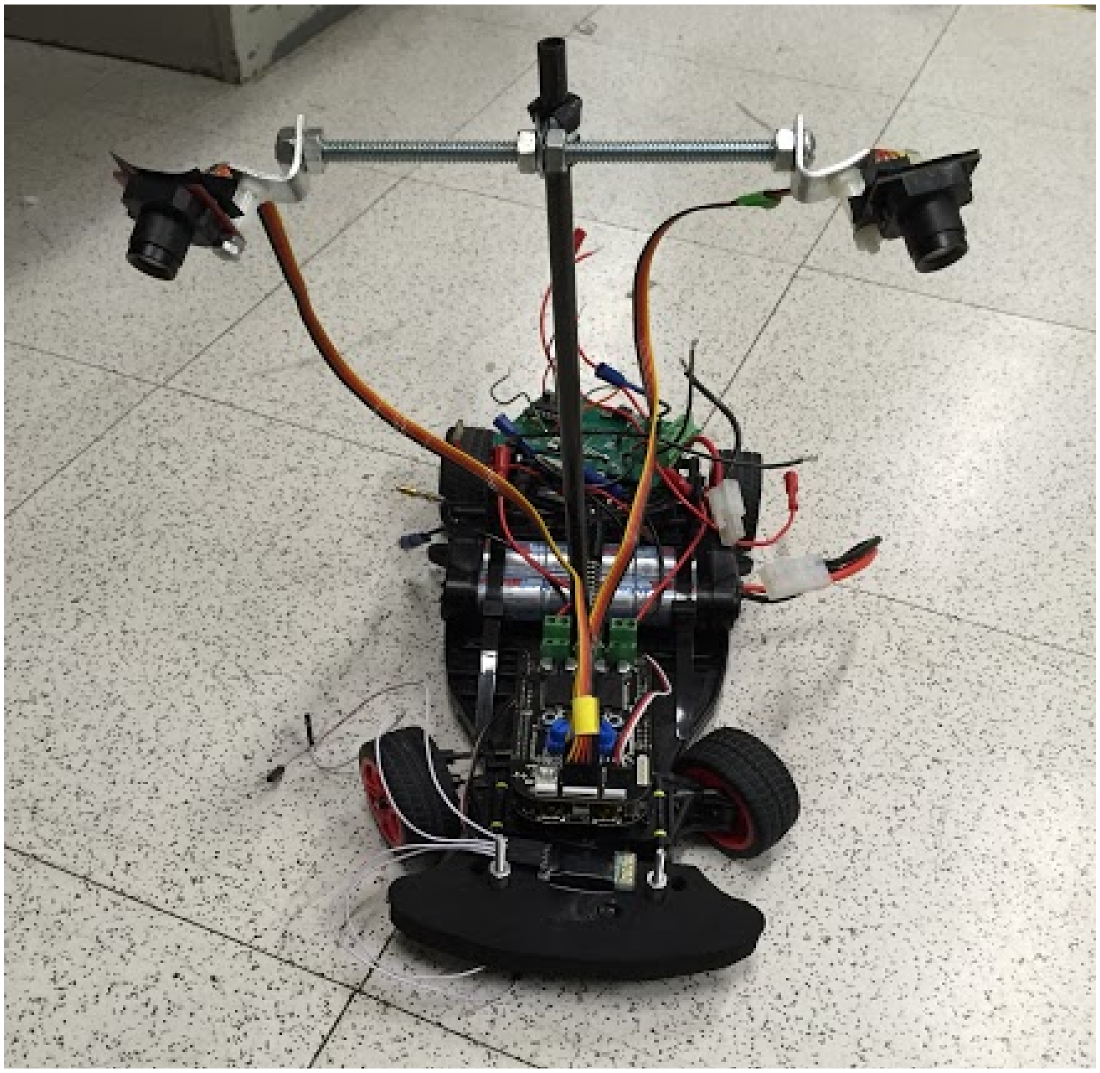
NATCAR FINAL REPORT



Patrick Yu, Ronald Lai, Shawn Xiang

Team RPS

Professor Halsted

EEC 195

March 17, 2015

Acknowledgements

In the creation of our project, we received much help from our instructor, Lance Halsted and teaching assistants, Zhiwei Zhao, Roger Lo, Vivek Dubey, and Patrick Mcgregor. We would like to express our gratitude to their guidance.

"Logic will get you from A to B. Imagination will take you anywhere."

-Albert Einstein

Table of Contents

I.Executive Summary

During the past two quarters, we have been working on an autonomous race car that is controlled by a microcontroller with linescan cameras and powered by a DC battery and servo motor. The aim of the project is to complete the entire length of the track as fast as possible. For the first quarter, we worked on the lab assignments that cover all the basics of the Natcar. For the second quarter, we started to assemble the car using the kit and additional materials and test our algorithm. During this quarter, we also implemented the bluetooth, PCB and various tuning to perfect our control algorithm. We wrote the algorithm, working with linescan camera and spent time testing together. For the individual work, Shawn implemented the bluetooth and explored with the 2D camera. Ronald tested and modified the speed, PID, and linescan camera parameters. Patrick designed the layout of the car, comparator PCB, and motor control PCB.

II.Detailed Technical Report

A.Car Design and Layout *Written by Patrick Yu*

The overall objective for the design and layout of the car can be summarized in four points: (1) Minimal weight added, keeping the car lightweight, (2) even weight distribution, (3) low center of gravity, (4) sturdy and stiff chassis. In addition to the kit car, this report will present our reasoning aspects of our design, and explain the tradeoffs we have made in determining our final layout of the car.

The platform for our build came in kit from Freescale, and we were limited on how we could modify the chassis. One legal modification we made to the chassis was to stiffen the spring compression and rebound by placing C-shaped collars at the end of the suspension shaft to shorten the spring travel. Originally, the car had very low ground clearance, especially in the midsection of the car. This modification allowed the car to support more weight without risking the car bottoming out. There is no negative tradeoff for this fix, except for adding necessary weight for the car to function properly. All other aspects of the chassis are unchanged from the original design of the car in the kit.

While brainstorming where to mount the KL25Z board with the TFC shield on, we determined that there were limited spaces where one could mount these components. We decided to mount the components right above the servo motor in the front section of the car. All we had to do was add two standoffs and screw them onto the existing, pre-drilled holes in order to mount the boards. This solution required the minimal amount of support bracing, which means adding the least amount of weight possible to keep the car lightweight. Since the DC motors are mounted in the rear section of the car's chassis, the car was inherently rear heavy by design. Adding weight to the front would help balance the weight distribution toward the front. Similarly, we want to have sensitive circuitry away from high current lines like the DC motors that produce large amounts of EMI and noise. The furthest point in the car to mount such circuitry, like the KL25Z board, was in the front. Consequently, this arrangement still allowed for a low center of gravity. The area above the servo motor was the lowest part in the front of the car we could mount the KL25Z.

As for the pole to mount the cameras, we wanted to place the pole in a central location. We decided on securing the pole slightly in front the suspension meets the center of the chassis. This is the most centered location on the car. Objectively, we wanted the pole to be at the axis of rotation, which would be near the center of the car. The tradeoff for securing the pole in the center is the chassis flex. The car is separated into two sections, the rear section with the DC motors, and the mid to front section. There is lateral movement in the chassis when the car makes quick changes in direction, because two sections of the car are not connected together except at two points. As for the pole itself, we have used a carbon fiber tube. Carbon fiber is strong, stiff, and not very flexible, while still being lightweight. We chose to use a single tube instead of a rod, because hollow pole means less weight. The two cameras are extended out to the sides with a bolt and L brackets on each end, which allowed us to adjust the cameras up and down. Mounting the cameras up high means the center of gravity is also higher. However, we opted to mount the cameras up high in order to see further up ahead with greater field of view.

We placed the custom motor control PCB in the rear of the car right above the DC motors. The reasons for this is to keep the high current wires leading to custom motor control PCB close to the DC motors. We did not want unnecessarily long, high current wires to affect our noise sensitive, sensing circuitry. However, the tradeoff for this placement is that the car will be heavier in the rear, affecting the weight distribution biased toward the rear. We are willing to make this tradeoff in order to maintain stability and consistency of the sensory data. The custom motor control PCB will also increase the overall weight of the car.

B. PCB Design and Implementation: Motor Control *Written by Patrick Yu*

To qualify for the NATCAR competition, our team needs to have a custom motor control board. We could not use a commercially available motor control board , like the one built into the TFC shield. Although we have made a pre-designed custom motor control PCB in lab 8, that board is not Freescale Cup legal because it does not use Freescale technology. The motivation for creating our own custom PCB is for it to be both Freescale Cup and NATCAR legal. Additionally, we want to separate the motor control circuitry away from noise sensitive circuits, and place the motor control board closer to the motor. In the TFC shield, the connectors to the sensors, pass through pins to the KL25Z, high voltage motor control interface are all on the same board. Objectively, we want to separate high voltage components, like the DC motor control, from the microcontroller and sensors to reduce the noise and interference.

In designing the motor control PCB, we decided that it should have several features. For example, we want the physical dimensions of the board to be as compact as possible within the design specifications that the PCB fabricator allows and have a fairly small footprint. The tradeoff was that all the components would be crammed onto the board, making it more difficult to solder the parts on. However, a smaller footprint means the board would not be hanging off the edge of the chassis and fit on the car. In deciding on the layout, we wanted to place the connectors in convenient locations, so external wires could be only as long as they need to be. Additionally, we included specific drill holes that matched the location of pre-existing screw holes on the chassis above the motor, so we could secure the custom PCB onto the chassis. Overall, our design aimed for a custom fit motor control board that would replace the TFC shield's motor control.

The actual design of the motor control PCB is referenced from the TFC shield board, but only includes the motor control portion and battery to 5 volts voltage regulator from the TFC shield. Just like the TFC shield, each of the signals from the microprocessor goes into a M74VHC1GT50DTT1G buffer acting as 5V voltage regulators going into each of the MC33887 IC. The MC33887 is an H-bridge DC motor control IC with current feedback, and is the same chip used in the TFC shield. Our control algorithm relies on the feedback signal for speed control over hills. For this reason, we chose to use the same IC with a similar design for our custom PCB. The motors are connected across OUT1 and OUT2 pins of the MC33887. The motor speed is controlled by pulse width of the PWM signals produced on the microprocessor, up to 10 kHz. In addition, the MC33887 has protection against shorting out the H-Bridge built in, which is good design against failure. The MC33887 is powered at the battery voltage at 7.2V when full charge, which is within the 5.0V to 28V operation range. In a separate part of the board, we included a battery voltage to 5V voltage regulator using the NCV1117ST50T3G. The 5V output would be used to power other parts that need 5V input, like the AD8032 operational amplifier for the comparator circuit. As for the capacitors and resistors, they are used for filtering out noise.

