Speed and Power: Taming the TI-RSLK

Roye Fang

Electrical and Computer Engineering Department, University of California, Los Angeles

EC ENGR 3: Introduction to Electrical Engineering

Dr. Dennis M. Briggs

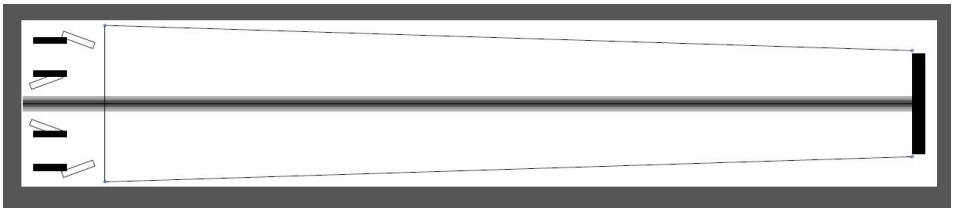
June 13, 2020

Introduction and Background

The Texas Instruments Robotics System Learning Kit (TI-RSLK) is an autonomous robot that can be programmed for logical tasks. In particular, this project utilizes the Texas Instruments MSP432P401R microcontroller and Energia integrated development environment (IDE), allowing the user to program the TI-RSLK to individual needs. This project consisted of programming a TI-RSLK using the Energia IDE to traverse two different paths. The selected paths were a straight track and a ribbon (circular) track (Figure 1 and Figure 2).

Figure 1

Straight Track and Stability Check

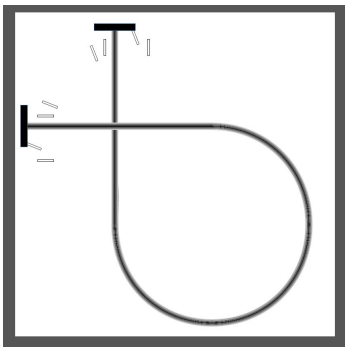


Note. This track is helpful for exposing stability issues in the car.

Credit. Dennis M. Briggs.

Figure 2

Ribbon Track



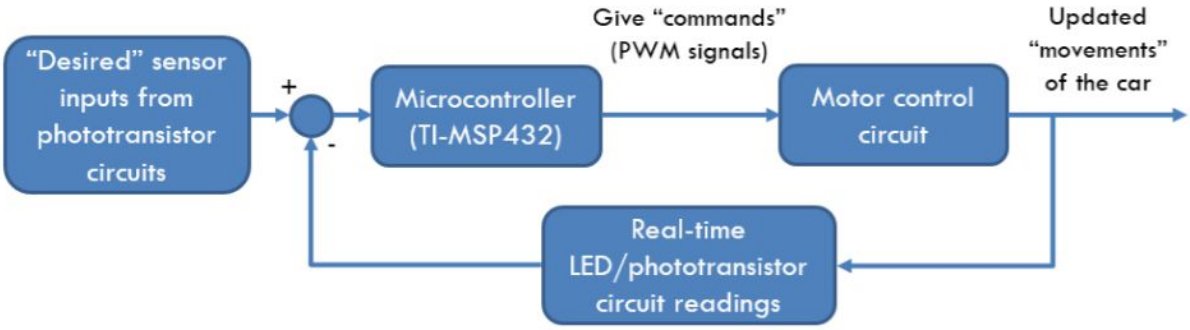
Note. This track is helpful for exposing steering and control issues in the car.

Credit. Dennis M. Briggs.

The general design of the entire system can be understood as a simple loop (Figure 3). First, the phototransistor sensors underneath the car detects its own location. The information is transferred to the MSP432P401R microcontroller, which supplies the car with commands.

Figure 3

Overall System Design



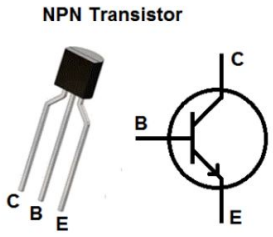
Credit. Dennis M. Briggs.

Phototransistors and Sensors

In order to determine its position on the track, the robot utilizes eight infrared sensors on the bottom of the car. All eight sensors reflect infrared light onto the surface directly beneath itself, returning different values based on location. The sensors are essentially phototransistors; to understand phototransistors, regular transistors must be understood first [8]. Transistors are electronic components that act as electrical amplifiers, making them similar to an electronic switch. In the context of this project, the transistors can be understood as bipolar junction transistors (Figure 4). As the name implies, the transistor can have two different states based on the voltage that is supplied to the transistor.

Figure 4

NPN Transistor Appearance and Schematic

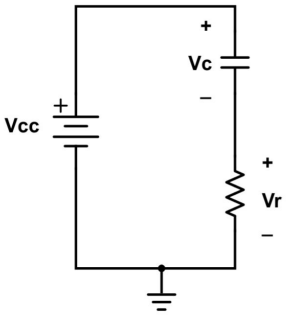


Credit. [What is the Difference Between PNP and NPN?]().

For example, a high voltage could result in current flow, but a low voltage can cut current altogether. Therefore, in the context of the project, the IR sensors on the bottom of the car can be understood as "variable resistors" that can output different readings as the light intensity varies [8]. The car will use this method to determine its position on the track, allowing the user to code a program that can navigate the car. As far as quantitatively reading these outputs, the change in RC time constant allows the user to view the changes in light intensity (Figure 5).

Figure 5

First Order Series Connection RC Circuit



Note. Resistance and capacitance affect the voltage in the circuit.

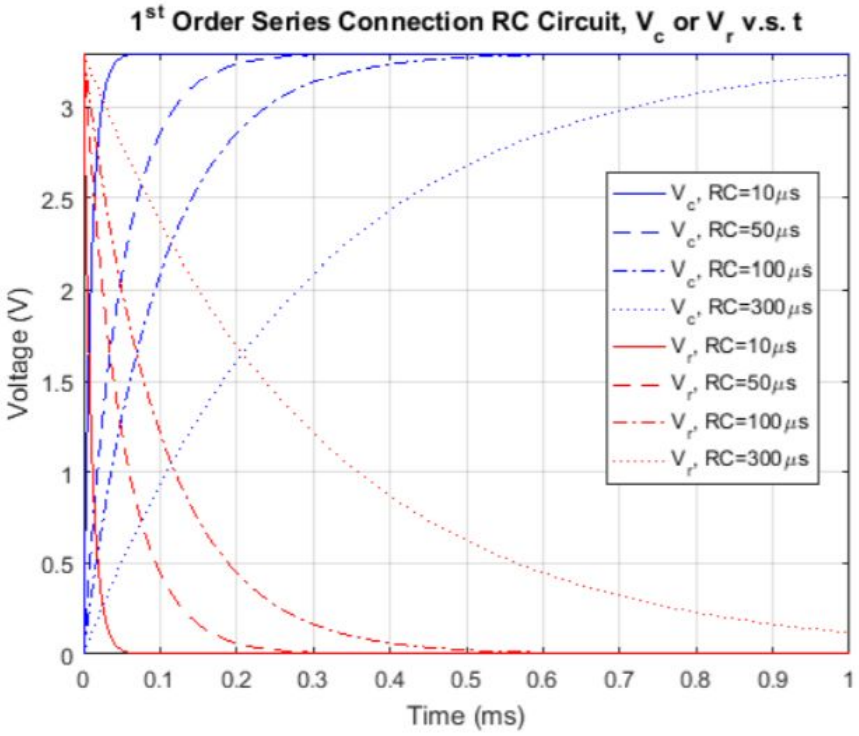
Credit. Dennis M. Briggs.

