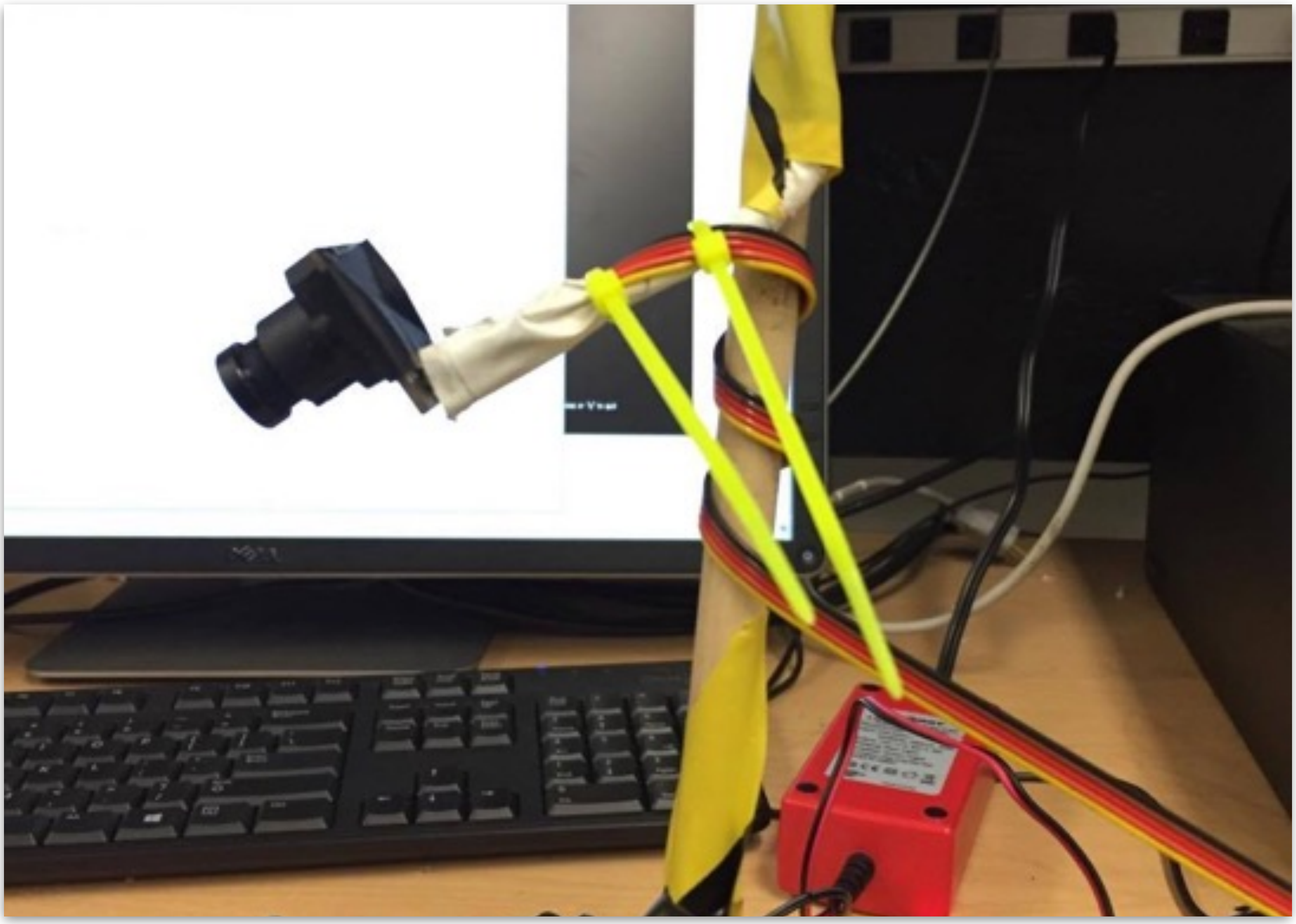
# NATCAR Final Report

## Team Vivake and Bake

Karun Dhillon, Ronald Diaz, Nicholas Jens - EEC 195



## Executive Summary

Like most other groups in EEC 195, our autonomous car uses a linescan camera to follow the track rather than an electro-magnetic sensor. The data from the camera is then processed by our code and sent to the servo and motor controls via the PCB we designed for ourselves and the FREEDM board's microcontroller. The controls register the camera data and change the speed of the motors, and in turn, the wheels. The cycle is then repeated as the car moves and  the camera begins to input a new set of data from the next portion of the track. The car performs well based on the standards of the class, but needs to be worked on in order to perform admirably in the upcoming Spring Quarter competitions.