C. PCB Design and Implementation: Comparator *Written by Patrick Yu*

When trying to sample the full 320 by 240 resolution of the OV5116N 2D camera, we are presented with a problem. We need external hardware to sample, convert, and interface the fast video output from the camera to the KL25Z. We weighed two potential solutions for our problem: use an external 8-bit ADC or use a comparator to do a single bit conversion. The AD7278 ADC could only provide us with about 80 pixels per line using the SPI at 12MHz baud rate, which is not full resolution. With the intention of getting full resolution and maximizing the capabilities of the 2D camera, we chose to use the comparator to do a single bit conversion.

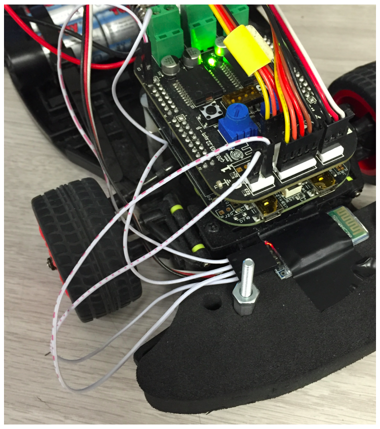
In designing a comparator that interfaced the 2D camera to the KL25Z board, we had two principles that guided our design and layout. First, our circuit had to be simple and small, but still function the same without compromises. The footprint of the PCB needed to be small enough to fit in the confined space we have remaining, and to mount the PCB onto the chassis with existing screw holes. Consequently, a small and simple PCB also means we would be adding the least amount of additional weight to the car. Secondly, we put another constraint on the design of the comparator circuit. We wanted to use resistors and capacitors we have available to us without having to specially order specific parts. It is cost effective and convenient to use parts that are available on hand, because we can maximize efficiency.

All things considered, we came up with a solution that meets the objectives described. Our design consisted of two first order amplifier circuits put together in a chain. Looking at the schematic, the lower amplifying circuit, configured as a comparator, works by comparing the potentiometer set voltage with the camera data to determine the threshold value. The threshold value is then fed back into a second comparator on the top half of the schematic, to it with the 2D camera's composite video output signal. The overall output of the circuit, D1, is the single bit that delineates a black pixel on a signal low, and white on a signal high.

Since we needed to make two comparisons, we chose to use the AD8032 dual operational amplifiers IC, configured as two comparators. The AD8032 runs at high speed and has fast settling on 5V (125ns settling time), which is fast enough for our sampling needs. Additionally, it runs at 800 μA per amplifier, so it draws a low amount current. The AD8032 fits the need for a fast comparator and low power draw, which is beneficial for operating the circuit on battery power. In order to verify that our design works, we built our design on a breadboard. First, we powered the 2D camera and circuit at 5V using a power supply. Then we connected the composite video output signal and D1 output signal from our comparator circuit to an oscilloscope. Next, we turned the potentiometer in order to set the threshold value to compare with. The circuit was working as intended if the D1 signal remained low for the active video portion of the camera signal looking at a completely black image. If the vice versa were true for an all-white image, then the comparator circuit is functioning as intended. For the most part, this is what we saw, except for the line blanking portion of the signal.

D. Bluetooth Serial Communication *Written by Shawn Xiang*

Setup



To make the tuning process more convenient, we implemented the bluetooth module. The chip we used is HC-05. There are many bluetooth arduino devices that use this chip.[1] It has a recommended voltage of 3.3 V (operating voltage: 2.7 to 4.2V) and a maximum baud rate of 1,382,400. We had tested the working distance that it can transmit data in about 20 meters (65 ft). After about 20 meters, the bluetooth transmits data in an exponentially slower speed. Since we are using the speed sensor, we can use the 3.3 V supply from the Vcc pin for speed sensor in TFC shield.

From the code given for serial communication, Rx and Tx are set to be PTD5 and PTA1. Therefore, simply connect Rx to PTA2 and Tx to PTA1, in addition to the power and ground, and the LED indicator on the bluetooth will light up– the bluetooth is now ready to use.

Configuring the Baud Rate

Because the default of the bluetooth module has 9,600 baud rate, we need to set it to 115,200 that matches with our program. Simply changing the baud rate of our program to 9,600

[1] I chose the one with the best ratings on Amazon: http://www.amazon.com/gp/product/B0093XAV4U/ref=oh_aui_detailpage_o00_s00?ie=UTF8&psc=1

to match the default baud rate would not work[2], given the baud rate in the program is about 10 times larger than the default baud rate. To change the default setting, we need to enter the AT mode. In HC05 chip, it is only allowed to enter to the AT mode when the device is idle or *not* paired. However, typing to a terminal can not work because it is impossible to type a full command fast enough for the bluetooth to identify as a string in AT mode. After reading many articles, I found that the only solution to this problem is to send files via the terminal. A serial terminal, such as TeraTerm, has the functionality of sending a text file. Writing appropriate command, for example, changing the baud rate to 115,200 would be "AT+BAUD8", and sending it in a file successfully changed the baud rate as the terminal echoes "OK115200". You can also configure other settings, such as password, name, etc.

<u>Bluetooth in Action</u>

After the device is set, we can receive strings sent from the microcontroller as if it is connected with a serial port. We can read the bluetooth data in a laptop, tablet or smartphone.[3] We no longer have to limit ourself with a long cord of wire and chasing the car while looking at the data.

The method we used to test the car using bluetooth is to first enter a few parameters for the test run. Then if the car runs out of the track, we immediately reset the car. Then we will take a look at the data, such as value for center of the line, derivative to figure out why it ran out the track. Lastly, we will conclude what we need to change either the parameters or some part of the code to make the car run better for the next test run. With bluetooth, instead guessing what was wrong, we can see what the car is seeing. This significantly reduces the time for guessing.

[2] If we change the baud rate 9,600, the program would not output anything, from either the bluetooth or serial port
[3] HC05 does not support iOS devices but it works with any Android, Mac and Windows devices.

However, the main disadvantage of using bluetooth is as it draw about 35 mA current from the board that drains the battery more quickly. It is also important to note that we are not allowed to use bluetooth in the either competition because one can technically control the car via bluetooth. Bluetooth is hence used for testing or tuning only.

E. Linescan Camera *Written by Shawn Xiang*

Setup

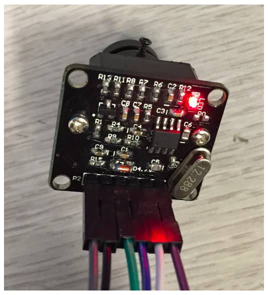
Because we originally anticipated that the linescan camera will not able to see the whole track. We decided to use two cameras as recommended in class. However, because two sensors are used, we have to make sure the scanned camera data are matched. If they are not adjusted, for example, when the car enters to the intersection, the left one might see the intersection a fraction of seconds before the right camera which will cause some problems. Another way to deal with it is to account for the time difference between the two cameras. This will need more variables to be used and sometimes the result may vary in high speed. Therefore, the better approach is to align the two camera as symmetrical as possible.

Exposure Time

The exposure time is depending on the lighting condition. If the room is bright, we should have shorter exposure time to allow contrast whereas longer exposure time to allow pixel to reach the allowed voltage. The PIT period and delay in the ADC and PIT handlers, if any, will determine the exposure time. The PIT period we found adequate to most of the time is about 5000 to 8000 microseconds.

F.2D Camera *Written by Shawn Xiang*

Setup

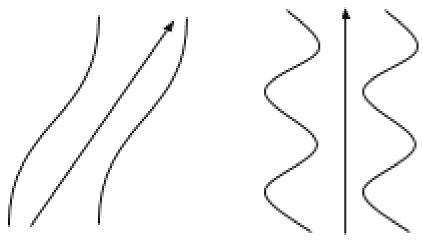


We tried to implement the OV5116, the only allowed 2D camera in the freescale competition. The camera we brough has an integrated video sync separator chip , LM1881 to separate out the signals. After the camera is connected with power (grey) and ground (black), it outputs four signals, analog signal(green), digital signal or D1 signal (blue), field sync signal (white), and line sync signal (purple).

The camera can be operated with 5V input. The analog signal outputs an analog signal for each of the line in a voltage range of 1 V, much like the linescan camera output AO. The digital signal outputs very similar image as analog signal, but it is in 5 V and inverted compared to the analog signal. The field signal indicates the field of the camera. NTSC camera has two fields, even and odd field. The line signal indicates the trigger of each line.

Advantages

We think having a 2D camera would help the car to run faster and more accurately. The biggest advantage of the 2D camera that it can see further away with more data. If we mount the camera at the maximum height, it can see up to 2 meters of distance ahead. The data are more conclusive and essentially larger than that of the linescan camera. It can prevent the car to follow an erroneous path when the track is unclean or hindered by shadow. For turns and intersection, the camera can identify the shortest route possible. (left image) In chicanes (snake shaped



edge lane), we can make the car go straight instead of oscillating around the edges. (right image).