Based on the RC circuit, the car's voltage will be affected by the resistance and capacitance travelling through the circuit (hence the RC time constant).

Using differential equations and integration, the following formulas describe the relationship between voltage as functions of resistance or capacitance, and the RC time constant (Figure 6).

Figure 6

First Order Series Connection RC Time Constant Graph



Note. The differing times that voltages require to stabilize help define the RC time constant. *Credit*. Dennis M. Briggs.

Therefore, the differing resistances and capacitances defined by the phototransistors on the TI-RSLK require differing amounts of time for the voltage to reach a stable position, known as

the steady state value [8]. These differing settling times, described using the RC time constant, is a way for the car to understand “brightness” or “darkness” based on the IR sensors (Figure 7).

Figure 7

RC Time Constant Equation Based on Varying Phototransistor Resistances

$$V_c(t) = V_{cc}e^{-\frac{t}{C(R_{ph}//R_{OSC})}} \approx V_{cc}e^{-\frac{t}{C \cdot R_{ph}}}$$

Credit. Dennis M. Briggs.

PWM Signals

Once information is collected from the IR sensors, the information needs to be translated and delivered to the wheels. The MSP432P401R can read data using pulse-width modulation (PWM) signals [9]. PWM signals are a method of converting digital inputs into analog inputs (Figure 8). In this project, PWM signals are used to control the wheel motors [8].

Figure 8

Analog versus Digital Signals

Note. Analog signals change gradually, while digital signals are discrete.

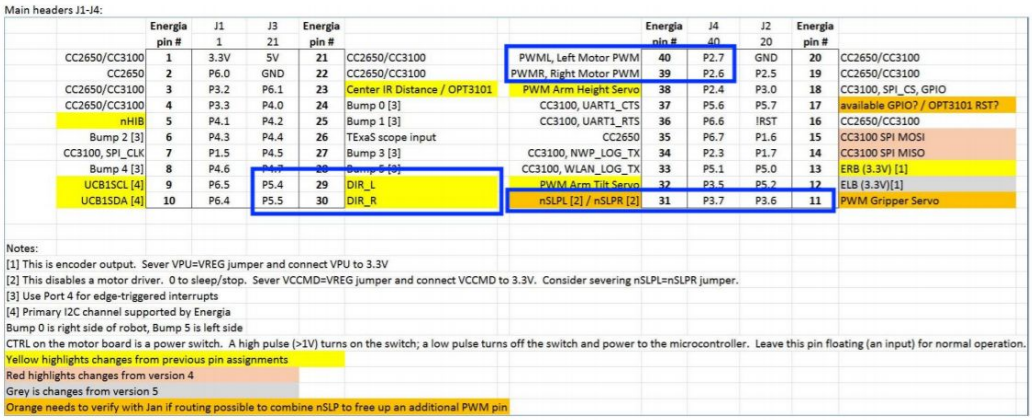
Credit. Digital Communication - Analog to Digital

In reality, the microcontroller can only read discrete values (for example, on or off). However, specific pins on the microcontroller can translate digital values into PWM signals (Figure 9).

Since the user will inevitably require the wheels to travel at different speeds, not just zero or maximum speed, PWM signals can basically translate discrete values into analog values using code (Figure 10).

Figure 9

MSP432P401R Microcontroller Pinout in Reference to Energia



Main headers J1-J4:

	Energia pin #	J1	J3	Energia pin #			Energia pin #	J4	J2	Energia pin #	
		1	21					40	20		
CC2650/CC3100	1	3.3V	5V	21	CC2650/CC3100	PWML, Left Motor PWM	40	P2.7	GND	20	CC2650/CC3100
CC2650	2	P6.0	GND	22	CC2650/CC3100	PWMR, Right Motor PWM	39	P2.6	P2.5	19	CC2650/CC3100
CC2650/CC3100	3	P3.2	P6.1	23	Center IR Distance / OPT3101	PWM Arm Height Servo	38	P2.4	P3.0	18	CC3100, SPI_CS, GPIO
CC2650/CC3100	4	P3.3	P4.0	24	Bump 0 [3]	CC3100, UART1_CTS	37	P5.6	P5.7	17	available GPIO? / OPT3101 RST?
nHIB	5	P4.1	P4.2	25	Bump 1 [3]	CC3100, UART1_RTS	36	P6.6	!RST	16	CC2650/CC3100
Bump 2 [3]	6	P4.3	P4.4	26	TExaS scope input	CC2650	35	P6.7	P1.6	15	CC3100 SPI MOSI
CC3100, SPI_CLK	7	P1.5	P4.5	27	Bump 3 [3]	CC3100, NWP_LOG_TX	34	P2.3	P1.7	14	CC3100 SPI MISO
Bump 4 [3]	8	P4.6	P4.7	28	Bump 5 [3]	CC3100, WLAN_LOG_TX	33	P5.1	P5.0	13	ERB (3.3V) [1]
UCB1SCL [4]	9	P6.5	P5.4	29	DIR_L	PWM Arm Tilt Servo	32	P3.5	P5.2	12	ELB (3.3V)[1]
UCB1SDA [4]	10	P6.4	P5.5	30	DIR_R	nSLPL [2] / nSLPR [2]	31	P3.7	P3.6	11	PWM Gripper Servo