# Table of Contents

# Technical Reports

## Camera[1]

The linescan camera is the central part to the structure of our automated car. Precise camera data ensures that the steering algorithm will work correctly. It is safe to assume that the steering algorithm works perfectly because every part of it is controllable by manipulating the source code. However, this is not the case with the camera because it is the only part in the entire system with an input from the real world, so there is bound to be error.

In order to configure the camera to the microcontroller, certain signals must be sequenced in a particular order to begin reading in data. The SI signal should first be set to high to begin reading in a line from the linescan camera. While SI is high, the clock signal should begin its first pulse. SI should go low before clock oscillates back to low. Each clock pulse is a pixel of data received, and once it receives 128 pixels, the next SI pulse begins signaling the retrieval of the next line of 128 pixels.

The microcontroller cannot use the fill up a line buffer and use that data set to control the motor and steering at the same time. To solve this problem a ping pong buffer is set up. In other words, while one buffer is being filled up with new data, the other is being used to control the steering and motor. If derivative error control was to be implemented, this particular buffer being filled will be sent to a queue of other buffers, or the error of that particular buffer would be sent to a queue in order for derivative control to work.

[1] by Nick Jens

Once configuration is complete, the microcontroller receives a signal from the linescan camera as an input. This pulse is a higher voltage where it sees white and a lower voltage where it sees darker colors. This means a NATCAR track that has black carpet and white tape will send a signal with a sharp and skinny plateau on the pixels where the white line is.

The PIT (periodic interrupt timer) would control the amount of time that light could enter the camera per each line of data by changing the number its initialization function was set to. The minimum amount was 0.5 ms while the maximum was 10 ms. The more time that was given for light to enter, the greater the voltage difference was between white and black pixels. Right around the middle at 6 ms was the sweet spot for most lighting in most rooms.

After receiving camera data in a buffer, that 128 pixel array is transformed into only zeroes and ones, zero for black and one for white. Five pixels from each side were deemed as unneeded data and were set as arbitrary values that did not correspond to black or white. Determining if a pixel is white or black is done with two possible methods. One is with a dynamic voltage threshold, and the other is with a slope detection method. The dynamic voltage threshold would set a pixel to white if its value was higher than its calculated threshold, and black if it is below. The threshold would be calculated using the averages of each buffer when it is read it. To avoid the problem of setting a threshold that would mess up where the lighting in unfavorable but it should be seeing all white, a threshold was only set correctly when its peak to peak value was over some limit. The slope detection method would see where there was a sharp increase or decrease in the values of the buffer. It would use this info to set its black and white pixels. This method was unfavorable however and the dynamic voltage threshold method seemed to work a lot better.

To get around unreliable camera data, artificial camera data was set in certain cases to ensure that the car would continue to run in a desired direction when its data is uncertain. For NATCAR, it would fill the outer pixels with black once it detects the white line in the middle. For Freescale Cup, it would fill the inner pixels with white pixels once the edge of the track on either side has been detected.

After a buffer's data set has been converted into only zeroes and ones for black and white, this data set can now be used to calculate the degree of error of which the camera is sensing. For NATCAR detecting this error was very easy because the middle of the track is always given by the white line. However, because of the design of the Freescale Cup's course, and the fact that the placement and angle of the linescan camera made it incapable of seeing both edges of the track at all times. This made it so the error had to based off of how far the edges of the track come inward. This caused some problems when coding the algorithm for steering in the Freescale Cup.

There were a few ideas that could have been implemented involving cameras to improve the overall performance of the car. One would have been to get a better structural angle in order to see both edges of the track at all times. Another idea would be to use two separate linescan cameras to do the same trick. The last idea that could be used would be to implement one camera to look closely at the track and the other to look a lot further down the track. This could be used to improve straightaway speed performance.

## Servo Control[2]

The car runs based on the data recorded by the TSL1401-DB linescan camera. It is only one-dimensional, the lens being a sensor array of photodiodes 128 pixels across, which allows for faster data processing than if a two-dimensional camera or electro-magnetic sensor were used. This is important because the camera is the "eye" of the car, and it needs to pass this information along quickly as the track in front of it changes.

The data from the camera is passed along to the steering control. This steering control works as a function of the buffer[128] in the code. This steering control in turn runs the servo and motor controls using a form of proportional control. Proportional control works as a function of camera error, that is, it changes the speed of the car proportional to the data input to the system from the camera. The error is calculated differently in the two competitions.

The servo is run by pulse width modulations (PWMs), which is implemented into the system by the KL25Z processor's on-chip TPM (Timer/PWM Module). The TPM works by counting towards a certain MOD value, which, when met, is passed along to the onboard clock. This sets the overflow mask and the pulse changes for that amount of time. The higher the channel value, the longer the TPM pulse. The minimum pulse width is 1.0 ms, which corresponds to TPM channel value of 3000 and causes the car to turn left. Right turns occur at the maximum pulse width of 2.0 ms, which corresponds to a channel value of 6000.