Sampling

Because the camera has a very fast sampling speed, we can not use the internal ADC converter. Instead, we need to use a comparator circuit. (for more discussion of comparator circuit, see PCB section) Based on the variable resistor, we can adjust the threshold value. We then use the sync signals and the digitized signals from the comparator to the board. The detailed comparator circuit is in PCB section

The sync signals are used as the inputs to trigger GPIO port interrupts. To do that we need to initialize the GPIO to update interrupts. For example, if we connect PTA1 with the field signal:

```
PORTA->PCR[12] = PORT_PCR_MUX(0x1) | PORT_PCR_IRQC(0xA);   // on PTA12
```

We also need to initialize the PORT A handler:

```
NVIC_SetPriority(PORTA_IRQn, 64); // Priority:  0, 64, 128 or 192
NVIC_EnableIRQ(PORTA_IRQn);
```

We do the same thing for the line sync signal, camera sync signal and port D handler. The GPIO handlers will be used to collect the data. So first when port D receives a field signal triggered, it will initialize all the variables

```
unsigned int CamData [ROWS][COLS]; // 2D array camera data
unsigned int row; // update the number of line in the image
unsigned int doneFlag; // indicate the image process is done
unsigned int fieldFlag; // identify even and odd field of the image
/***** Processing a new line ******/
void PORTA_IRQHandler (void){  // triggers by line sync signal
      int i = 0;
```

```
	NVIC_ClearPendingIRQ(PORTA_IRQn);

	for (i = 0; i < COLS; i++) {

	CamData[row++][i] = (FPTC->PDIR != 1 << 1) // read the data   }

	if (row > MAX_ROW){  // check for the end of the image

		doneFlag = 1;

	}

		PORTA->ISFR|=PORT_ISFR_ISF_MASK;}

	/***** Processing a new image *****/

void PORTD_IRQHandler (void){  // triggers by field sync signal

	NVIC_ClearPendingIRQ(PORTD_IRQn);

	row = 0;

	fieldFlag = !fieldFlag; // toggle between even and odd field

	PORTD->ISFR|=PORT_ISFR_ISF_MASK;

}
```

Challenges

The data we received now can differentiate all white and black images but not very precisely matched with the actually picture. There are some issues with delays of sync and camera that we need to take into account and there are occasionally some garbage data when it is bright. Sometimes the program would quit unexpectedly, so we need to implement DMA(Direct memory access) in the program to prevent processor overload.

Due to the time constraint, we were not able to finish it this quarter. We still have to solder on the comparator PCB as well as works involving filtering out noise, matching sync signals, threshold based on lighting condition and image processing. But we are hopeful to use it in the coming Freescale competition and NatCar competition. If everything goes as expected, we will write an update on our work in 2D camera for the design showcase.

G. Speed, PID, and Linescan Camera Tuning *Written by Ronald Lai*

To fine-tune our car's turning accuracy and speed, we implemented a PID and made adjustments to a list of parameters that will be described in this section. The proportion and derivative constants were adjusted alongside the turning and speed parameters to maximize our car's track completion time. By actively conducting test runs while varying our turning, speed, and PID parameters, we can evaluate which of the variables need modifications in order to improve the car's overall efficiency.

Speed Tuning

To tune the car's speed, we made a slight modification to how we controlled the speed of the car. Originally managed by a potentiometer on the TFC shield, the speed of the car did not have a discrete numerical value, which made precisely tuning the speed difficult. To fix this issue, we modified our code to induce the duty cycle of the POT digitally, through data entry on a terminal to the variable, dutyCycle1. Setting the ports for the POT's, using TPM0->CONTROLS[0].CnV and TPM0->CONTROLS[2].CnV, we are able to set the speed of the motors for the left and right wheels independently. After taking note that the maximum value of the POT's for the wheels should not exceed 9600, we found that for our car's current setup, values of around 3500 to 4000 yield the best results for turning accuracy.

In order to effectively convey how we tuned our constants, we can examine one of the equations we used in our code in main.c of the Appendix. When the car attempts to turn right, our program calculates the left motor's speed as follows:

PW1 = dutyCycle1 + (c1CurrentLocation - leftEdgeMacro)*kpMotorR;

Here, the left motor speed (PW1) is adjusted by our left camera's current center and it's true center of line pixel, the difference of which is multiplied by our right-turn motor constant. A similar calculation is used for the right motor speed (PW2), using our left-turn motor constant (kpMotorL).

An important concept we applied for our speed control is that when our car turns right, the left wheel accelerates, causing the car to make the turn faster. The idea is that the left wheel pushes the car during the turn, which increases the turn speed. Explained with example realistic values, if the left camera sees a right turn, i.e. if c1CurrentLocation = 74, then c1CurrentLocation – leftEdgeMacro = 74 – 44 = 30. This would then be multiplied by our kpMotorR, which we tested and tuned to work most effectively when it's a value of 3. The final result of the left motor's speed if its' original speed was 3500, would be 3500 + 90 = 3590, which means that the car's left wheel would spin faster for a right turn.

P/D of PID

The PID is especially important in controlling our car's turns. As the car's traveling speed varies, our car's servo turning speed also needs to be altered. To do this, we established our PID constants, Kp1, Kd1, KpL, and KpR, for variable servo speed and angle. (The kpMotorL and kpMotorR's are abbreviated to save space.) Note that we attempted to implement the integral control, but found the oscillation it caused to be too big of a disadvantage to utilize effectively.

While KpL and KpR changes the left and right motor speeds, Kp1 modifies the servo's angle or how much the car turns. Furthermore, we added the derivative constant, Kd1, to reduce the oscillation of our car, particularly on straightaways. During testing, we noticed that before we implemented the faster turns, the car would turn too slowly at high speeds, causing the car to run off the track. Adding the faster turns allowed our car to clear the track at higher speeds.

During the testing and tuning process, we were cautious about keeping our kpR and kpL's small for if the values are too large, the left and right wheels spins too fast while turning right or left, respectively, and thus cause the car to turn too much. Consequently, the wheel motor speeds are kept around its original speed prior to the turn, i.e. just a bit above dutyCycle1. Although from inspection, this speed modification may seem to have only a small effect, we saw an improvement of about 2 seconds in track completion time when we implemented these values. So, the results of this modification influenced our decision to keep it.

Linescan Camera Tuning

Set by our center of line functions in cal.c of the Appendix, the center of line pixels (the variables, leftEdgeMacro and rightEdgeMacro, for their respective cameras) were important for the car's turning accuracy. We noted that values of around 44 for the left camera and 78 for the right camera were the optimal setup for each linescan camera's range of 128 pixels. In other words, the 44th and 78th pixels of the respective cameras were the left and right "centers" used for determining when the car should turn. During the testing process, particularly after a crash, we noted that the center of line pixels tend to change due to a shift in camera angle or positioning caused by the car crashing into an object. To avoid constantly having to recompile and load the new center of line values, we utilized our Bluetooth innovation to change the center of line values wirelessly, which in turn allowed our testing to flow more smoothly.

In addition to the center of line parameters, we modified our voltage threshold and hill threshold values to adapt to the lighting conditions as well as the car's speed while traveling over a hill. For voltage threshold, we noticed that in well-lit conditions, a value of 45 would yield the best results, where if the camera's output exceeds 45, the car would treat the pixel as white, and if the output is less than 45, the car would treat the pixel as black. For hill threshold, we tuned this value to be around 34. When the feedback currents sum up to be less than the hill threshold, the descent of a hill is detected and the motor speed is set to 100, causing the car to brake and slow down when traveling down a hill.

Finally, the Bluetooth implementation used to set and tune our parameters is only one of the innovations to our design. Another unique concept we applied is the idea of having bursts of speed on straightaways for the track. To also tune these speed boosts for our car, we used separate parameters for how much speed to increase ("speedup"), how long the speed boost lasts ("interval"), and how much speed to decrease after a speed boost ("slowdown"). We added the slowdown feature to ensure that our car did not travel too fast when climbing down a hill, since the car's acceleration when descending a hill would cause it to run off the track. Combined with our separate motor speed parameters for a faster turn, and our Bluetooth innovation, this speed burst concept contributes yet another unique variation to our design.

III. Design and Performance Summary

The design overall was successful in completing the track in about 5ft/sec. We have concluded that with two linescan cameras, there is limited vision and poor coordination for control algorithm due to the instability of the cameras' positioning. With extensive time spent in tuning and discussion, we think going with one linescan camera or a 2D camera would be a good approach for the future, given that the carpet is black. We can also improve our speed PID control, as it is an essential part of the algorithm to level up our car's game. We are also looking forward to implement some kind of speed sensor or accelerometer to improve the PID control.

IV. Safety

In terms of safety, our team took standard precautions to ensure our car would not be a hazard. During the construction of car, we secured our battery using the provided zip-ties to hold it and added the crash-friendly, foam bumper onto the front of the car. In case our car crashed, the foam bumper provides sufficient coverage to cushion the impact of the accident. We also implemented in the algorithm that when the car sees all black, the car turns off its motor. Most of the time, our car stops when it runs off the track. We are the first to implement this approach.

V. Appendix

We, Ronald Lai, Patrick Yu and Shawn Xiang, have agreed that all contributed equally in this project and read the executive summary.

