Leeksun Cho
Nhat Ho
Professor Briggs
EC Engr 3
15 December 2020

ECE 3 20F Final Report

1. Introduction and Background:

The goal of this project was to program a small electronic car, using the TI-RSLK (Robotics System Learning Kit) and TI MSP432 microcontroller, that followed a black track on the floor using sensors and error corrections. The track consisted of a black line with gradient edges, which overall represented the path the car must follow. The main struggle was to make the car utilize its light sensors and proper coding to follow the line as smoothly and as fast as possible. The car also needed to turn 180 degrees at the end of the track to turn and maneuver itself back to the beginning.

For our path sensing system, three main processes must be concerned: Path sensing circuitry, calibration process, and sensor fusion process. To understand the path sensing system, we need to cover the car's design first. It was controlled by the MSP432 microcontroller, mounted on top of the TI-RSLK, sits on the top of the car. The microcontroller's pin map is shown below in Figure 1.

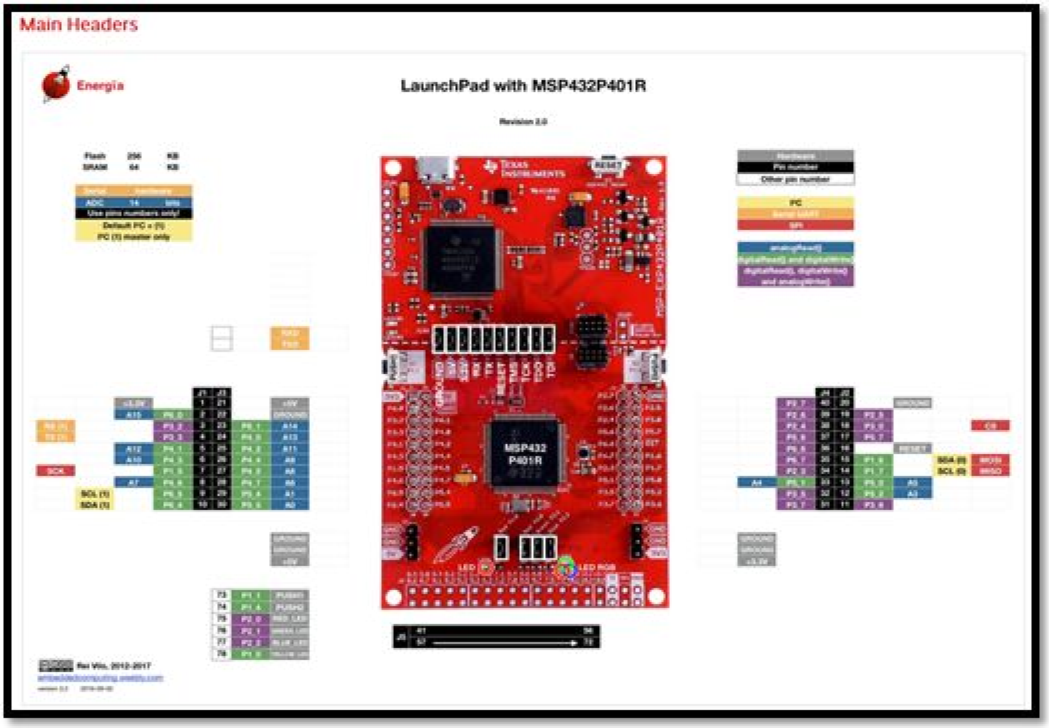
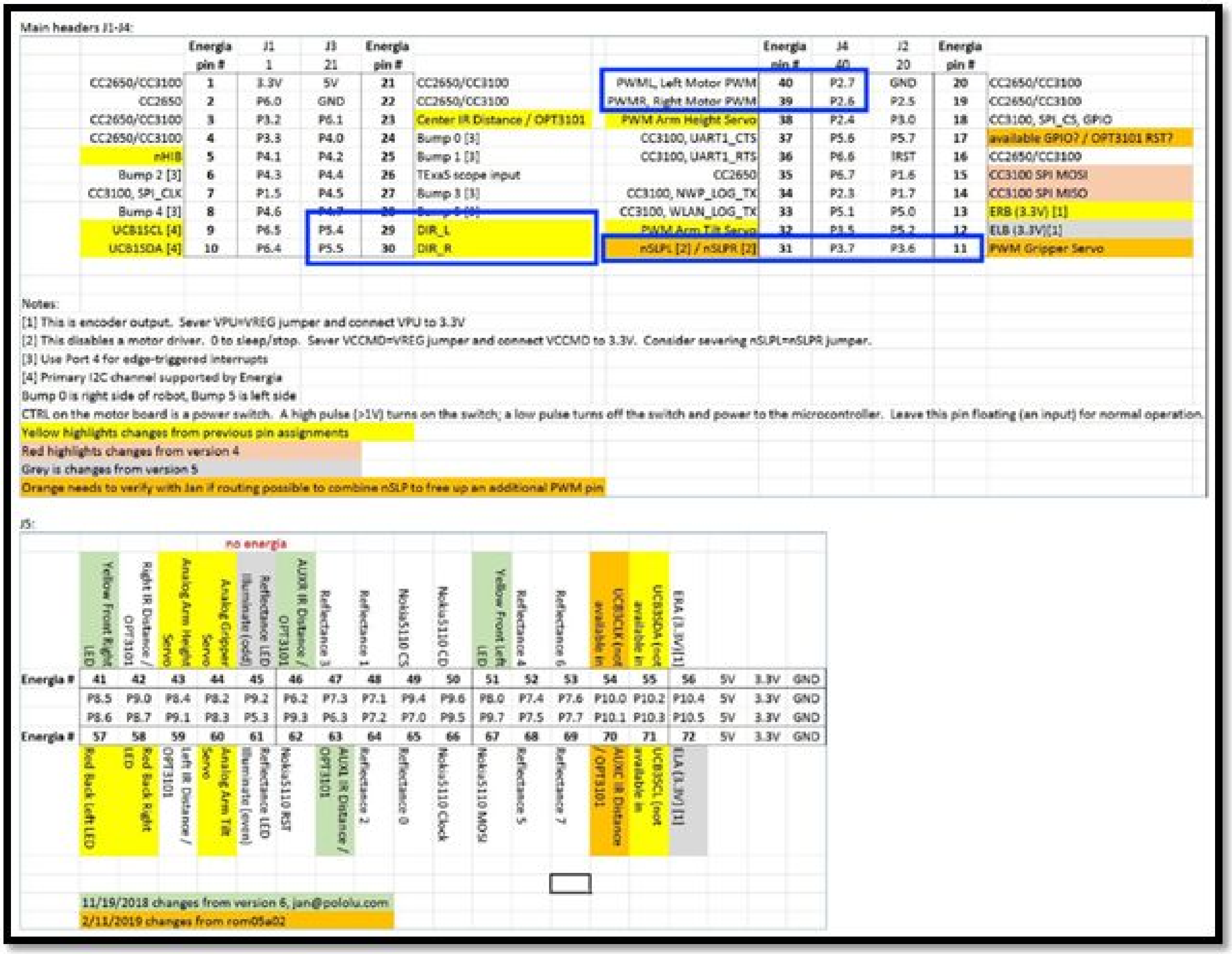