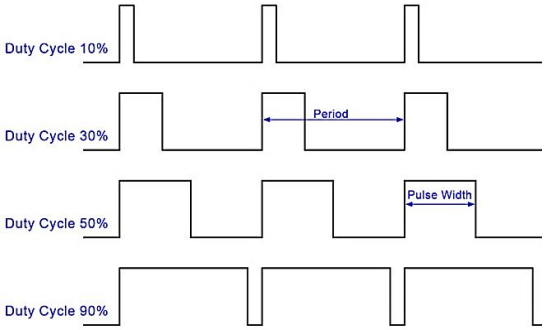
Notes:
[1] This is encoder output. Sever VPU=VREG jumper and connect VPU to 3.3V
[2] This disables a motor driver. 0 to sleep/stop. Sever VCCMD=VREG jumper and connect VCCMD to 3.3V. Consider severing nSLPL=nSLPR jumper.
[3] Use Port 4 for edge-triggered interrupts
[4] Primary I2C channel supported by Energia
Bump 0 is right side of robot, Bump 5 is left side
CTRL on the motor board is a power switch. A high pulse (>1V) turns on the switch; a low pulse turns off the switch and power to the microcontroller. Leave this pin floating (an input) for normal operation.
Yellow highlights changes from previous pin assignments
Red highlights changes from version 4
Grey is changes from version 5
Orange needs to verify with Jan if routing possible to combine nSLP to free up an additional PWM pin

Note. Energia recognizes pins 39 and 40 as the PWM pins that control the left and right wheels. *Credit.* Dennis M. Briggs. and Texas Instruments.

Having different speeds is vital for steering (reducing the speed of one wheel and increasing the speed of the other) and even personal preference. Therefore, the microcontroller can achieve analog inputs by using different duty cycles to simulate analog values through arithmetic averages [7].

Figure 10

Various Duty Cycles in PWM signals



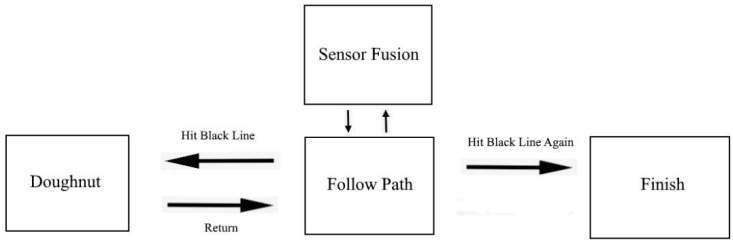
Credit. *Control of Electrical Appliances through Voice Commands*

Energia and Code Design

Energia is an Arduino based programming environment that can be used for the TI-RSLK; it uses a C++ based coding language [2]. Using Energia, the user writes code that can be uploaded onto the MSP432P401R. The microcontroller then operates the car. The overall design of the code is a simple loop that is analogous to the overall design of the entire project (Figure 3 and Figure 11).

Figure 11

Overall Code Design



Created Using. paint.net

Using a counter system, the program can determine what state it is in (follow the path, do a doughnut, or finish) and instruct the car on how to operate at any point in the trial. The counter begins in its path following state. In this state, the code begins by using sensor fusion to determine its position on the track. Using this fused value, the program will use a PID controller to further refine the car's fused value to ensure stability during movement. Using the output of the PID controller, the car will follow then drive along the path. When the program detects the bold black at the opposite end of the track, it will increment the counter and engage the doughnutting state. Once the doughnut is executed, the counter will increment and return the car to its path following state. Once it returns to the start line (on the ribbon track, since there is no start line on the straight track), it will increment the counter and perform another doughnut, completing the trial (Figure 12).

Figure 12

Code for Main Program

```
void loop()
{
    sensorFusion();

    //counter = 1: away from start, counter = 3: return to start
    if(counter == 1 || counter == 3)
    {
        followPath();
    }

    //counter = 2: do a doughnut
    if(counter == 2)
    {
        doughnut();
    }

    //finish
    if(counter == 4)
    {
        finish();
    }
}
```

Created Using. Energia.

Sensor Fusion. The program begins by conducting sensor fusion. To ensure a smooth and accurate performance, sensor fusion is conducted to balance and combine the eight values in a single, meaningful fused value that the user can understand to steer the car [3]. Prior to coding the car, the user should conduct sensor fusion inside a spreadsheet application to visualize all possible scenarios for the car on the track. The sensor fusion part of the code will provide a single value that determines the car's current position on the track.

PID Control. Afterwards, the program uses a proportional-integral-derivative controller (PID controller) to assist the car in consistently and accurately navigating the track. A PID controller is a versatile algorithm that provides a method of controlling a system [6]; as the name implies, the PID can be understood as an equation that sums the proportional, integral, and derivative terms alongside arbitrary constants (Figure 13).

Figure 13

PID Controller Equation

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}$$

Note. This equation variation uses the same error term for all three constants.

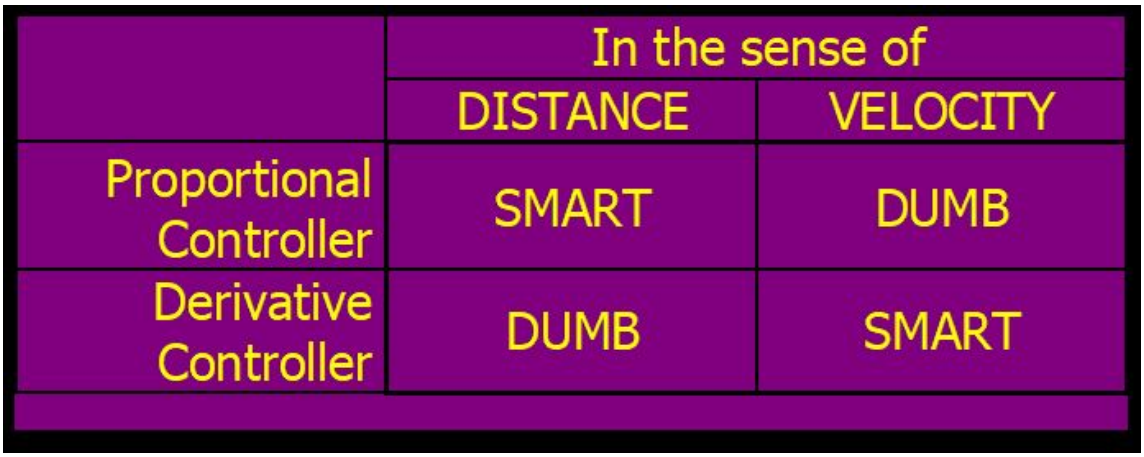
Credit. How To Implement A ControlLogix PID Controller.