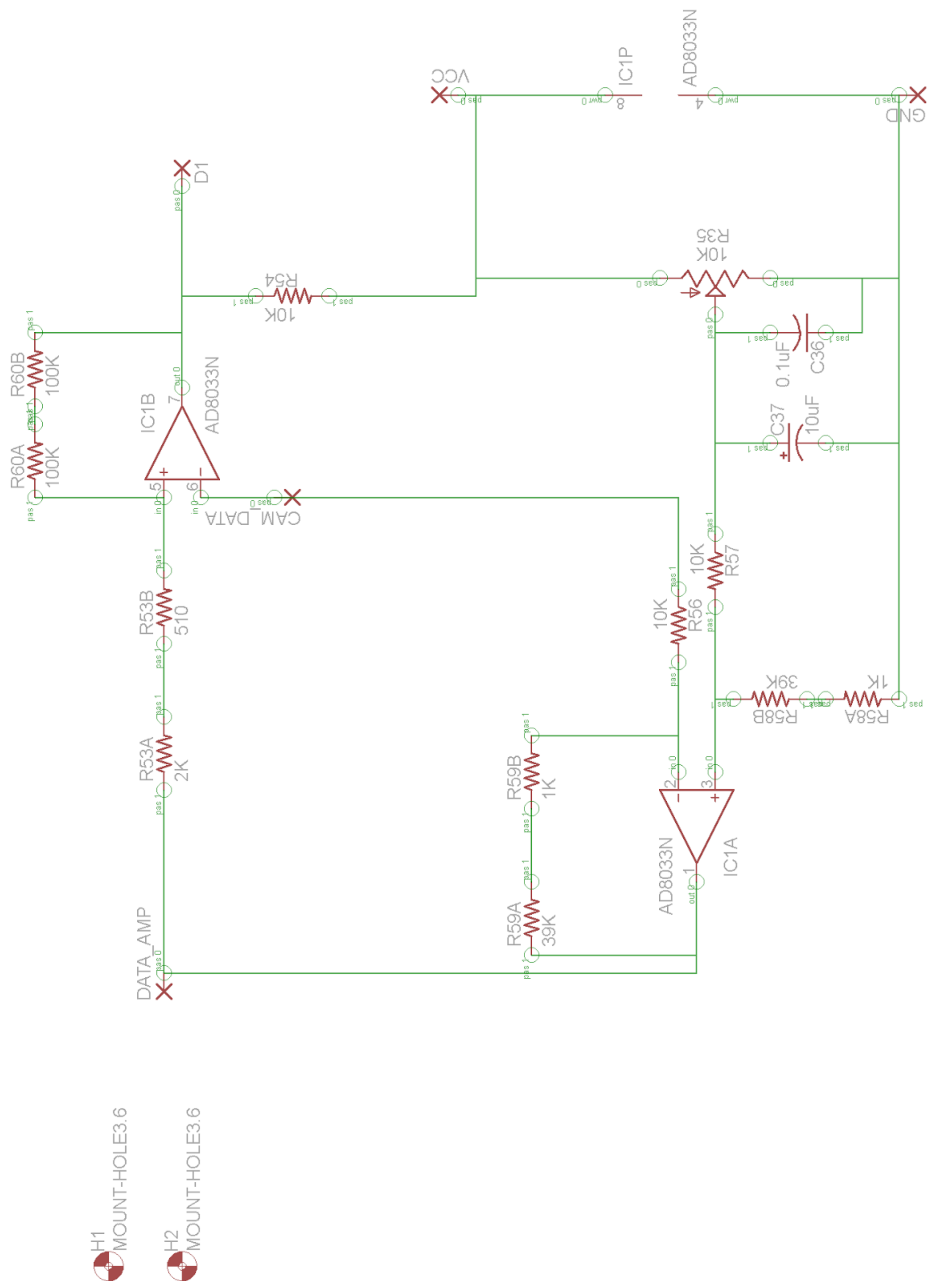
Signed by

Ronald Lai

Patrick Yu

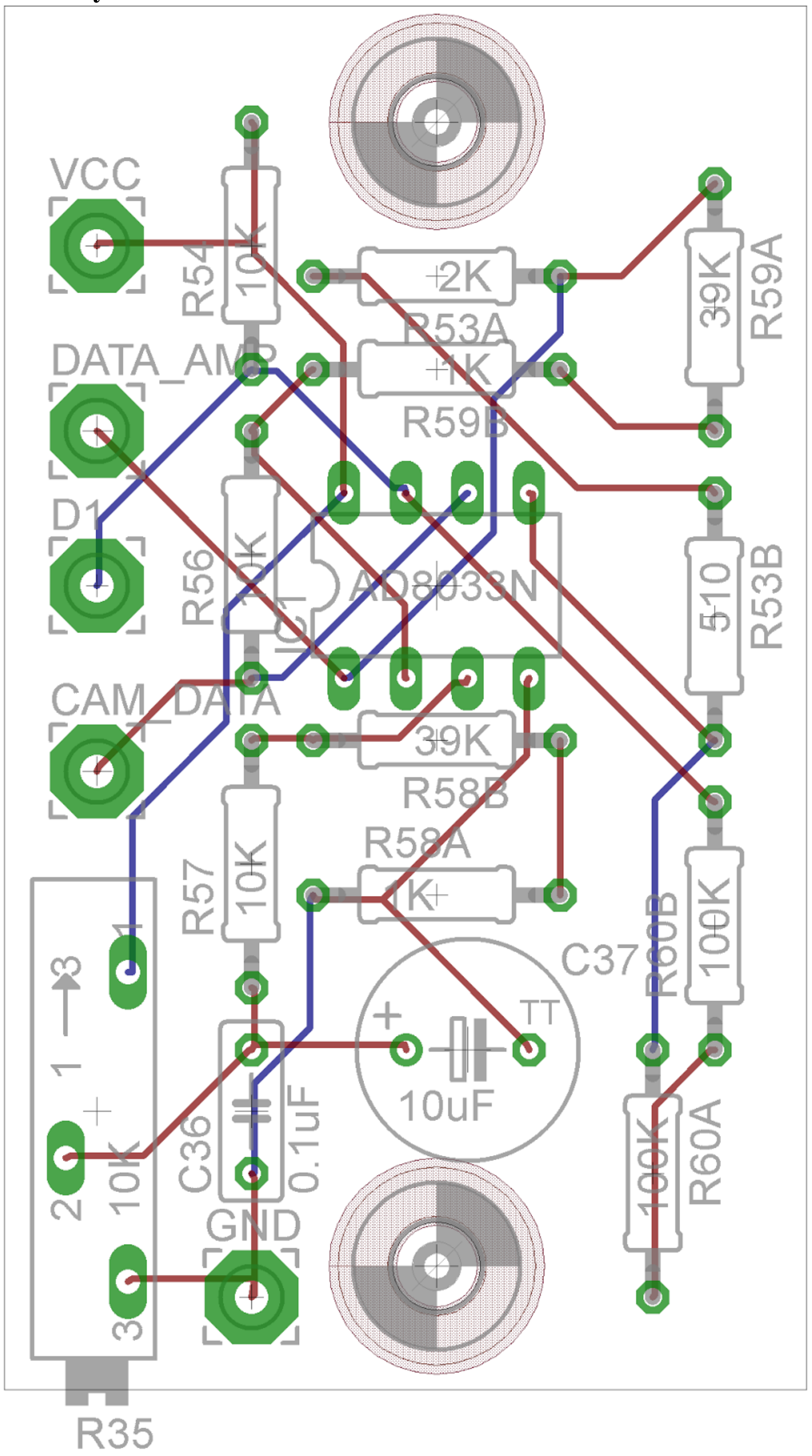
Shawn Xiang

Comparator Schematic



**AD8033N shown in the schematic is a placeholder for the AD8032BNZ
(All done with EAGLE PCB Design Software)

Comparator PCB Layout

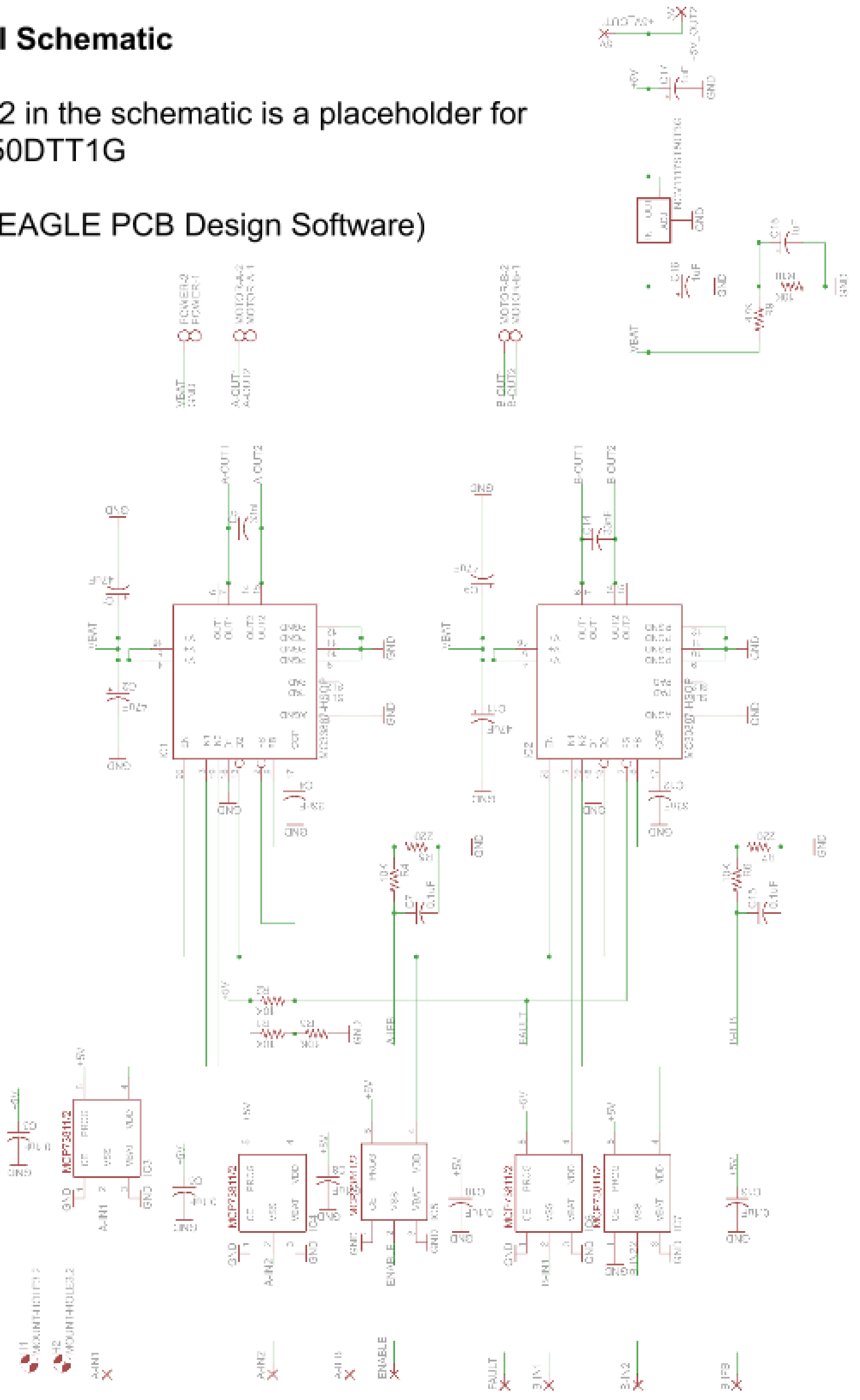


**AD8033N shown in the layout is a placeholder for the AD8032BNZ
Dimensions: 1.30" W × 2.25" L
(All done with EAGLE PCB Design Software)