Figure 1: MSP432 pin map (source: https://energia.nu/pinmaps/msp-exp432p401r/)

Firstly, we use the Energia IDE, a stream of Arduino using for Launchpads from Texas Instruments [1], to program the microcontroller to set up the necessary pin to make the car move, such as the motor's sleep SLP, pulse width modulation PWM, and direction DIR pin. We can check these pins in the microcontroller pin map in Figure 2 below.



Main headers J1-J4:

	Energia	J1	J3	Energia			Energia	J4	J2	Energia	
	pin #	1	21	pin #			pin #	40	20	pin #	
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

J5:

					no energia														
	Yellow Front Right LED	Right IR Distance / OPT3101	Analog Arm Height Servo	Analog Gripper Servo	Reflectance LED Illuminate (odd)	AUXR IR Distance / OPT3101	Reflectance 3	Reflectance 1	Nokia5110 CS	Nokia5110 CD	Yellow Front Left LED	Reflectance 4	Reflectance 6	UCB3CLK (not available in	UCB3SDA (not available in	ERA (3.3V)[1]			
Energia #	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	5V	3.3V	GND
	P8.5	P9.0	P8.4	P8.2	P9.2	P6.2	P7.3	P7.1	P9.4	P9.6	P8.0	P7.4	P7.6	P10.0	P10.2	P10.4	5V	3.3V	GND
	P8.6	P8.7	P9.1	P8.3	P5.3	P9.3	P6.3	P7.2	P7.0	P9.5	P9.7	P7.5	P7.7	P10.1	P10.3	P10.5	5V	3.3V	GND
Energia #	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	5V	3.3V	GND
	Red Back Left LED	Red Back Right LED	Left IR Distance / OPT3101	Analog Arm Tilt Servo	Reflectance LED Illuminate (even)	Nokia5110 RST	AUXL IR Distance / OPT3101	Reflectance 2	Reflectance 0	Nokia5110 Clock	Nokia5110 MOSI	Reflectance 5	Reflectance 7	AUXC IR Distance / OPT3101	UCB3SCL (not available in	ELA (3.3V) [1]			