These constants (K_p, K_i, K_d) are unique to each system and are entirely up to the user to adjust. However, this system only utilizes proportional and derivative control, omitting integral control because it is less important in a constantly variable system. The integral controller affects

overall control using error over a period of time [5]. Therefore, it is inherently futile in a dynamic system like this project, since the car is constantly moving. For example, when the car suddenly turns onto an extreme turn on the track, the integral component could make unnecessary corrections to the car's steering and cause problems.

Figure 14

Comparing Proportional and Derivative Control



Credit. Dennis M. Briggs.

Therefore, combining proportional and derivative control should result in a relatively stable car (Figure 14). Using PID control, the program scales the output of the PID equation down and into a value that can steer the car.

The overall project goal is developing a stable, yet relatively fast program for the TI-RSLK that is applicable to both the straight and ribbon tracks.

Testing Methodology: Sensor Fusion

The first step of testing the TI-RSLK is sensor fusion. Based on the aforementioned description of sensor fusion, it consisted of taking eight sensor readings from the bottom of the robot and combining them into a single value.

Test Setup

The physical characteristics of testing and collecting sensor fusion data are defined by an accurate and relevant setup. Holistically, the sensor fusion data collection design should be nearly identical to all possible positions a car can be on the track (Figure 15). It is vital that the user mediates the amount of sunlight in the testing conditions; extreme brightness or darkness can skew the results of sensor fusion, especially due to the subtle nature of the sensors. Furthermore, since the track will probably be placed on the floor, the user should also collect sensor fusion data on the floor.

Figure 15

TI-RSLK on Track

Uploaded On. Youtube.

The electrical characteristics of sensor fusion is defined by the nature of the IR sensors. Each sensor utilizes infrared light in order to output a value. The sensors are read as an array of

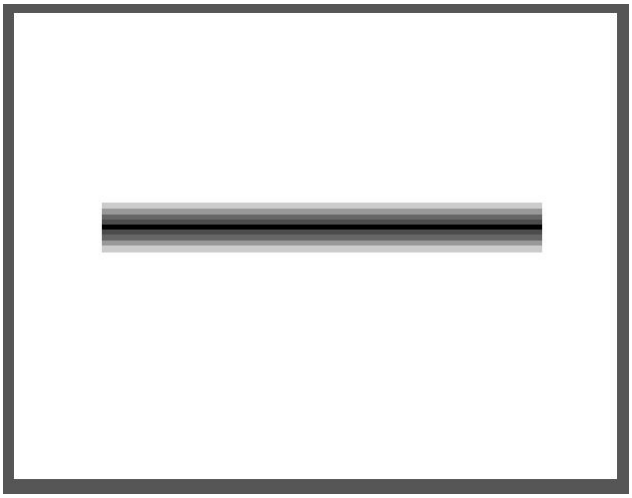
eight values on Energia [8]. As the aforementioned description of phototransistors implies, the different values are based on the surface directly underneath the car. Therefore, an efficient way to accumulate all possible sensor readings is slowly and gradually sweeping the car across the track.

How Tests Were Conducted

The procedure of sensor fusion can be summarized in three steps. First, obtain the code inside the **`ECE3.zip`** file and upload the sensor fusion code onto the TI-RSLK. Second, as delicate as can be, gradually slide the car across a single sheet of the track (Figure 16). Taping the sheet of paper with scotch tape can ensure the paper will not move as the car slides across.

Figure 16

Short Straight



Note. The short straight is useful for conducting sensor fusion.

Credit. Dennis M. Briggs.

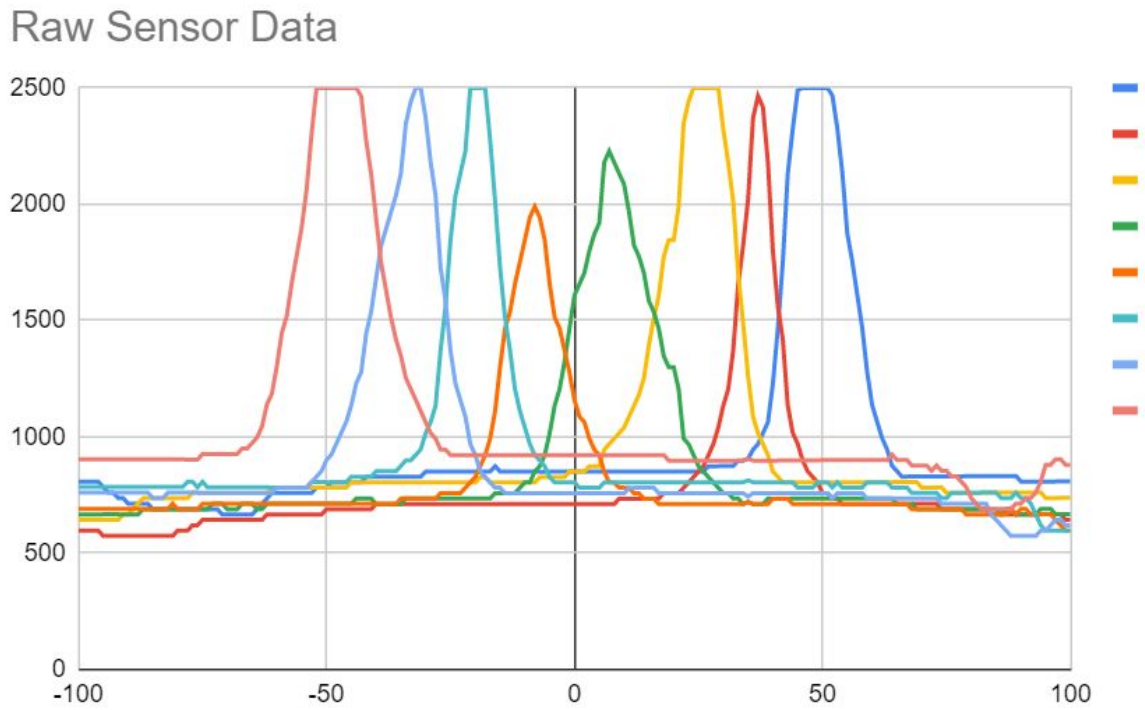
Finally, once the car has covered the entire range of the car, copy and paste the values from the serial monitor of Energia into a spreadsheet application. This specific project utilizes Google Sheets. After transferring the Serial monitor values, the process of sensor fusion begins.

Data Analysis

Once the values from the serial monitor are pasted into a spreadsheet application, the process of sensor fusion begins (Figure 17). In order to analyze the aforementioned sensor data, sensor fusion is completed in three steps.