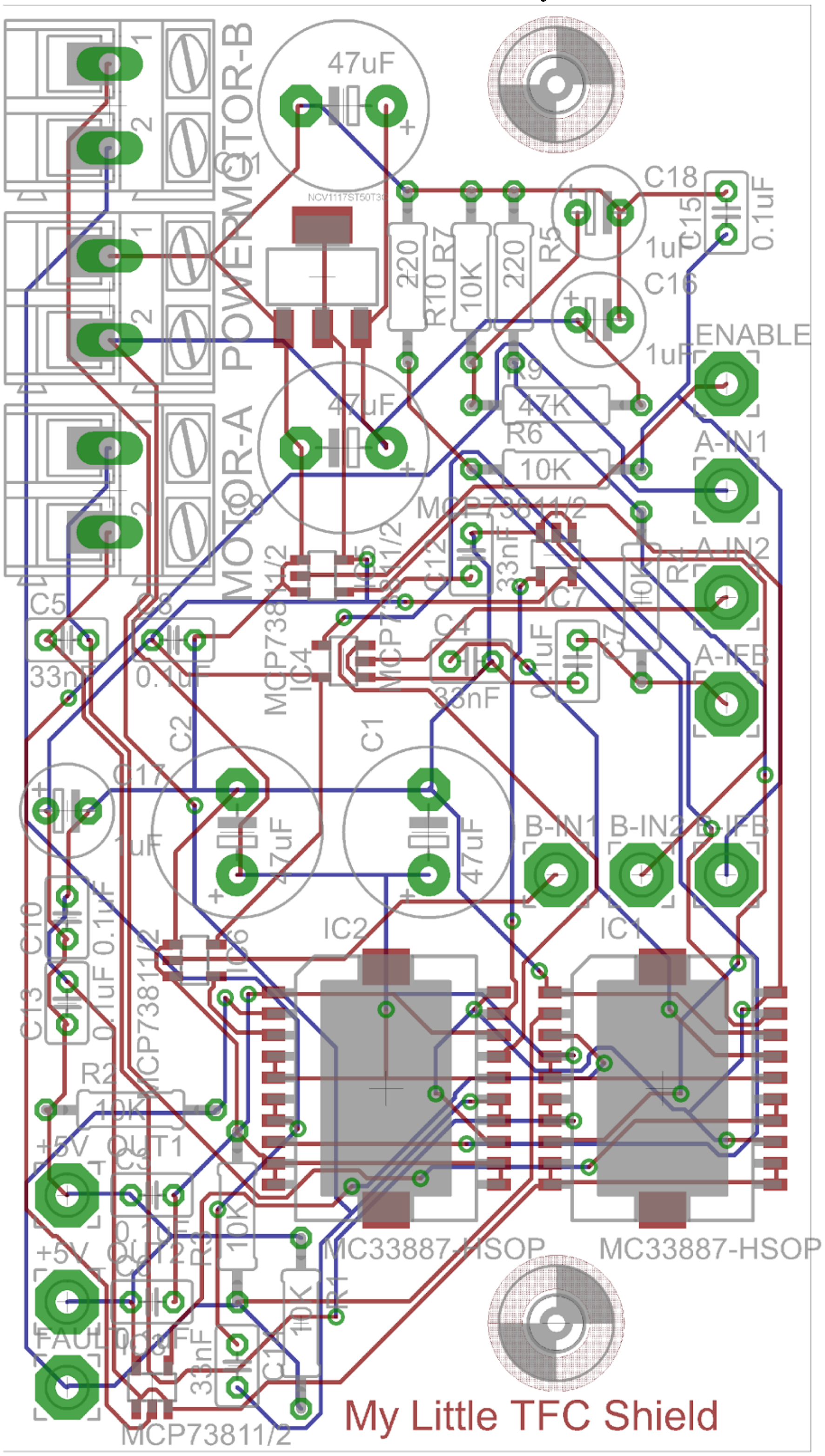
Motor Control Schematic

** MCP73811/2 in the schematic is a placeholder for M74VHC1GT50DTT1G

(All done with EAGLE PCB Design Software)



Motor Control PCB Layout



** MCP73811/2 in the layout is a placeholder for M74VHC1GT50DTT1G

Dimensions: 1.90” W × 3.40” L

(All done with EAGLE PCB Design Software)

cal.c

```
#include "MKL25Z4.h"
#include "main.h"

#include "main.h" /* include peripheral declarations */
#include <MKL25Z4.H>
#include "timers.h"
#include "adc16.h"
#include <stdlib.h>
#include <stdio.h>

int voltageThreshold (char p){

	//if(p < VTH){ // BLACK
	if(p < vthresh){ // BLACK
		p = '1';
	}
	else  // WHITE
		p = '-';
	return p;
}

int voltageThreshold2 (char p){

	//if(p < VTH){ // BLACK
	if(p < vthresh){ // BLACK
		p = 1;
	}
	else  // WHITE
		p = 0;
	return p;
}


// please note the base case and end case to avoid seg fault
// 1: PLUS, 2: MINUS
int slopeThreshold (int p, int q) {
	int slope;
	slope = (q - p)/2;

		//slope = abs(slope);

	return slope;
}


// please note the base case and end case to avoid seg fault
// 1: PLUS, 2: MINUS

/**************************************************************************/
/* Determine the center pixel of black line for Voltage Threshhold Scheme ()*/
/**************************************************************************/
int centerOfLine1 (void) {   // left camera
	int j;
	int lensEdge = 0;
  int leftBlackEdge = 0;
	int rightBlackEdge = 0;
	int possibleEdge = 0;
	int addUp = 0;
	int thirdCaseFlag = 0;
```

```
		c1ReverseFlag = 0;
		flag = 0;
		//char str0[8];
		for(j = 21; j < (BUFFER_SIZE - 20); j++){

			if (bufferPointer == 0) {
				addUp += pingBuffer2[j];

				// first case : black and then white
				if (pingBuffer2[j] == 1 && flag == 0 && lensEdge == 0 &&
c1ReverseFlag == 0) {
					flag = 1; // for debug purpose
				} else
				if (pingBuffer2[j] == 0 && flag == 1 && lensEdge == 0 &&
c1ReverseFlag == 0) {    //outside of track   EDIT: 1 to changed to 0
					lensEdge = 1;
					possibleEdge = j;
					c1ReverseFlag = 0;
					flag = 2; // for debug purpose
				}  else if (pingBuffer2[j] == 0 && flag == 1 && lensEdge == 1 &&
c1ReverseFlag == 0) {
					thirdCaseFlag = 1;
				}

				// second case: white and then black (the reverse case)
				if (pingBuffer2[j] == 0 && flag == 0 && lensEdge == 0 &&
c1ReverseFlag == 0) {
					flag = 3;
				}
		if (pingBuffer2[j] == 1 && flag == 3 && lensEdge == 0 && c1ReverseFlag == 0) {
//INTERSECTION REVERSE CASE
					flag = 4;
					possibleEdge = j;
					lensEdge = 1;
					c1ReverseFlag = 1;
				}

			} else {
				addUp += pongBuffer2[j];

				// first case : black then white
				if (pongBuffer2[j] == 1 && flag == 0 && lensEdge == 0 &&
c1ReverseFlag == 0) {
					flag = 5;
				} else
				if (pongBuffer2[j] == 0 && flag == 5 && lensEdge == 0 &&
c1ReverseFlag == 0) {   //outside of track   EDIT: 1 to changed to 0
					lensEdge = 1;
					possibleEdge = j;
					c1ReverseFlag = 0;
					flag = 6;
				}

				// second case : white then black
				if (pongBuffer2[j] == 0 && flag == 0 && lensEdge == 0 &&
c1ReverseFlag == 0) {
					flag = 7;
				} else
				if (pongBuffer2[j] == 1 && flag == 7 && c1ReverseFlag == 0) {   //
INTERSECTION REVERSE CASE
					lensEdge = 1;
```

```
                                possibleEdge = j;
                                c1ReverseFlag = 1;
                                flag = 8;
                        }
                        }
                }

         if(addUp == 0) {    // all white
                return -1;
         }
          else if (possibleEdge == 0) {  // all black
                                  return -2;
                              }
        else {

           return(possibleEdge);
        }
}

int c2centerOfLine1 (void) {    // right camera
        int j;
        int lensEdge = 0;
  int leftBlackEdge = 0;
        int rightBlackEdge = 0;
        int possibleEdge = 0;
        int addUp = 0;

        c2ReverseFlag = 0;
        flag = 0;

        for(j = 25; j < (BUFFER_SIZE - 18) ; j++){
                addUp += c2PingBuffer2[j];

                if (bufferPointer == 0) {

                        // first case: white and then black
                        if (c2PingBuffer2[j] == 0 && flag == 0 && lensEdge == 0 &&
c2ReverseFlag == 0) {
                                flag = 10;
                        } else
                        if (c2PingBuffer2[j] == 1 && flag == 10 && lensEdge == 0 &&
c2ReverseFlag == 0) { //edge of track - black
                                        possibleEdge = j;
                                  c2ReverseFlag = 0;
                                  lensEdge = 1;
                                  flag = 11;
                                        //uart0_putchar(leftBlackEdge + '0');
                                }

                                // second case : black and then white
                        if (c2PingBuffer2[j] == 1 && flag == 0 && lensEdge == 0 &&
c2ReverseFlag == 0) {
                                flag = 13;
                        } else
      if (c2PingBuffer2[j] == 0 && flag == 13 && lensEdge == 0 && c2ReverseFlag == 0)
{   //INTERSECTION REVERSE CASE
                                possibleEdge = j;
                                lensEdge = 1;
                                c2ReverseFlag = 1;
                                flag = 14;
                        }
```

```
            } else {
                   addUp += c2PongBuffer2[j];

                   // first case: white and then black
                   if (c2PongBuffer2[j] == 0 && flag == 0 && lensEdge == 0 &&
c2ReverseFlag == 0) {
                          flag = 15;
                   }
                   else if (c2PongBuffer2[j] == 1 && flag == 15 && lensEdge == 0 &&
c2ReverseFlag == 0) {
                                 possibleEdge = j;
                            c2ReverseFlag = 0;
                            lensEdge = 1;
                            flag = 16;
                                 //uart0_putchar(leftBlackEdge + '0');
                          }
                   else
                          // second case: black and then white
                   if (c2PongBuffer2[j] == 1 && flag == 0 && lensEdge == 0 &&
c2ReverseFlag == 0) {
                          flag = 17;
                   }
                          else
                   if (c2PongBuffer2[j] == 0 && flag == 17 && lensEdge == 0 &&
c2ReverseFlag == 0){  // REVERSE CASE
                          flag = 18;
                          c2ReverseFlag = 1;
                          lensEdge = 1;
                          possibleEdge = j;
                   }

            }
     }


     if(addUp == 0) {     // all white
            return -1;
     }
     else if (possibleEdge == 0) {  // all black
            return -2;
     }
  else if (c1ReverseFlag == 1) {
     return (possibleEdge);
     }            else {
     return(possibleEdge);
     }
}
```