11/19/2018 changes from version 6, jan@pololu.com
2/11/2019 changes from rom05a02

Figure 2: MSP432 pin chart (source: MSP432 Pinchart file on week 3 CCLE)

Secondly, we need to know about the QTR-8RC reflectance sensor array, the sensor board on the car's bottom. Generally, this sensor helps the car detect the light level and the voltage corresponding with this level of light. The schematic of this board is shown in Figure 3, and the circuit to calculate the voltage based on the light level is shown in Figure 4.

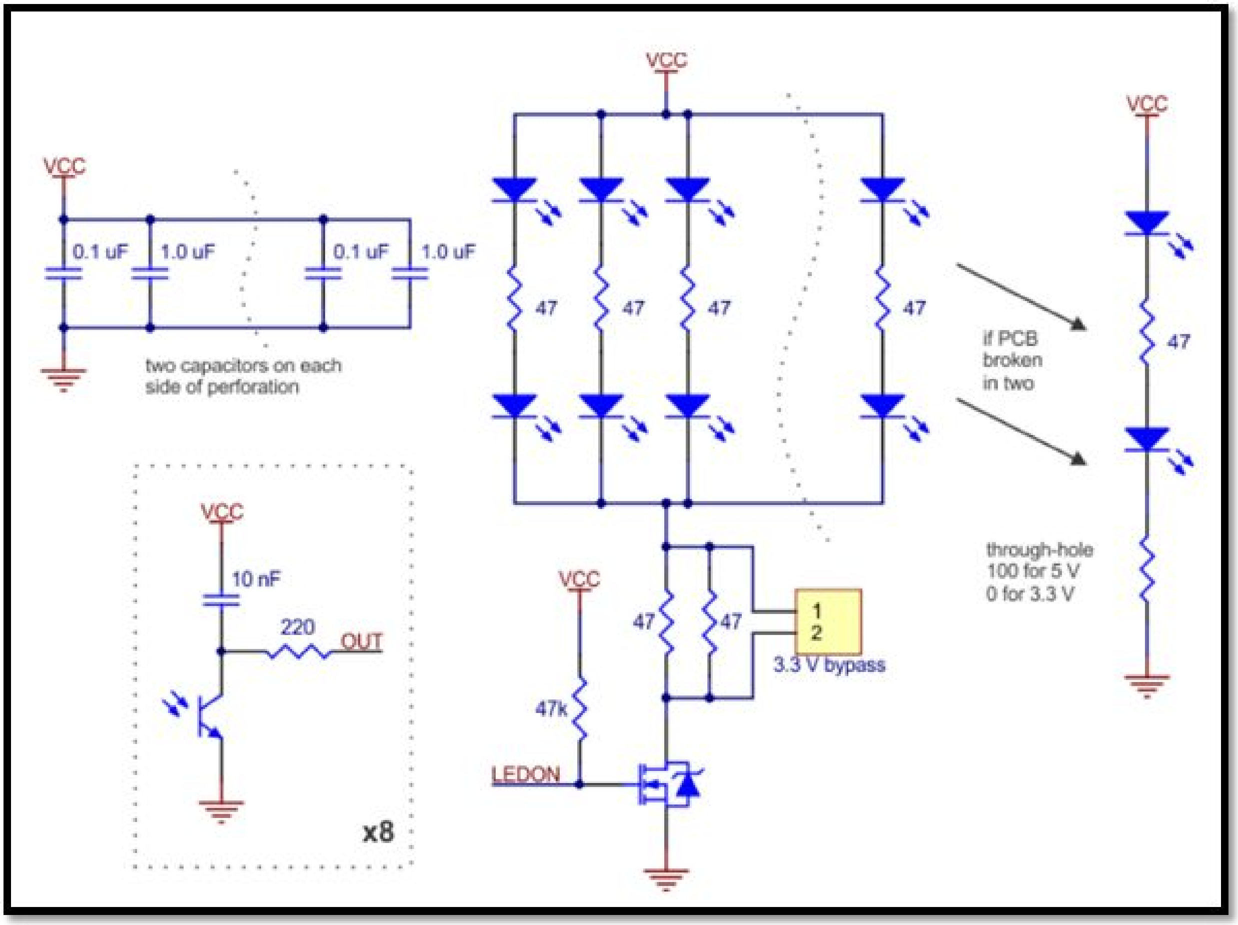


Figure 3: Schematic diagram of a sensor board (source: https://www.pololu.com/product/961)