Figure 17

Raw Sensor Data



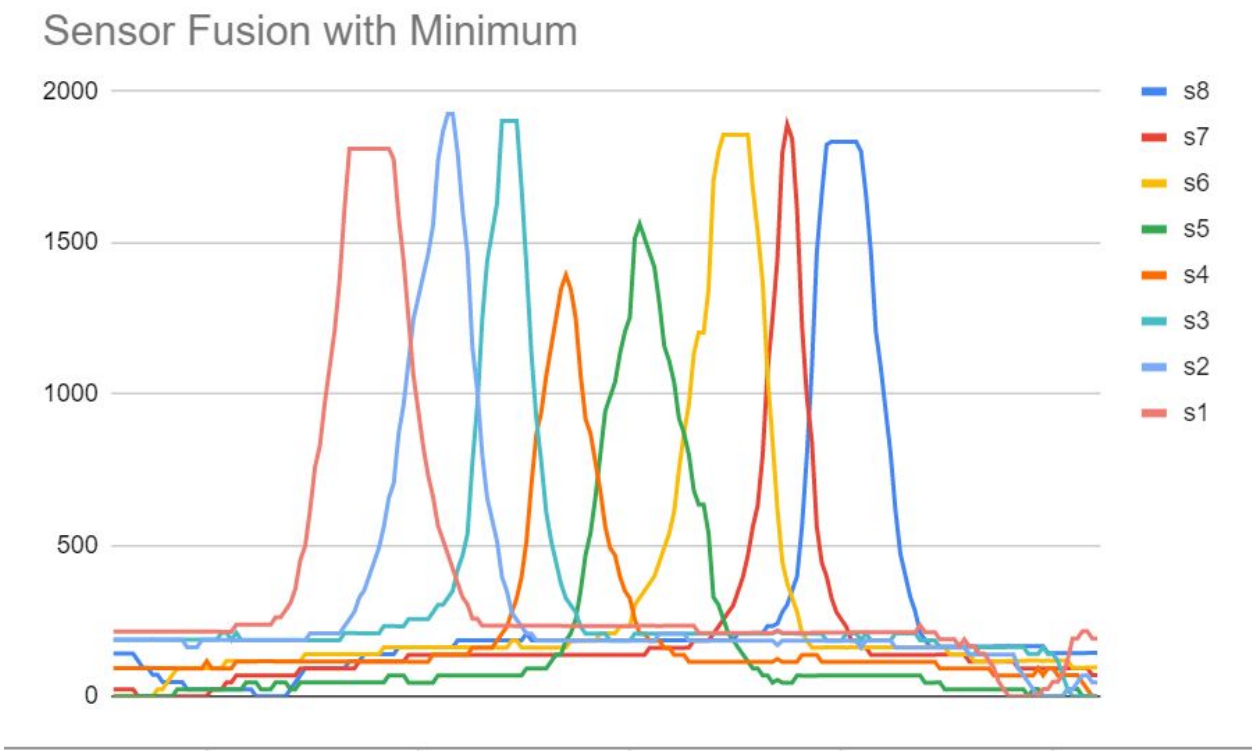
Note. Note how all minimum values are around 500-1000.

Created Using. Google Sheets.

First, subtract the minimum values from each sensor (Figure 18).

Figure 18

Sensor Data After Subtracting Minimum



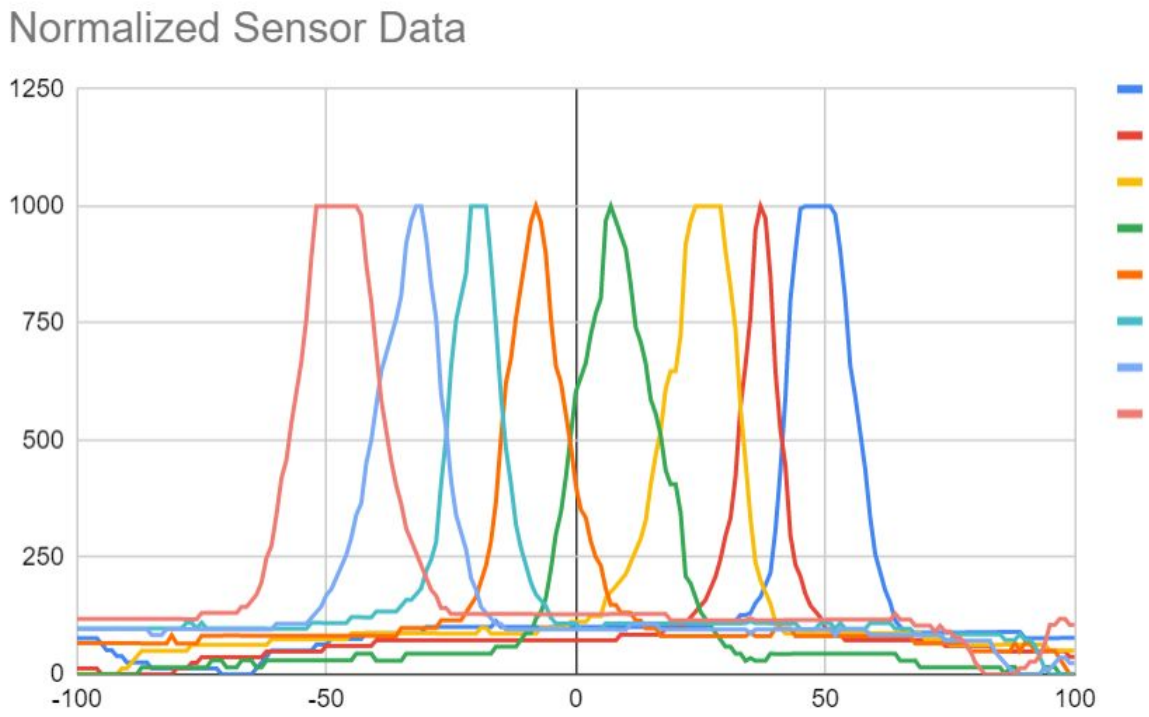
Note. Now all the minimum values are zero.

Created Using. Google Sheets.

Second, normalize each sensor to have the same maximum value. This specific project normalized each sensor to have a maximum of 1000 and minimum of 0 (Figure 19).

Figure 19

Normalized Sensor Data



Note. All values now vary between 0 and 1000.

Created Using. Google Sheets.