main.c

```
int main (void) {
       int uart0_clk_khz;

       char key = ' ';
       char key1 = ' ';
       char key2 = ' ';
       char key3 = ' ';

       int temp = 0;
       char str[12];
       unsigned int res;
       unsigned int res2;
       unsigned int dutyCycle1;
       unsigned int dutyCycle2;

       //char str[] = "\r\nADC conversion\r\n h = voltage threshold;\r\n s = slope
threshold;\r\n r = raw data;\r\n other = load data\r\n";

       int i = 0;
       int k = 0;

       // record location
       int c1CurrentLocation = -4;
  int c1LocationDifference  = 0;

       int c2CurrentLocation = -4;
       int c2LocationDifference = 0;

       int averageDifference = -6;

  // location flag: 0 = MIDDLE/CENTER, 1 = RIGHT, -1 = LEFT
       int c1LocationFlag = 0;
       int c2LocationFlag = 0;

       int firstRound = 1;
       int oldestIndex = 0;
       int currentIndex = 1;

       int differenceSum = 0;  // used in integration
       int integrationCounter = 0;

       int integrationValue = 0;

       unsigned int fb1;
       unsigned int fb2;

       unsigned int previousFb;

       unsigned int previousDutyCycle1;

       unsigned int kp1 = 20;
       unsigned int kd1 = 10;
       unsigned int ki1 = 10;

       //unsigned int kp_motor = 1;
       unsigned int speed = 500;

       unsigned int leftEdgeMacro = 34;
       unsigned int rightEdgeMacro = 91;

       unsigned int deadbandWidth = 1;

       //unsigned int vth = 50;
```

```
        unsigned int turning = 0;

        int reverseCounter = 0;
        int reverseNumber = 0;

        char stopKey = ' ';

        int derivativePeriod;

        int divisor = 0;

        int quotient = 0; // count

        int quotient2 = 0;

        int intersectionFlag = 0;

        int intersectionValue = 0;

        int hillThreshold = 0;

        int straightd = 0;
        int upSpeed = 0;
        int slowDown = 0;

        unsigned int kpMotorL = 1;

        unsigned int kpMotorR = 1;
        /****************
        clocks
        ****************/

        SIM->SCGC5 |= (SIM_SCGC5_PORTA_MASK
                                                  | SIM_SCGC5_PORTB_MASK
                                                  | SIM_SCGC5_PORTC_MASK
                                                  | SIM_SCGC5_PORTD_MASK
                                                  | SIM_SCGC5_PORTE_MASK );

        SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK; // set PLLFLLSEL to select the PLL for this
clock source
        SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1); // select the PLLFLLCLK as UART0 clock source

         SystemCoreClockUpdate();

         SysTick_Config(48000000/10);       // Change the SysTick interrupt frequency to 100
Hz


        /******************
        GPIO initialzation
        ******************/

        // STEP 2:  change the analog input pin to ADC0_SE6b (PTD5)
 PORTD->PCR[5] = PORT_PCR_MUX(0x2);         // Enable the UART0_RX function on PTD5
 PORTD->PCR[6] = PORT_PCR_MUX(0x2);         // EDITED: Enable the UART0_RX function on PTD6
 PORTA->PCR[2] = PORT_PCR_MUX(0x2);         // Enable the UART0_TX function on PTA2
 PORTA->PCR[1] = PORT_PCR_MUX(0x2);   // on PTA1

 //PORTC->PCR[3] = PORT_PCR_MUX(0x2);   // on PTC3 , A, IN1
 //PORTC->PCR[1] = PORT_PCR_MUX(0x2);       // ON PTC1, B, IN1

        uart0_clk_khz = (48000000 / 1000); // UART0 clock frequency will equal half the PLL
frequency
        uart0_init (uart0_clk_khz, TERMINAL_BAUD);    // step 6
	PORTB->PCR[0] = (1UL << 8);
			  // Pin PTB0 is GPIO
	PORTB->PCR[1] = (1UL<<8);
	PORTB->PCR[2] = (1UL << 8);
		// POT2
	PORTB->PCR[3] = (1UL << 8);
		// POT1

	PORTD->PCR[7] = (1UL << 8);
	PORTE->PCR[1] = (1UL << 8);
	//PORTB->PCR[9] = (1UL << 8);
	// Assert a GPIO signal for measuring conversion time
	PORTB->PCR[4] = (1UL << 8);
	PORTE->PCR[21] = (1UL << 8);   //edited , enabling the GPIO

	PORTC->PCR[17] = (1UL << 8);  // SW 1
	PORTC->PCR[13] = (1UL << 8);  // SW 2

	 // H bridge enabling a in 2
	PORTC->PCR[4] = (1UL << 8);   //H-BRIDGE_A-IN2

	 // b
	PORTC->PCR[2] = (1UL << 8);   //H-BRIDGE_B-IN2


	  //feedback
	PORTE->PCR[22] = (1UL << 8);
	PORTE->PCR[23] = (1UL << 8);

	FPTD->PDOR |= 128;
	FPTD->PDOR |= 64;
	FPTE->PDOR |= 0;                           // ***EDITED, disable H bridge
		//A
	FPTC->PDOR &= ~(1UL << 4);  // as output H-BRIDGE_A-IN2
	  // b
	FPTC->PDOR &= ~(1UL << 2);

	 //FPTD->PDOR |= 32;
	 //    FPTD->PDDR |= 256;

	FPTE->PDOR |= 2;

	//FPTC->PDOR |= 1 << 1;
	//FPTC->PDOR |= 1 << 3;

	//FPTD->PDDR |= 1UL << 7;  // PTD 7
	FPTB->PDDR |= 1UL << 1 | 1UL << 4;  // PTB 1, 4
	//FPTE->PDDR |= 1UL << 1;             // PTE 1
	FPTD->PDDR |= 128;
FPTD->PDDR |= 64;
	FPTE->PDDR |= 2;
	FPTE->PDDR |= 1 << 21;   // set PTE21 as an output

	FPTD->PDDR |= 256;
	FPTB->PDDR |= 1; // configure PTB0 as output
	FPTB->PDOR |= 1;
						   // initialize PTB0
	FPTB->PDDR |= 1;

	//FPTC->PDDR |= 1 << 1;
	//FPTC->PDDR |= 1 << 3;

	FPTC->PDDR |= 0 << 17;                                                  // SW2
	FPTC->PDDR |= 0 << 13;                                                  // SW1

	FPTB->PDDR |= 0 << 2;                                            // POT 2
	FPTB->PDDR |= 0 << 3;                                                                        // POT 1

	// h BRIDGE
	FPTC->PDDR |= (1 << 4);                                                                            // in2
- A

	FPTC->PDDR |= (1 << 2);                                                                            // in2
- B

	//FPTB->PDDR = 1UL<<9;

	//feedback
	FPTE->PDDR &= ~(1UL << 22);                                                              //PTE22 as an
input
	FPTE->PDDR &= ~(1UL << 23);                                                              //PTE23 as an
input

	//put("Starting PIT\r\n");

	/*************
	starting interrupt and initialization
	**************/

	Init_PIT(8000);
                                                          // count-down period = 5000 us
	//Init_PIT(1000);

	Start_PIT();

  Init_PWM();
  Init_PWM1();

	Init_ADC();

	NVIC_SetPriority(ADC0_IRQn, 192); // 0, 64, 128 or 192
	NVIC_ClearPendingIRQ(ADC0_IRQn);
	NVIC_EnableIRQ(ADC0_IRQn);
	__enable_irq();


	 put("\r\n\r\n\r\n\r\n\r\n");

	 put("Please set the kp1 value:\r\n");

	 key = uart0_getchar();
	 key1 = uart0_getchar();

	kp1 = (key - 48 ) * 10 + (key1 - 48);

	sprintf(str, " %i\n\r", kp1);

	put(str);

	put("Please set the derivative value:\r\n");

	key = uart0_getchar();
	key1 = uart0_getchar();

	kd1 = (key - 48 ) * 10 + (key1 - 48);

	sprintf(str, " %i\n\r", kd1);

	put(str);

	 /*
	put("Please set the integration value:\r\n");
```

```
key = uart0_getchar();
key1 = uart0_getchar();

ki1 = (key - 48 ) * 10 + (key1 - 48);

sprintf(str, " %i\n\r", ki1);

put(str);
*/

 ki1 = 0;

 put("Please set the kp_motor value LEFT:\r\n");
 key = uart0_getchar();
 key1 = uart0_getchar();

//sprintf(str, "%c%c\n\r", key, key1);

       kpMotorL = (key - 48 ) * 10 + (key1 - 48);

       sprintf(str, "%i\n\r", kpMotorL);

 put(str);

 put("Please set the kp_motor value RIGHT:\r\n");
 key = uart0_getchar();
 key1 = uart0_getchar();

//sprintf(str, "%c%c\n\r", key, key1);

       kpMotorR = (key - 48 ) * 10 + (key1 - 48);

       sprintf(str, "%i\n\r", kpMotorR);

 put(str);


 put("Please set the speed:\r\n");
 key = uart0_getchar();
 key1 = uart0_getchar();
 key2 = uart0_getchar();
 key3 = uart0_getchar();
//sprintf(str, "%c%c\n\r", key, key1);

 speed = (key - 48 ) * 1000 + (key1 - 48)*100 + (key2 -48)*10 + (key3 - 48);

 sprintf(str, " %i\n\r", speed);

 put(str);

 put("Please set the left value:\r\n");
 key = uart0_getchar();
 key1 = uart0_getchar();

//sprintf(str, "%c%c\n\r", key, key1);

       leftEdgeMacro = (key - 48 ) * 10 + (key1 - 48);

       sprintf(str, " %i\n\r", leftEdgeMacro);

 put(str);

 put("Please set the right value:\r\n");
 key = uart0_getchar();
 key1 = uart0_getchar();
```

```
      rightEdgeMacro = (key - 48 ) * 10 + (key1 - 48);

      sprintf(str, " %i\n\r", rightEdgeMacro);

put(str);

put("Please set the turning:\r\n");
key = uart0_getchar();
key1 = uart0_getchar();
key2 = uart0_getchar();
key3 = uart0_getchar();

turning = (key - 48 ) * 1000 + (key1 - 48)*100 + (key2 -48)*10 + (key3 - 48);

      sprintf(str, " %i\n\r", turning);

 put(str);
/*
 put("Please set the deadband width value:\r\n");
      key = uart0_getchar();
      key1 = uart0_getchar();

      deadbandWidth = (key - 48 ) * 10 + (key1 - 48);

      sprintf(str, "%i\n\r", deadbandWidth);
      */

      deadbandWidth = 0;
      put(str);

      put("Please set the voltage threshold:\r\n");
      key = uart0_getchar();
      key1 = uart0_getchar();

      vthresh = (key - 48 ) * 10 + (key1 - 48);

      sprintf(str, "%i\n\r", vthresh);

      put(str);

      /*
      put("Please set time period:\r\n");
      key = uart0_getchar();
      key1 = uart0_getchar();

      derivativePeriod = (key - 48 ) * 10 + (key1 - 48);

      sprintf(str, "%i\n\r", derivativePeriod);

      put(str);
      */
      derivativePeriod = 1;

      put("Hill threshold:\r\n");
      key = uart0_getchar();
      key1 = uart0_getchar();

      hillThreshold = (key - 48 ) * 10 + (key1 - 48);

      sprintf(str, "%i\n\r", hillThreshold);

      put(str);

      put("StraighD:\r\n");
      key = uart0_getchar();
      key1 = uart0_getchar();
      key2 = uart0_getchar();

      straightd = (key - 48 ) * 100 + (key1 - 48) * 10 + (key2 - 48);

      sprintf(str, "%i\n\r", straightd);

      put(str);

      put("Speed up interval:\r\n");
      key = uart0_getchar();
      key1 = uart0_getchar();
      key2 = uart0_getchar();

      upSpeed = (key - 48 ) * 100 + (key1 - 48) * 10 + (key2 - 48);

      sprintf(str, "%i\n\r", upSpeed);

      put(str);

      put("Slow Down rate:\r\n");
      key = uart0_getchar();
      key1 = uart0_getchar();

      slowDown = (key - 48 ) * 10 + (key1 - 48);

      sprintf(str, "%i\n\r", slowDown);

      put(str);

currentIndex = RING_BUFFER_SIZE - 1;

if (derivativePeriod > 50) {
       derivativePeriod = 50;
}

      put("Please turn on the power supply and then press SW2 on the shield\r\n");

      while (temp != (1 << 17)){   //sw2 pressed
              temp = FPTC->PDIR;
        //sprintf(str, "%i\r\n", temp);
             //put(str);
             //temp = temp & (1 << 17);
}
      FPTE->PDOR |= 1 << 21; // enable H bridge