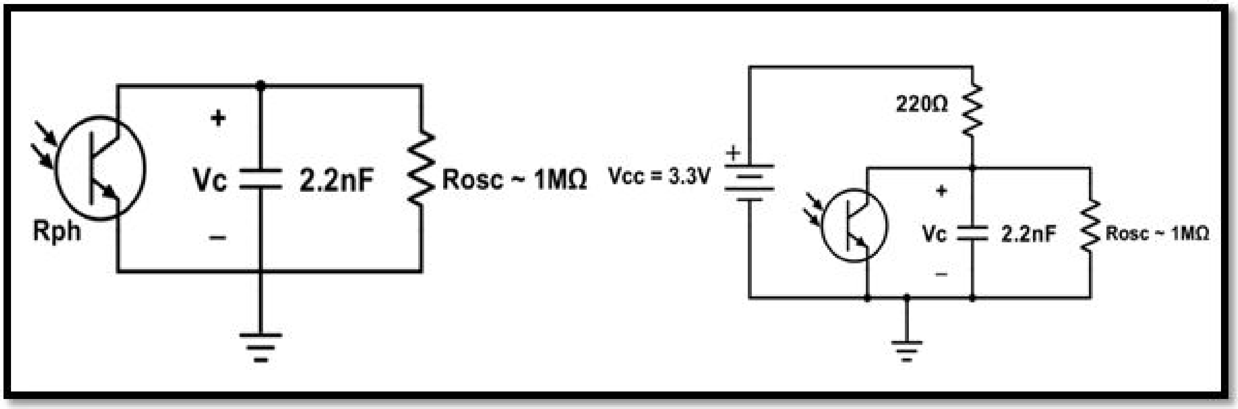


Figure 4: Phototransistor circuit in stage 1 and stage 2 (Source: Lab 3 on CCLE)

The default sensors read values from 0-2500, respectively, from white to black. However, the sensors have not the same ranges in real life. For example, sensor 1 has a range of 0-2500, but sensor 2 has a range of 0-2450. Therefore to make sure the sensors work consistently, we need to convert them to the same fixed range of 0-1000. The chart of the sensors' data before and after the calibration is shown below[2].