Finally, use a sequence of numbers to weigh each individual sensor. The goal of this step is to achieve a single value for each position on the track. The sequence of numbers should weigh the outer sensors greater than the inner sensors. The sequences [-8 -4 -2 -1 +1 +2 +4 +8] and [-15 -14 -12 -8 +8 +12 +14 +15] were provided to the user as sample guidelines. This specific project uses [-8 -4 -2 -1 +1 +2 +4 +8], but the sequence of numbers should be adjusted

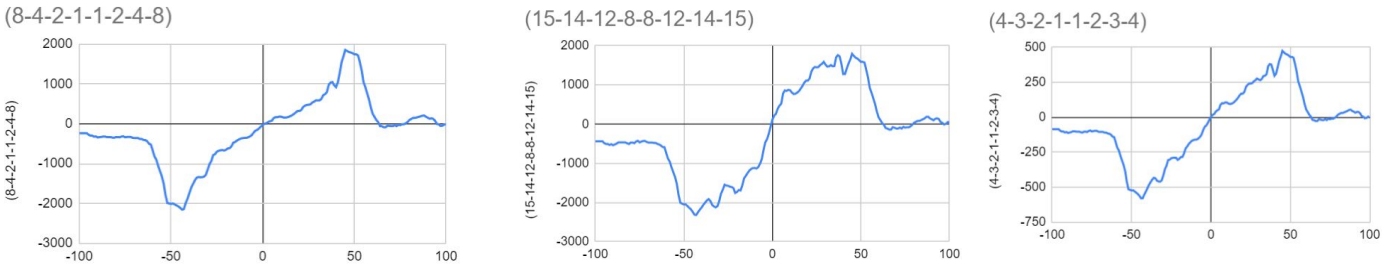
accordingly to each individual TI-RSLK. Ideally, the sequence of numbers should result in an exponentially increasing and decreasing and/or cubic model when graphed.

Test Data Interpretation

Based on the final graph, the [-8 -4 -2 -1 +1 +2 +4 +8] sequence appears to provide the best exponential relationship; the graph should also resemble a cubic function. Although it was advised to attempt to follow a linear model, this model is preferred because straying away from the line (a high or low fused displacement value) should cause more extreme action than a scenario where the car is nearly straight. As a result, the car is less likely to veer off the track, but will still remain stable when it is directly on top of the center of the black line. Other sequences of numbers resulted in an unfavourable graphical model for the car.

Figure 20, 21 and 22

Different Sequences of Numbers to Weight Sensors



Note. Figure 20 resembles a desirable cubic function.

Created Using. Google Sheets.

Whilst the sequence of [-4 -3 -2 -1 +1 +2 +3 +4] was initially chosen for its linearity, the user found the car to be unreliable at extreme positions on the track (Figure 22). For example, when the car starts on the parallel and offset position on either track, the car would fail to latch on the track altogether from the start. In contrast, the [-15 -14 -12 -8 +8 +12 +14 +15] sequence resulted in the car being able to latch onto the track at extreme offsets, but an unstable and wiggly system when it is nearly straight. Although the [-15 -14 -12 -8 +8 +12 +14 +15] provides a cubic-like function, the increase/decrease is front-loaded and drops off in the higher values (Figure 21). The subsequent program will cause the car to be unstable when it is nearly on top of the line, and not have powerful enough steering when it begins to veer off course to fix itself. As a result, the car would never truly travel in a straight line and would violate the tapering lines on the straight track.

Therefore, the [-8 -4 -2 -1 +1 +2 +4 +8] sequence is chosen for its balance in steering and smoothness (Figure 20). As a side note, the outer values that resemble a horizontal line on the sensor fusion graphs should be ignored for this specific project. Although this can be useful for visualizing an extreme emergency path recovery system, this project did not incorporate it.

Testing Methodology: Track Tests

Track testing involves coding in the Energia IDE and repeated trials on the actual track. This includes the process of debugging code and calibrating each constant in the code.

Test Setup

The physical characteristics of track tests are broadly similar to the process of sensor fusion. Since sensor fusion is supposed to simulate the conditions of the race track, the track tests should, likewise, account for the initial conditions of sensor fusion. This includes the lighting of the room and location where the track is placed (probably the floor).

The electrical characteristics of track tests are rooted in Energia. Having prior experience with C++ or Arduino is largely beneficial to this project. A key function for debugging code during the testing process is the `Serial.print()` function [1]. This function allows any variable or value to be printed for the user to see, allowing anyone to quickly detect a logic error.

How Tests Were Conducted

There is no specific procedure for this portion of testing. However, there are a broad set of checkpoints that the user should fulfill throughout the process. By the end, the user should have a set of K_p, K_d, and various other constant values.

First, the user should calibrate the nature of the wheels for a straight line. Every car may have individual blemishes that the user should try and account for in their code. For example, this project features a TI-RSLK with a slightly damaged right wheel. Therefore, the user had to

increase the speed of the right wheel when implementing the default straight speed in order to maintain a straight line if the car were to theoretically travel in a straight line (Table 1-3).

Table 1

Low Speed Tests

Left Wheel	50	50	50	50	50
Right Wheel	50	51	52	53	54

Table 2

Medium Speed Tests

Left Wheel	100	100	100	100	100
Right Wheel	100	101	102	103	104

Table 3

High Speed Tests

Left Wheel	150	150	150	150	150
Right Wheel	150	151	152	153	154

After accounting for various individual imperfections for the car, the user should determine the various constant values inside the code. While the constants might vary depending on the design of the code, it will probably contain a default speed constants, the maximum steering constants, and the K_p and K_d constants in the PID controller equation. These constants are completely arbitrary and will vary across each system based on the TI-RSLK and personal preferences (Table 4).

Table 4

Various K_p and K_d Values

1	2	3	6	8	10	15	16	20	24	27	30	33	36	40

Another constant to account for is doughnut times. In this project, the donut is executed by reversing one wheel, and using `delay()` to set a constant donut duration. Since this project aimed for a high speed, yet consistent car, a maximum speed of 255 was selected in order to reduce the number of factors affecting the doughnut [1]. Since both the straight and ribbon track demands a 180° doughnut, a constant donut duration could be selected for both tracks. Various doughnut durations were tested to determine the most accurate time interval (Table 5).

Table 5

Different Doughnut Durations

Duration	100	150	200	250	300	500	1000

Data Analysis

Data analysis was conducted through continuous testing. There is no specific pattern or path; when taking into account the subtle nature of the TI-RSLK and the project conditions, the user determined the correct combination of all constants through brute force testing. Every combination was tested and the final choices in the code were tailored to personal preference and/or the best performance. All data is processed into desirable or undesirable outcomes.

For the default speeds, since the right wheel is visually faulty, all 5 speeds for the right wheel were tested with a constant left wheel speed to find how the car would behave (Table 6-8).