      //put("Please set duty cycles using POT1 and POT2\r\n");

      temp = FPTC->PDIR;
      //sprintf(str, "%i\n\r", temp);
      //put(str);

 /************************
       motor
       Reading POT -> load to the register
       ************************/

// Reading POT value
  res = Read_ADC();  // pot 1
// sprintf(str, "READ1: %i\n\r", res);
//    put(str);

       res2 = Read_ADC2(); // pot 2
      // sprintf(str, "READ2: %i\n\r", res2);
 //    put(str);
```

```
            dutyCycle1 = (res*MODVAL)>>8;
            dutyCycle2 = (res2*MODVAL)>>8;

            dutyCycle1 = speed;

             PW1 = dutyCycle1;
                  PW2 = dutyCycle1;
                  previousDutyCycle1 = dutyCycle1;


                  TPM0->CONTROLS[0].CnV = dutyCycle1;  // ******speed********



                  TPM0->CONTROLS[2].CnV = dutyCycle1;  //*******speed********

      stopKey = ' ';

      while (1) {

            temp = FPTC->PDIR;

              if  (doneFlag == 1) {

            /***********
                         Feedback current
            ************/

             ADC0 -> CFG2 &= ~(1UL << 4);
             fb1 = Read_ADC_Feedback(7);
             ADC0 -> CFG2 &= ~(1UL << 4);
             fb2 = Read_ADC_Feedback(3);


                         if (hillFlag == 1){
                                dutyCycle1 = 100;
                                put("***hillFlag detected***");
                         }
                         if( fb1 > 1 && fb2 > 1)
                         {
                          if ((fb1 + fb2) < hillThreshold){


                                hillFlag = 1;
                                //ticks = 0;
                         }
                  }
                         if (hillFlag == 0) {
                                dutyCycle1 = previousDutyCycle1;
                                previousFb = fb1 + fb2;
                         }

                         sprintf(str, "1: %i. FB: %i, tick:%i, flag: %i, fbsum: %i,
rightside: %i\r\n", dutyCycle1, fb1, ticks, hillFlag, fb1 + fb2, previousFb - FBTHRESHOLD );
                         // put(str);
                         sprintf(str, "2: %i, FB: %i\r\n", dutyCycle2, fb2 );
                         // put(str);

                         /**********************
                         FIRST CAMERA
                         **********************/
       put("FI");

                         for (i = 20; i < (BUFFER_SIZE - 20) ; i++) {
                         //for (i = 0; i < (BUFFER_SIZE) ; i++) {
                         if (bufferPointer == 0) {
```

```
                pingBuffer2[i] = voltageThreshold2(buffer[i]);
        //      sprintf(str, "%c", voltageThreshold(buffer[i]));

        //      put(str);

        }
        else {
                pongBuffer2[i] = voltageThreshold2(pongBuffer[i]);
        //      sprintf(str, "%c", voltageThreshold(pongBuffer[i]));

        //      put(str);

        }
    }

    // process data
    c1CurrentLocation = centerOfLine1();
    sprintf(str, " (%3i) ", c1CurrentLocation);
    put(str);

    sprintf(str, " flag %i ", flag);
    put(str);

    put("|||");


    /********************
    SECOND CAMERA
    *******************/

    put("SE");
    //  reading data
    for (k = 25; k < (BUFFER_SIZE - 18); k++){
    //for (k = 0; k < (BUFFER_SIZE); k++){

        if (bufferPointer == 0) {

                c2PingBuffer2[k] = voltageThreshold2(c2Buffer[k]);
    //          sprintf(str, "%c", voltageThreshold(c2Buffer[k]));

    //          put(str);


        }
        else {

                c2PongBuffer2[k] = voltageThreshold2(c2PongBuffer[k]);
                //sprintf(str, "%c", voltageThreshold(c2PongBuffer[k]));

                //put(str);
        }
    }

    // process data
    c2CurrentLocation = c2centerOfLine1();
    sprintf(str, " R(%3i) ", c2centerOfLine1());
    put(str);

    sprintf(str, " flag %i ", flag);
    put(str);

/**********
    derivative

    derivativePeriod is the time (denominator
                        LocationDifference is the numertor term, aka difference
                        ***********/
                if(firstRound){
                        if(oldestIndex == derivativePeriod-1)
                                        firstRound = 0;
                        oldestIndex+=1;
                }
                else{
                        if(oldestIndex == 2){
                                c1LocationDifference = c1CurrentLocation + 3 * c1RingBuffer[0] - 
3 * c1RingBuffer[1] - c1RingBuffer[oldestIndex];
                                c2LocationDifference = c2CurrentLocation + 3 * c2RingBuffer[0] - 
3 * c2RingBuffer[1] - c2RingBuffer[oldestIndex];
                        }
                        else if(oldestIndex == 3){
                                c1LocationDifference = c1CurrentLocation + 3 * c1RingBuffer[1] - 
3 * c1RingBuffer[2] - c1RingBuffer[oldestIndex];
                                c2LocationDifference = c2CurrentLocation + 3 * c2RingBuffer[1] - 
3 * c2RingBuffer[2] - c2RingBuffer[oldestIndex];
                        }
                        else{
                                c1LocationDifference = c1CurrentLocation + 3 * 
c1RingBuffer[oldestIndex + 2] - 3 * c1RingBuffer[oldestIndex + 1]

                - c1RingBuffer[oldestIndex];
                                c2LocationDifference = c2CurrentLocation + 3 * 
c2RingBuffer[oldestIndex + 2] - 3 * c2RingBuffer[oldestIndex + 1]

                - c2RingBuffer[oldestIndex];
                        }
                }
                oldestIndex +=1;
                c1RingBuffer[currentIndex] = c1CurrentLocation;
                c2RingBuffer[currentIndex++] = c2CurrentLocation;

                quotient = 0;
                quotient2 = 0;
                divisor = 6 * derivativePeriod;
                if(c1LocationDifference > 0){
                        while(c1LocationDifference > 0){
                                if(c1LocationDifference > divisor)
                                        quotient+=1;
                                c1LocationDifference -= divisor;   // dividend: 
c1LocationDifference, divisor: 6T
                        }
                }

                if(c2LocationDifference > 0){
                        while(c2LocationDifference > 0){
                                if(c2LocationDifference > divisor)
                                        quotient2+=1;
                                c2LocationDifference -= divisor;   // dividend: 
c1LocationDifference, divisor: 6T
                        }
                }
                if(c1LocationDifference < 0){
                        while(c1LocationDifference < 0){
                                if(c1LocationDifference < -1*divisor)
                                        quotient -=1;
                                c1LocationDifference += divisor;   // dividend: 
c1LocationDifference, divisor: 6T
                        }
                }
                quotient2 = 0;
                if(c2LocationDifference < 0){
                        while(c2LocationDifference < 0){
                              if(c2LocationDifference < -1*divisor)
                                     quotient2-=1;
                              c2LocationDifference += divisor;   // dividend:
c1LocationDifference, divisor: 6T
                       }
               }
               if(c1LocationDifference == 0){
                      quotient = 0;
               }
               if(c2LocationDifference == 0){
                      quotient2 = 0;
               }


               if(oldestIndex > (derivativePeriod - 1)){
                      oldestIndex = 0;
               }
               if(currentIndex > (derivativePeriod - 1)){
                      currentIndex = 0;
               }

               if (c1LocationDifference > 0){  // turned right; say it was 20, now it is 30;
the dark line turned to right
                              c1LocationFlag = 1;
               }
                      else if (c1LocationDifference < 0) { // left
                        c1LocationFlag = -1;
               }
                      else {
                              c1LocationFlag = 0;
                      }

               if (c2LocationDifference > 0){  // right
                              c2LocationFlag = 1;
               }
                      else if (c2LocationDifference < 0) {  // left
                              c2LocationFlag = -1;
                      }
                      else {
                              c2LocationFlag = 0;
                      }

                      if ((c1LocationDifference < -2 && c2LocationDifference < -2) ||
(c1LocationDifference > 2 && c2LocationDifference > 2 )){
                              //deadband
                      averageDifference += (abs(c1LocationDifference) +
abs(c2LocationDifference)) >> 1;
                              if (c1LocationFlag == -1 && c2LocationFlag == -1)
                      averageDifference *= -1; // assume left has a negative sign
                      } else {
                              averageDifference = 0;
                      }

                      /*************
                      integration
                      Summing up all the difference; it will fluctuate based on the derivative
sum
                      **************/
                      if (integrationCounter < 10) {
                      differenceSum += (c1LocationDifference + c2LocationDifference) >> 1;
//adding up the difference
                      integrationCounter ++;
                      } else {
                              integrationCounter = 0;
                        integrationValue = differenceSum;
                 }
```

```
                    /***********************
                    ran off the track, all black
                    ************************/
                    if (c1CurrentLocation == -2 && c2CurrentLocation == -2) {  // all black
                           TPM0->CONTROLS[2].CnV = 37;
                           TPM0->CONTROLS[0].CnV = 37;  // turn off the motor
                    }


                    // -1 = white, -2 = black

                    /***************
                    proportional and special cases
                    ***************/

                           if (c1CurrentLocation == -1 && c2CurrentLocation == -1) {  //