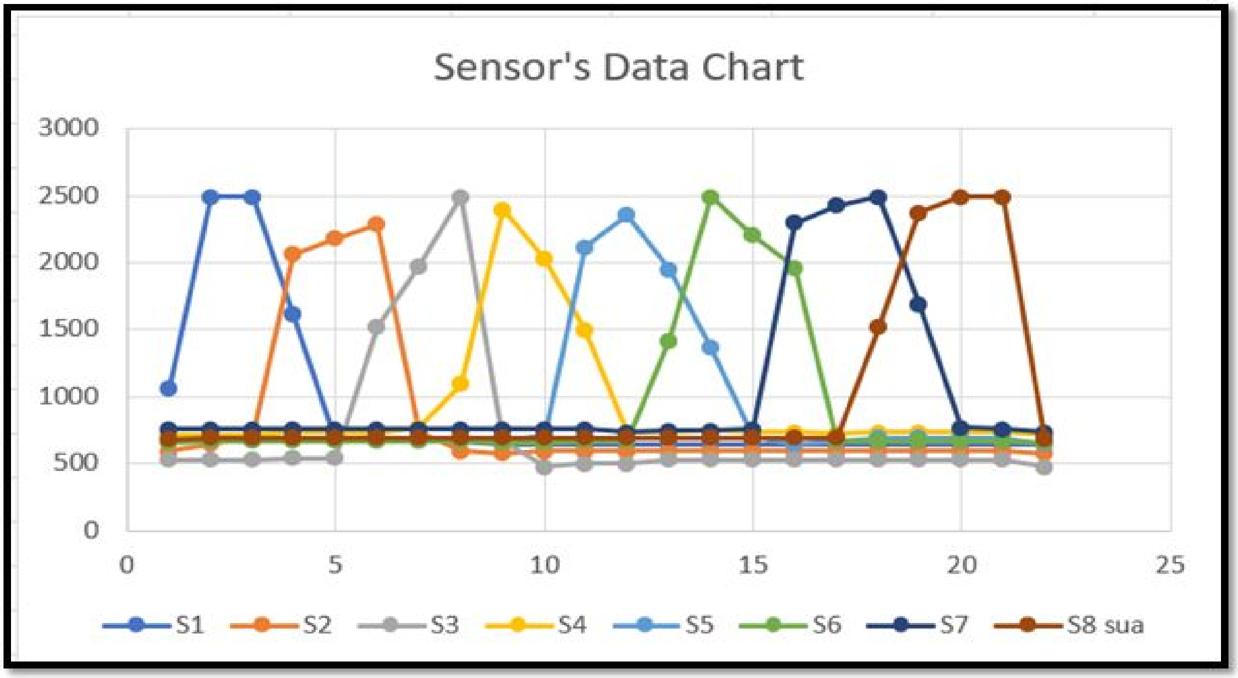


Figure 5: Sensor's Data Chart

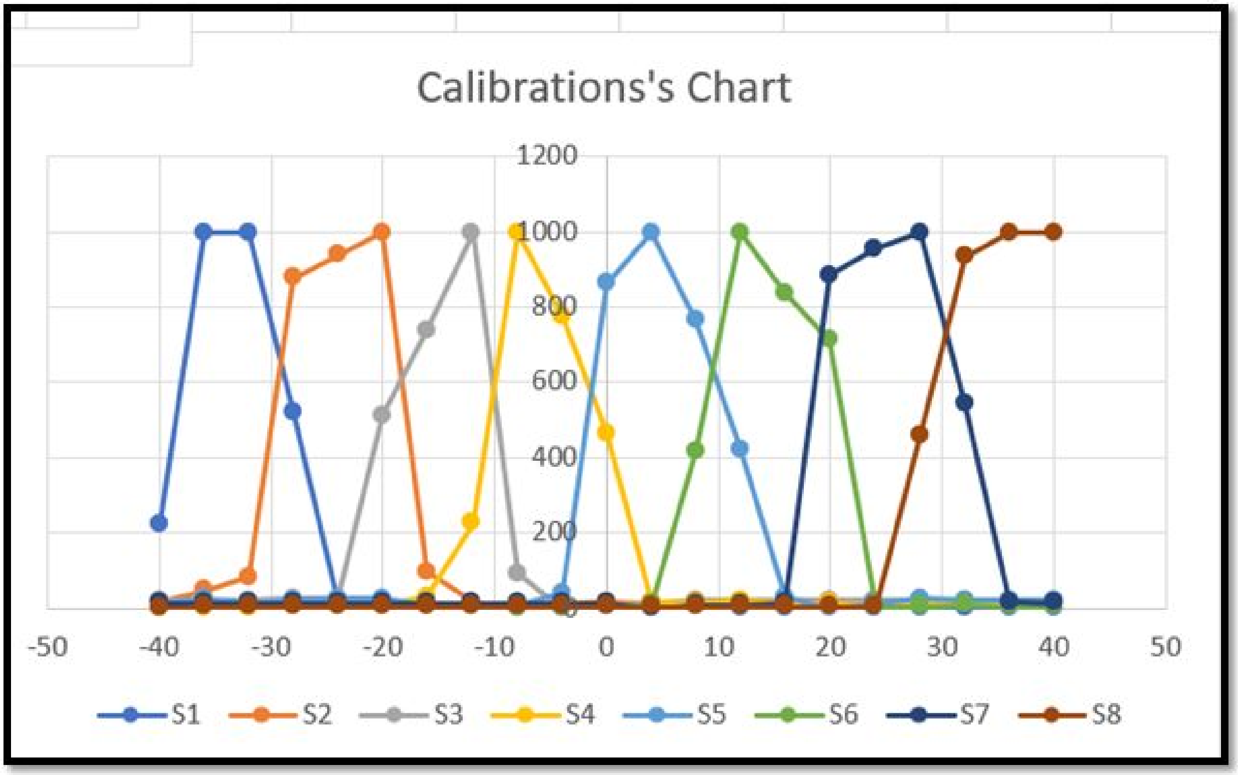


Figure 6: Calibration's Chart

After obtaining data from the sensor, the microcontroller will use this data to determine whether the car's current position is on the black line or not. If the car is not on the black line, we know that it is necessary to steer the vehicle to follow the black line. Or a more specific example, if the rightmost sensor is detecting a black line, it means the car is facing the left to get out of the black line, so what we need to do is control the car again towards the right. Besides, to increase the accuracy, we can use the vehicle's horizontal position relative to the track using sensor fusion. First, we combine all the weights of the sensors that detect the black line according to the set of numbers (-8 -4 -2 -1 1 2 4 8) we have learned in the lab section, then use this value to determine the car’s movement. And the Sensor fusion output is shown below.