Table 6

Low Speed Test Results

Left Wheel	50	50	50	50	50	50
Right Wheel	50	51	52	53	54	55
Result	Slight Right	Slight Right	**Straight**	Slight Left	Slight Left	Slight Left

Table 7

Medium Speed Test Results

Left Wheel	100	100	100	100	100	100
Right Wheel	100	101	102	103	104	105
Result	Slight Right	Slight Right	Slight Right	**Straight**	Slight Left	Slight Left

Table 8

High Speed Test Results

Left Wheel	150	150	150	150	150	150
Right Wheel	150	151	152	153	154	155
Result	Slight Right	Slight Right	Slight Right	Slight Right	Slight Right	**Straight**

In order to process the vast quantity of possible K_p and K_d values, combinations of K_p and K_d values are visualized on a grid to test every possible combination of K_p and K_d values (Table 9).

Table 9

Different K_p and K_d Pairs

	1	**2**	**3**	**6**	**8**	**10**	**15**	**16**	**20**	**24**	**27**	**30**	**33**	**36**	**40**
1	1/1	1/2	1/3	1/6	1/8	1/10	1/15	1/16	1/20	1/24	1/27	1/30	1/33	1/36	1/40
2	2/1	2/2	2/3	2/6	2/8	2/10	2/15	2/16	2/20	2/24	2/27	2/30	2/33	2/36	2/40
3	3/1	3/2	3/3	3/6	3/8	3/10	3/15	3/16	3/20	3/24	3/27	3/30	3/33	3/36	3/40
6	6/1	6/2	6/3	6/6	6/8	6/10	6/15	6/16	6/20	6/24	6/27	6/30	6/33	6/36	6/40
8	8/1	8/2	8/3	8/6	8/8	8/10	8/15	8/16	8/20	8/24	8/27	8/30	8/33	8/36	8/40
10	10/1	10/2	10/3	10/6	10/8	10/10	10/15	10/16	10/20	10/24	10/27	10/30	10/33	10/36	10/40
15	15/1	15/2	15/3	15/6	15/8	15/10	15/15	15/16	15/20	15/24	15/27	15/30	15/33	15/36	15/40
16	16/1	16/2	16/3	16/6	16/8	16/10	16/15	16/16	16/20	16/24	16/27	16/30	16/33	16/36	16/40
20	20/1	20/2	20/3	20/6	20/8	20/10	20/15	20/16	20/20	20/24	20/27	20/30	20/33	20/36	20/40
24	24/1	24/2	24/3	24/6	24/8	24/10	24/15	24/16	24/20	24/24	24/27	24/30	24/33	24/36	24/40
27	27/1	27/2	27/3	27/6	27/8	27/10	27/15	27/16	27/20	27/24	27/27	27/30	27/33	27/36	27/40
30	30/1	30/2	30/3	30/6	30/8	30/10	30/15	30/16	30/20	30/24	30/27	30/30	30/33	30/36	30/40
33	33/1	33/2	33/3	33/6	33/8	33/10	33/15	33/16	33/20	33/24	33/27	33/30	33/33	33/36	33/40
36	36/1	36/2	36/3	36/6	36/8	36/10	36/15	36/16	36/20	36/24	36/27	36/30	36/33	36/36	36/40
40	40/1	40/2	40/3	40/6	40/8	40/10	40/15	40/16	40/20	40/24	40/27	40/30	40/33	40/36	40/40

Finally, each doughnut duration was tested with a constant speed of 255. The data is processed for both the straight and ribbon track (Table 10).

Table 10

Different Doughnut Duration Results

Duration	100	150	200	250	300	500	1000
Result	Undershoot	Undershoot	**Ideal**	Overshoot	Overshoot	Overshoot	Overshoot

Test Data Interpretation

Overall, the processed test data can be interpreted and applied fairly easily. Interpreting the track testing data equivalent to actually applying the results in the code. For track tests, the process of applying the data in the program is largely observational.

When determining the default speed the car should be when it is theoretically perfectly on the path and travelling in a straight line, the user should pick speeds that result in a straight travel path. The user can test the correct default speed using a separate program in Energia to have the car travel at a constant speed in `void loop()`.The final selected speeds for the left and right wheels are 50 and 52 for a slow speed, 100 and 103 for a medium speed, and 150 and 155 for a high speed, respectively. Although Energia's `analogWrite()` can accept a maximum speed of 255, the maximum default speed needs a ceiling to ensure enough space for steering the car [1]. For example, a maximum default speed of 150 and 155 enables a space of 100 to be dedicated for the sole purpose of steering the car. Taking the risk of reducing steering might be acceptable for the straight track, but the sharp turns on the ribbon track enable the car to be susceptible for going off the track. Since the program is aimed to be as universally applicable as possible, a safer speed was chosen in consideration for the ribbon track. Since the project aims for a high speed, the 150 and 155 pair was chosen for the final program, but the slower speeds are useful steps for incrementally building consistency throughout your code.

The K_p and K_d constants were also determined through numerous tests. Although the grid might seem daunting at first, the user can quickly eliminate many pairs of values. In this particular project, the user very quickly determined that the K_d value must be significantly greater than the K_p value. For example, a pair like [K_p = 12/K_d = 1] or [K_p = 40/K_d = 16] led to

an unstable car; the user quickly disregarded the pairs with a high K_p ratio. As a result, about one-third of the possible combinations are already eliminated. After filtering through the remaining data, the user concluded that only a high K_d ratio can lead to a mostly stable car (such as [$K_p = 1/K_d = 9$], [$K_p = 4/K_d = 40$], or [$K_p = 3/K_d = 30$]. Eventually, the user settled on [$K_p = 2/K_d = 36$] as the values that fit this specific TI-RSLK.

In general, the need for a higher K_d value can be understood by observing the inherent nature of this project. In this highly dynamic program, the car is constantly changing location and the program constantly needs to determine the current and, to a limited extent, future choice of steering [4]. By including a higher K_d value, the car will make more extreme decisions when in a high intensity situation (banking the curve of the ribbon track), but remain smooth and calm in a low intensity situation (simply travelling down a straight portion of the track). As long as there is some proportionality being factored in, the car should have a broad ability to steer itself and remain on the track. The relationship between K_p and K_d in this specific context can be summarized with a simple understanding of their subsequent focus. The K_p constant provides a broad framework for the car to steer, while the K_d constant fine-tunes the car to make choices based on its location on the track. It is important to note that these values should and likely will vary across each TI-RSLK (and every PID controlled based system in general) due to many external factors in the testing environment and internal factors in the system itself [6].