intersection, white and white
                                  PW = 4500;
                                  PW1 = dutyCycle1;     // left motor original speed
                                  PW2 = dutyCycle1;            // right motor original speed
                                  put("case 1 ");

                                  if (intersectionFlag == 0) {
                                         intersectionFlag = 1;
                                  }
                           }
                           else if (c1CurrentLocation == -1 && c2CurrentLocation == -2) {
// white and black
                                  //PW = 3500; //left turn
                                  PW = 4500 - turning;

                                  put("case 2 ");
                           }
                           else if (c1CurrentLocation == -2 && c2CurrentLocation == -1) {
// black and white
                                  //PW = 5500; //right turn
                                  PW = 4500 + turning;
                                  put("case 3 ");
                           }
                           else if (c1CurrentLocation > 0 && c2CurrentLocation == -1){  //
left is valid, right is white
                                  PW = 4500 + (c1CurrentLocation - leftEdgeMacro)*kp1;
                                  //PW = 4500 + (c1CurrentLocation - 0)*kp1;
                                  //PW1 = dutyCycle1 - (c1CurrentLocation -
leftEdgeMacro)*kpMotorL;
                                  //PW2 = dutyCycle1 + (c1CurrentLocation -
leftEdgeMacro)*kpMotorR;
                                  if (PW > 4500) {
                                         PW1 = dutyCycle1 + (c1CurrentLocation -
leftEdgeMacro)*kpMotorL;
                                  }
                                  else if(PW < 4500){
                                         PW2 = dutyCycle1 - (c1CurrentLocation -
leftEdgeMacro)*kpMotorR;
                                  }
                                  put("case 4 ");
                           }
                           else if (c1CurrentLocation == -1 && c2CurrentLocation > 0) {   //
left is white, right is valid
                                  PW = 4500 + (c2CurrentLocation - rightEdgeMacro)*kp1;
                                  //PW = 4500 + (c2CurrentLocation - 128)*kp1;
                                  //PW1 = dutyCycle1 - (c2CurrentLocation -
rightEdgeMacro)*kp_motor;
                                  //PW2 = dutyCycle1 + (c2CurrentLocation -
rightEdgeMacro)*kp_motor;
                                        if(PW > 4500){
                                                PW1 = dutyCycle1 + (c2CurrentLocation - 
rightEdgeMacro)*kpMotorR;
                                        }
                                        else if(PW < 4500){
                                                PW2 = dutyCycle1 + (c2CurrentLocation - 
rightEdgeMacro)*kpMotorL;
                                        }
                                        put("case 5 ");
                                }
                                else if (c1CurrentLocation == -2 && c2CurrentLocation > 0) {  // 
left is black, right is valid
                                        PW = 4500 + (c2CurrentLocation - rightEdgeMacro)*kp1;
                                        if(PW > 4500){
                                                PW1 = dutyCycle1 + (c2CurrentLocation - 
rightEdgeMacro)*kpMotorR;
                                        }
                                        else if(PW < 4500){
                                                PW2 = dutyCycle1 + (c2CurrentLocation - 
rightEdgeMacro)*kpMotorL;
                                        }
                                        //PW1 = dutyCycle1 - (c2CurrentLocation - 
rightEdgeMacro)*kp_motor;
                                        //PW2 = dutyCycle1 + (c2CurrentLocation - 
rightEdgeMacro)*kp_motor;
                                        put("case 6 ");
                                }
                                else if (c1CurrentLocation > 0 && c2CurrentLocation == -2) {   // 
right is black, left is valid
                                        PW = 4500 + (c1CurrentLocation - leftEdgeMacro)*kp1;
                                        //PW1 = dutyCycle1 + (c1CurrentLocation - 
leftEdgeMacro)*kp_motor;
                                        //PW2 = dutyCycle1 - (c1CurrentLocation - 
leftEdgeMacro)*kp_motor;
                                        if(PW > 4500){
                                                PW1 = dutyCycle1 + (c1CurrentLocation - 
leftEdgeMacro)*kpMotorR;
                                        }
                                        else if(PW < 4500){
                                                PW2 = dutyCycle1 + (c1CurrentLocation - 
leftEdgeMacro)*kpMotorL;
                                        }
                                        put("case 7 ");
                                }
                                else {
                                        //if (abs(c1CurrentLocation - leftEdgeMacro) > 
deadbandWidth || abs(c2CurrentLocation - rightEdgeMacro) > deadbandWidth)
                                        PW = 4500 + ((((c1CurrentLocation - leftEdgeMacro)*kp1) + 
((c2CurrentLocation - rightEdgeMacro)*kp1))>>1);
                                        PW = PW + ((kd1*quotient + kd1*quotient2) >> 1 ) + 
ki1*integrationValue;
                                        if(PW > 4500){
                                                PW1 = dutyCycle1 + (PW-4500)*kpMotorL;
                                        }
                                        else if(PW < 4500){
                                                PW2 = dutyCycle1 + (4500-PW)*kpMotorR;
                                        }
                                        put("case 8 ");
                                }


                                if (c2ReverseFlag == 1 && c1CurrentLocation == -2) {  // left is 
black, right is reverse
                                        //if(reverseCounter != 0)
                                PW = 4500 + kp1 * (rightEdgeMacro - c2CurrentLocation);
```

```
                              put ("REVERSE CASE 1");
                                     //reverseCounter ++;
                              }

                              if (c1ReverseFlag == 1 && c2CurrentLocation == -2) {  // right is
black, left is reverse
                                     //if (reverseCounter != 0)
                              PW = 4500 + kp1 * (leftEdgeMacro - c1CurrentLocation);
                              put ("REVERSE CASE 2");
                                     //reverseCounter ++;
                              }

                              if (c1ReverseFlag == 1 && c2CurrentLocation != -2 &&
c2CurrentLocation != -1) {  // right is reverse, left is valid
                                     PW = 4500 + kp1 * (leftEdgeMacro - c1CurrentLocation);
                                     put ("REVERSE CASE 3");
                              }

                              if (c2ReverseFlag == 1 && c1CurrentLocation != -2 &&
c1CurrentLocation != -1) {  // left is reverse, right is valid
                                     PW = 4500 + kp1 * (rightEdgeMacro - c2CurrentLocation);
                                     put ("REVERSE CASE 4");
                              }


                              if (c1ReverseFlag != 0 && c2ReverseFlag != 0) {
                                     reverseCounter = 0;
                              }

                              if (averageDifference > 2) {
                              //      PW = PW + averageDifference * 5;
                              }

                       if (abs(PW - 4500) <= straightd && PW1 < 9600 && PW2 < 9600 &&
(!hillFlag)){
                              PW1 += upSpeed;
                              PW2 += upSpeed;
                       }
                       else{
                              if(PW1 > dutyCycle1 && PW2 > dutyCycle1){
                                     PW1 -= slowDown*upSpeed;
                                     PW2 -= slowDown*upSpeed;
                                     /*
                                     if (PW1 < 3400 || PW2 < 3400) {  // ***changes
                                            PW1 = 3400;
                                            PW2 = 3400;
                                     }
                                     */
                              }
                              else{
                                     PW1 = dutyCycle1;
                                     PW2 = dutyCycle1;
                              }
                       }

                       if (PW > 6000){
                                     PW = 6000;
                       }
                       else if (PW < 3000) {
                                     PW = 3000;
                       }

                       if (PW1 > 9600){
                              PW1 = 9600;
                       }
                       else if (PW1 < 1) {
```

```
                                PW1 = 1;
                }

                if (PW2 > 9600){
                        PW2 = 9600;
                }
                else if (PW2 < 1) {
                                PW2 = 1;
                }


                        TPM1->CONTROLS[0].CnV = PW;

                TPM0->CONTROLS[0].CnV = PW1;     // ***change*** Assume it is the left
           TPM0->CONTROLS[2].CnV = PW2;

                sprintf(str, "%i" , flag);
                put(str);

                sprintf(str, "PW (%i)", PW-4500);
                put(str);

                //sprintf(str, "c1: %i", c1CurrentLocation);
                //put(str);

                //sprintf(str, "c2: %i", c2CurrentLocation);
                //put(str);

                sprintf(str, " c1D %i", c1LocationDifference);
                put(str);

                sprintf(str, " c2D %i", c2LocationDifference);
                put(str);

                sprintf(str, " Q %i", quotient*kd1);
                put(str);

                put("\r\n");


                bufferPointer = !bufferPointer;

        } //doneFlag

    } // while

    //TPM0->CONTROLS[2].CnV = 37;
    //TPM0->CONTROLS[0].CnV = 37;  // turn off the motor

}  //main
```