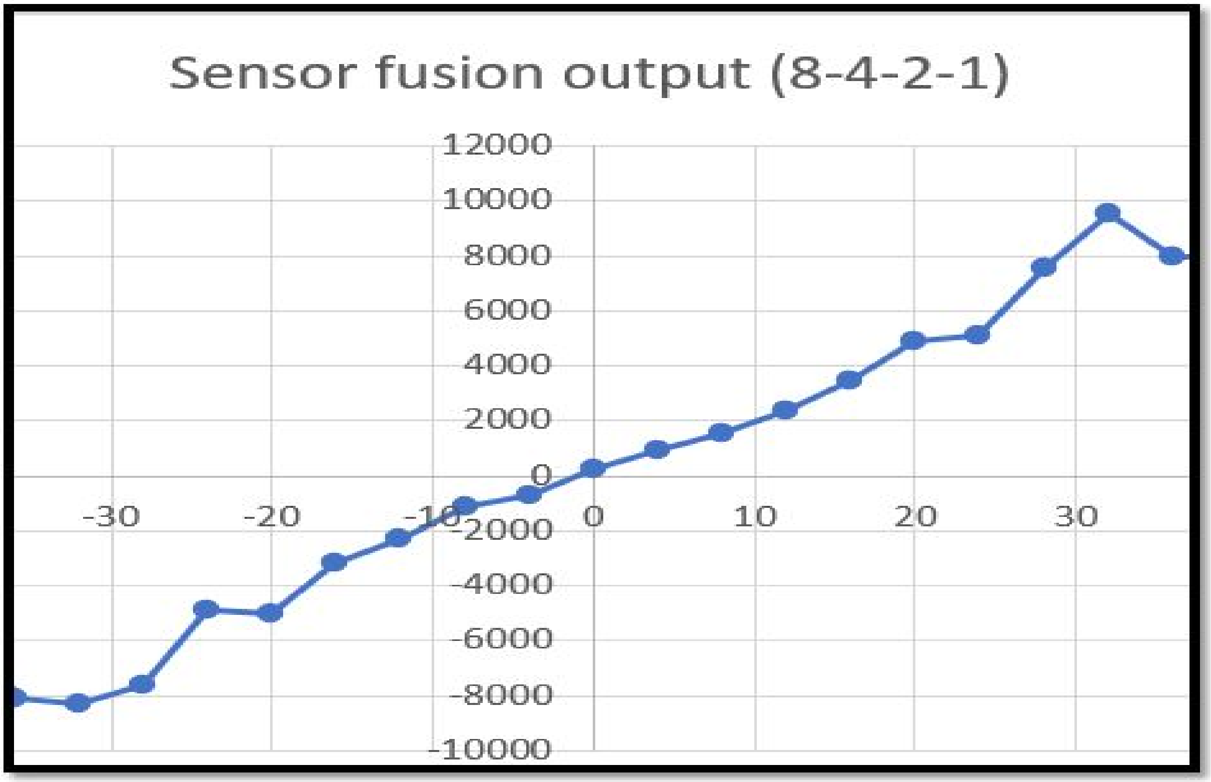


Figure 7: Sensor fusion output Chart

2. On-Track Development Methodology:

a. Test Setup

For our development and testing, we chose a spot indoors that had consistent lighting with minimal exposure to outside lighting. This allowed for the minimized effect of natural lighting as our testing was taken anytime during the day, whereas the race-day would be at a specific time.

For electrical characteristics, our codes were based on the values from the sensor values, which varied from 0 to 2500. However, we had to utilize different sensor values between the two cars as each car's sensors provide different sensor values on the same parts of the track. One major setback faced was that one of the cars could not correctly detect the end of the path indicated by a black bar. Theoretically, the black bar was supposed to be seen as 2500 by all eight sensors, but the problematic car never fully reached 2500 on the sensors. The problem was remedied by lowering the condition of detecting the black bar to 1600 on all sensors.

b. How the tests were conducted.

We first wrote a simple program to enable the car to run in a straight line at a fixed speed for its two engines. From here, we can check whether the configuration for the pins of the circuit or the output of PWM pulses for the motor is correct or not. Also, if the car runs straight as desired, it proves that the engine system works well.