Finally, doughnut durations were chosen based on how consistently they could execute a consistent 180° doughnut with minimal error. The `delay()` duration was chosen based on consistency and accuracy. The user determined that a delay of 200 ms provided the most accurate donut for this system. The `delay()` function has limitations that should be accounted

for if it is to be used inside a program. When the `delay()` function is called, all other processes are stopped and frozen in their current state [1]. Whilst this limitation is irrelevant in the context of this specific project, it might affect the functionality of a different project, if the user needs to perform a different action when executing a doughnut. A safe alternative uses the `millis()` function to replicate the effects of a delay for a possible future project [1].

Results and Discussion

Test Discussion

The sensor fusion and track test data are relevant and were collected and analyzed in a relatively efficient and useful manner. In general, the largest problem was balancing a proper steering system that would ensure the car never left the track and could steer even in extreme conditions, but remained stable once it was locked onto the line. However, this problem was largely remedied after the user analyzed the sensor fusion and test track data. Overall, combining the ideas behind sensor fusion and a large K_d constant resulted in a universally stable driving and steering system. As the testing methodology mentioned, the choice of number sequence for the sake of fusing [-8 -4 -2 -1 +1 +2 +4 +8] and the idea of weighing K_d more heavily results in the car making more extreme steering choices in extreme situations while staying stable once the car is locked on the line for a smooth straight travel. These two observations and choices in the sensor fusion and test tracks for the final program are greatly applicable to the project goal of developing a universal program for both tracks. Since the straight track tends to expose the stability of the car, while the ribbon track exposes the steering system of the car, having a program that can satisfy both tracks with a relatively high speed completely satisfies the project and the user's personal goals.

Race Day Discussion

The car was a success during Race Day. It completed all four requested trials (two on the straight track and two on the ribbon track) with no problems. The car completed all four trials with relatively quick speed, compared to other contenders in the same lab section. Links for videos of the trials are included in the appendix.

A limitation of the code is using a constant time when performing a donut on the finish/start line. In this program, the user has the robot donut for a set amount of time before reentering the `void followPath()` section of the code using the delay() function in the Energia library. If this car were tested on a different track, the car might go off the track after doing a donut, since the robot is designed for this specific track. A universally applicable program would ideally perform a donut, and wait until the black line is detected before reengaging `void followPath(),` regardless of time. Furthermore, the code does not contain an extreme path recovery. Instead, it relies on proportional and derivative control to never lose the path nor have a need for path recovery.

The car itself is relatively reliable and consistent. Users should note its battery levels while programming and completing the project. Spending many hours on the car can drain the batteries quickly, leading to malfunction and incorrect execution of the program. Due to the circumstances of COVID-19 and online learning, having to print out individual sheets of paper for the track can be inconvenient and wasteful. The track is flimsy and required significant lengths of scotch tape to fix the track in place for the car to navigate.

In future projects, the user should be wary of the car while spending time on it. Simple mistakes, such as uploading code while the chassis board is on and unknowingly letting the car run off a desk, can easily happen when spending long hours on code. In addition, taking precautions, such as a safe testing environment where the car can't break anything or be broken and a contingency plan so the car does not accidentally damage itself while stalling if it veers off course and hits an object. A longer USB could allow the user to utilize `Serial.print()` for debugging purposes (printing fused values, error values, etc.) and finding logic errors in real time

while the robot is performing a trial. Finally, taking an incremental approach and developing the program gradually allowed for an overall smooth learning experience. Given the amount of time it took to finalize the code (determining exact constants in the PID controller formula, speeds, and logic control), it would have been difficult and pointless if the user were to procrastinate and save it for the last minute.

Conclusions and Future Work

The project successfully fulfilled the project goal of understanding and programming the TI-RSLK to navigate and complete the desired trials for the straight and ribbon track. The program successfully incorporates sensor fusion and PID controllers to control the robot at a relatively high speed. Furthermore, it gave the chance to find tune and tamper with subtle details that may appear minor, but are, in fact, integral to the holistic performance of the car. This is something you can only achieve with hands-on engineering; watching a video or conventional homework assignments cannot provide anything comparable to a project like this. One possible extension could include a system that maintains a higher speed for a straight line portion of a track, but enables the car to slow down greatly when traversing through curves and extreme bends. This could allow the robot to navigate more complex and difficult tracks. In addition, the car could be programmed with different combinations of proportional, integral, and derivative control to determine the effects of each on a dynamic system.

Overall, the project successfully provided an opportunity to learn about the vital relationship between electrical and computer engineering. However, more importantly, it delivered the joy of “engineering” as a noun and as a verb. As a verb, engineering encompasses the process of persistence and patience, while engineering as a noun encapsulates the principles of brainstorming and completion. Together, they define engineering and what it means to be an engineer.

References

[1] Arduino Home. (n.d.). Retrieved June 13, 2020, from https://www.arduino.cc/

[2] Energia Home. (n.d.). Retrieved June 13, 2020, from https://energia.nu/

[3] Lee, J. (2016, July 19). How Sensor Fusion Works . Retrieved June 13, 2020, from https://www.allaboutcircuits.com/technical-articles/how-sensor-fusion-works/

[4] Messner, B., & Tilbury, D. (2017). Introduction: PID Controller Design. Retrieved June 8, 2020, from http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction§ion

[5] PID for Dummies. (n.d.). Retrieved June 13, 2020, from https://www.csimn.com/CSI_pages/PIDforDummies.html

[6] PID Theory Explained. (2020, March 17). Retrieved June 13, 2020, from https://www.ni.com/en-us/innovations/white-papers/06/pid-theory-explained.html

[7] Pulse Width Modulation Used for Motor Control. (2018, February 16). Retrieved June 13, 2020, from https://www.electronics-tutorials.ws/blog/pulse-width-modulation.html

[8] Stafsudd, O.M., EE3 Introduction to Electrical Engineering Laboratory Manual, as updated and appended.

[9] What is a Pulse Width Modulation (PWM) Signal and What is it Used For? (n.d.). Retrieved June 13, 2020, from https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019OkFSAU&l=en-US

Appendix

Energia Code

Full Energia code is uploaded in a separate .ino file.

Race Day Videos

https://www.youtube.com/watch?v=MsBcXVv27oM

https://www.youtube.com/watch?v=n18pafm9niw

https://www.youtube.com/watch?v=9Y4T0mqMIXw

https://www.youtube.com/watch?v=XFGRQ-46tLM

Raw Sensor Data

https://docs.google.com/spreadsheets/d/19ifSkZLI07H5ysvPeJpG4Ciff2U-aEjSRHl053JaVRM/edit?usp=sharing