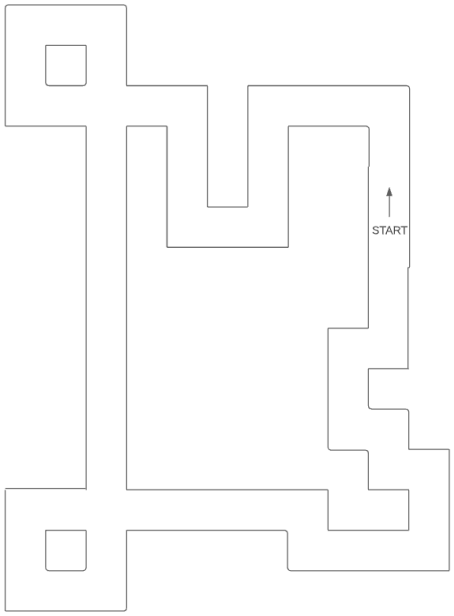

Fig. 5. Test Track 1

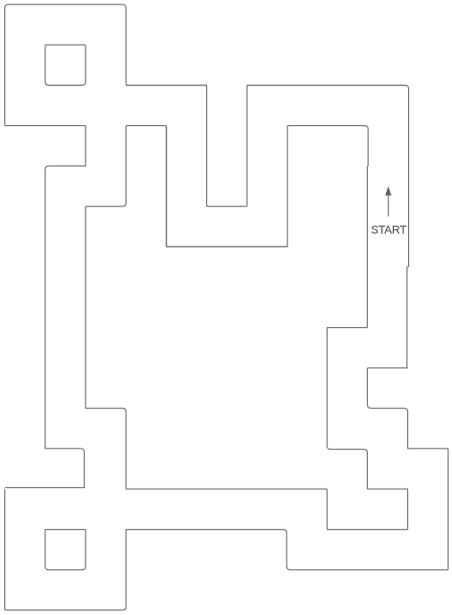

Fig. 6. Test Track 2

In testing, the car completed the test track six times in a row successfully, and in both directions. The table below shows the trial run times, the direction the car was run, and the mode of the car.

TABLE III
TEST TRACK 1 TIMES

Mode	Time (s)	Direction Around Track
slow	17.94	Forward
medium	16.22	Forward
fast	16.53	Forward
slow	16.83	Reverse
medium	15.97	Reverse
fast	15.76	Reverse

The track was then modified to have a long straightaway into a turn, then into an intersection. The new test track is shown below in Fig. 6.

Again, the car successfully completed six consecutive runs, three in each direction. The trial times are shown below in the table.

TABLE IV
TEST TRACK 2 TIMES

Mode	Time (s)	Direction Around Track
slow	18.41	Forward
medium	18.12	Forward
fast	19.20	Forward
slow	18.70	Reverse
medium	17.09	Reverse
fast	17.36	Reverse

C. Race Results

After a single run, the car was able to make it around the track in 21.93 seconds. This was done using the second mode. The second mode was used after a few test runs revealed that the higher speeds caused sloppier turns where time was lost. Out of twelve teams, the car placed first.

D. Bill of Materials

The following table shows the bill of materials and the total cost of the car and all its components.

TABLE V
BILL OF MATERIALS

Item	Quantity	Unit Price	Price	Link
Car Chassis	1	$98.75	$98.75	here
Tenergy 7.2 V Battery Pack (2-pack)	1	$53.99	$53.99	here
NXP Line Scan Camera	1	$55.73	$55.73	here
Motor Kit	1	$43.75	$43.75	here
NXP FRDM K64F	1	$41.18	$41.18	here
RPi Motor Driver Board	1	$28.99	$28.99	here
Total Cost			$322.39	

As shown in Table V, the full cost of the car was $322.39.

E. Possible Improvements

With so many different variables to take into consideration, there is always room for optimization. Scrubbing the tires with isopropyl worked just fine, but it was tedious and might not have been the most effective method for increasing traction. A more efficient method could be looked into. Another improvement could have been to the DC motors. The motors on this car appeared to run slower at maximum duty cycle when compared to other cars. The left motor also did not spin at the same speed as the right when set to the same duty cycle. Replacing these motors early on could have allowed for higher speeds and more accurate testing to be done. The axle on the servo motor could have also been tightened up to reduce the amount of error when setting the servo to a specific value.

V. CONCLUSION

The completed project provided an excellent hands on experience with many different aspects of engineering such as cooperation, problem solving, and prototyping. The project also gave a deeper understanding of the concepts behind interfacing different types of components such as the camera and the motors while also practicing disciplined programming skills. The project was majorly successful as the car designed was able to take home the first place prize in the final race.

REFERENCES

[1] Judge, J John & Prasathong, Suhail & Iqbal Daniyal. "Learning Through Racing: Rochester Institute of Technology's Imagine RIT NXP Car Cup." May 2017.

[2] Beato, Louis. Professor and Lab Instructor. January - May 2021.

[3] Brooks, Xavier & Montano, Eri & Sztuba, Brunon & Henley, Connor. Lab Teaching Assistants. January - May 2021.

[4] Augsburger, Christian & Mountford, Tom. Previous car owners. August - December 2020.

[5] Meyerson, Jacob & Poliwoda, Charlie. "K64 Timers, Interrupts, and Analog-to-Digital Converter" March, 2021.