Then, to make the car turn around, we tried to write a program that allowed the vehicle to rotate to see how to control the car to turn over. After knowing how to manage the car to go straight and rotate, two essential elements for the car to work on demand, we learned how to handle the car to follow the black road. Thanks to the lectures and the lab section experiments, we learned how to use the automatic control PID[3]. We started rewriting our entire code more intricately by incorporating the results of learning how to calibrate, use fusion, and, most of all, how to apply the PID algorithm to the code.

To understand and test how the PID algorithm works, we test cars running on straight-light. We first decided to give a small speed value to make it easier to see how responsive the vehicle is on the road. Next is the choice of values for Kp and Kd. Due to the inexperience at first, we changed the Kp and Kd values at the same time, so even though we experimented many times, we still could not choose suitable numbers for (Kp, Kd) the car can run stably even though the initial speed value was only 30/255. Following TA's advice, we decided to use only one Kp parameter first and try to adjust this until the car can keep the black line while the Kd value is given to 0. And the result was as expected with the set of numbers Kp = 0.01 and Kd = 0.

Although the car could move on the black track, the stability was not high because sometimes the car still ran out of the path, or the car was not running very smoothly. So we continued to use Kd's value by gradually increasing the value of Kd from 0 until the car could run smoothly. After the vehicle has stabilized with the Kp and Kd set, we started speeding up, so we repeated our tests to find Kp and Kd's numbers.

Once finished on the straight-line, we started running and testing on the official curved. Because there are many curves on this road, it is more challenging to keep the car running firmly when we set the car at high speed. If we put it to 60/255, the car can still finish the road, but it will take longer by 10 seconds. So what we need to do is to improve the code. At that point, to take the error for using the fusion, we only examine the difference between two consecutive points in time. Therefore, when running at high speed, the car will not keep the black line well because there is little false information. To solve the error problem, we decided to get more error information using an 8-element array to store the previous 8 errors information for each time the vehicle reads the sensors' values. With this approach, we have solved the problem of time while ensuring the car runs stably. Finally, after going through the steps to refine the Kp and Kd values again, the best running numbers in our group were Speed = 120/255, Kp = 0.02, and Kd = 0.15.

c. Data Analysis

To test our code and find the best performing values of speed to Kp and Kd ratios, we ran multiple tests and recorded each setup's results. The table below summarizes the results of our testing.

Nhat Ho's Data:

Speed	Kp	Kd	Comments
110	0.01	0.1	Fast but goes off track
50	0.01	0	Slow, but first completed a run
60	0.015	0.1	Oversteers but still completes the track
60	0.01	0.15	Oversteers sways on the track
60	0.01	0.1	Run well and completed the track
70	0.01	0.04	Completed the track, just a little bit sways
70	0.011	0.06	Runs well, overcompensates on turns
80	0.01	0.01	Sometimes runs off the corner in the middle of the track
80	0.01	0.05	Usually ignores the track, not stable
80	0.015	0.1	Completed the track, stable
90	0.02	0.2	Fail sometimes, not stable
90	0.015	0.1	Completed the track, stable
110	0.015	0.1	Fail sometimes at the curves and out of the track
110	0.015	0.2	More stable but still fail when running many times

110	0.02	0.15	Very stable, best record to this point
130	0.022	0.18	Faster but still fail sometimes at the curve of the path

Leeksun Cho's Data:

Speed	Kp	Kd	Comments
110	0.02	0.15	Fast but goes off track
110	0.015	0.18	Fails to detect bigger turns
50	0.01	0.1	Slow, but first completed run
60	0.01	0.05	Oversteers but still achieves the track
60	0.015	0.04	Oversteers sways on the track
70	0.015	0.04	Runs well, overcompensates on turns
80	0.02	0.04	Unable to detect the end, runs off the corner in the middle of the track
80	0.02	0.04	The issue on the way back, ignores the track
70	0.01	0.04	Runs very smooth, reliable
70	0.01	0.035	Runs smooth, no jitter
70	0.008	0.032	Very stable, will be used as control variable
84	0.0096	0.0385	Very stable, best record to this point
90	0.01	0.045	Does not work with position 2 and 3
90	0.02	0.045	Does not work, will revert back to 84 speed
84	0.096	0.039	Recalibrated on the track-day, consistent record of 9.1 seconds

d. Test Data Interpretation

The initial test began with high hopes for a fast car, and thus we started with the speed at 110. However, it soon proved too unstable the car was unable to adjust to the path correctly. Restarting the test from a slow speed of 50 and increasing from thereon, the results show that as we increased the speed of the wheels, oftentimes Kp and Kd needed to be increased as well to follow the track properly. Raising the K values led to the oversteering of the cars, and lowering them caused understeering. We noticed that Kp changes caused a larger deviation in the turning of the car, and Kd caused smaller changes. Despite the trend of the higher speed of the wheel needing a higher Kd value, the golden spot for speed between 70 to 90 seemed to lie between 0.01 and 0.02.

3. Results and Discussion:

a. Test Discussion

Our choice to use Kd and Kp values were ideal for the course as it allowed for both general changes and precise tuning of the smoothness of the turns and movement of the car. If we did not use two variables, we would have lacked the ability to fine-tune our car's motion. It allowed for a greater level of precision and led to time-effectiveness as we were able to figure out what Kd value was appropriate to complete the course at each corresponding speed. By changing the Kp values, we were able to improve the smoothness of the track runs. Practically we were able to use Kp to adjust the car's skittishness, and while it may not increase the speed of the runs, it definitely led to more stable and consistent runs.

As much as our goal was to design a car and code that can complete the course as fast as possible, a fast car would have been useless if it wasn't consistent in its runs. As evident from the table above, there were some combinations of variable values that led to fast completion of the track in certain positions, but not in others. This was frequently due to the oversteering or understeering (depending on the variable combination) in some of the places. Our usage of Kd and Kp allowed adjustments to such steering issues on both small and large scales.

It is possible for us to have achieved even faster speeds, but the problem rose with the sensor values' inconsistency. At higher rates, one of the cars would sometimes become unable to detect the track's end (indicated by the black bar) and simply run off the track. This was problematic as we would need to get the sensors' values at the time of error and fine-tune the entire code for that single car. That is why, despite developing and using the same code, 1 in 2 cars did not achieve better speed and stability than the other. As we can see in the data report, the first one ran best at speed 110 (7.4s), while the second was only stable at 84 (9.1s). The fact that the mistake happened occasionally and not consistently added onto the problem that it may be a hardware issue which we had no control over.

b. Race Day Discussion

On the race day, our cars performed wonderfully, with one car completing the track in 7.4 seconds and the second car completing its run in 9.1 seconds which are both within the range of extra credit requirement of 11 seconds or lower. The links to the videos of the runs are provided here:

Car1:
https://drive.google.com/drive/folders/1Ieue65hPGz4P8V9Rszp12IBW2rX29jOZ?usp=sharing

Car 2:
https://drive.google.com/file/d/1z4ltofruo8-aMAF6zVJo-sEpC34C63M9/view?usp=drivesdk

We did test Car 1 at a higher speed (at speed 130), but it failed to complete its run on the race day. It was probably affected by the difference in lighting during the time of our testing and the race day which led to slightly different sensor values. The problem with Car 2 was its occasional inability to detect the end of the track thus limiting the runs on the race day to lower speeds. As the speeds got higher, our code seemed to struggle even with the minimal errors in its starting positions caused by human error. At higher speeds, even slight angle differences on the track would affect its ability to complete the run on the track, thus we had to stick to lower values that worked from our previous testing results shown in our data table.

During the implementation process to our vehicles' successful run, we understand that when working with automated system devices, the first thing to do is test the devices to ensure they are fully functional works well. Besides, the environmental factor plays a significant role, so we have to control the device correctly based on the changing environment to get the best results.

4. Conclusions and Future Work:

Overall, our testing and development process was successful as we met the design goal within the development time of 5 weeks. Our group's code and design performed spectacularly with the first car completing the track in 9.1 seconds and the second car in 7.4 seconds. This speed was even fast enough to meet the criteria for extra credit which was a track time of 11 seconds or lower. We learned the importance of time management and data logging for testing. When our theoretical code was actually put to the test in the real environment, data log from testing tremendously helped to backtrack from problems and see what worked and what did not. If we had more time, we would like to develop an extension that will be able to handle vertical changes such as steep hills. Vertical changes may affect the distance between the track and the sensors leading to different sensor values than what is originally expected. This would require extreme amounts of testing and data to help differentiate between the car moving off track to the car's moving along a hill especially towards the top where the angle between the sensors and the track will be slanted.

5. References:

[1] https://energia.nu/

[2] The process taught by Professor Briggs

[3] Theories and ideas based on lectures by Professor Briggs