[2] by Karun Dhillon

To initialize the PWM in the code, we assigned the PTBO (servo) to TPM1's Channel 0 (TPM1.CH0). We set the period by changing the TPM1.MOD value to get the frequency we desired. The servo's pulse width is then set by changing the TPM channel value.

## Motor Control[3]

The motor is controlled similarly to the servo, except that it uses two TPMs instead of just one on the FREEDM board's microcontroller. The additional TPM lets us control both the left and right motors independent of each other. The motor control PWM is also unipolar, such that it only drives one side of the H-bridge. The other side is driven by a GPIO logic output signal. The duty cycles of these pulse widths are controlled  by the potentiometers on either the TFC shield or our own PCB depending on the competition. The Freedom KL25Z's on-chip analog-digital converter (ADC) converts the analog POT values into 8-bit digital values in order to change the TPM channel values.

The motors are controlled by code similar to that of the servo control, but with two notable differences in the variables. First, the motors operate with TPM0, different than the TPM1 used by the servo control. TPM0_CH0 controlled the left motor, while TPM0_CH2 controlled the right. Second, the motors are based off the variable *mph*'s value, which was the base speed  off which everything else was calculated.

This speed variable, *mph*, changed proportionally based on the camera error, that is, the error of the car in terms of its location relative to the track. When the camera detected a straight away, the car would

[3] by Karun Dhillon

remain on the right path. There was relatively little error detected by the camera on this path, and the channel value would remain at approximately 4500 ± 120. The motor's speed would then increase, and the wheel speed would then increase from 110% to 130% of its potentiometer value. If the error caused the channel value to change in the range of ±120-450, the wheel speed would remain at the same value as that set by the potentiometer. This generally corresponded with a gentle turn to the left or the right. A sharp turn would correspond to a channel value change greater than ±450. In this case, the inner wheel would slow down to 20% to 40% of its POT value, while the outer wheel would spin anywhere from 80% to 120% of its POT value.

## PCB Design[4]

The design of the custom printed circuit board that was created for the car was made with two major goals in mind that were necessary for the car to be fully functional and autonomous. The first job that was necessary to help the PCB meet those requirements was to take the power that was supplied by the battery that was given to us and provide the motors with the correct amount of power in order for the back wheels to functions and move the car. The second major function of the PCB that was necessary was to provide an interface that would simulate the TFC shield and allow us to control the car with our own code. With these two requirements in mind, the designed custom PCB contained two halves. One half of the custom PCB was the motor control board and the other contained the components necessary for the control of the camera and the servo.

[4] by Ronald Diaz

The left half of the custom PCB contains the motor control board portion that is necessary (Figure 1). This area consists of the MAX620CPN, DMOS transistors, polarized diodes, polarized capacitors of various values, and the three mounts that are terminals that allow us to connect the motor control portion of the board to the motors of the car. There are many pads throughout this half, so all the components could easily be connected and be connected correctly when the PCB was eventually wired together. This motor control board portion contains the H-Bridge that provides the voltage that goes into the motors of the two back wheels of the car, allowing the car to move forward.

The design for this portion was made with a strong influence from the motor control board design given to us earlier this quarter, along with a few changes that were made to help have the most efficient motor control board possible. The whole left half of the board was not grounded with ground planes, only certain areas were made ground planes to make it the most efficient. The lower third, upper third, and areas directly around the DMOS transistors were given ground planes. This way of assigning ground planes did create a problem with the ground of the lower and upper thirds, they were required to be connected together with a wire when the construction of the PCB came around, so they could be grounded properly. The three mounts that allowed the board to be connected to the battery and the two motors were placed at the bottom edge of the board to allow easier access for connecting them and made the placement of the board on top of the car more convenient.  The DMOS transistors for the H-bridge were placed in between the MAX620CPN and the port for the battery to be connected to the board. This placement was helpful because the MAX620CPN was now separated from the full voltage of the battery, which only needed 5V from a separate voltage regulator to be powered, and reduced the effects of the motor on the MAX620CPN. The new design for the H-bridge only required the two low-side DMOS transistors to function properly and provide the correct amount of voltage to the motors in the back of the car. This new way of implementing the H-bridge required the two high-side DMOS transistors of the initial H-bridge design to be taken off and they were not part of the PCB once it was constructed. The two high side DMOS transistors were not necessary to the functionality of the car's motors and would possibly cause problems with shorting the motor control portion of the board. They were replaced with two polarized diodes that connected the source and drain of the outline of the DMOS transistor; this prevented any problems with the outlines for the two high-side DMOS transistors that were still on the PCB.  Two 15nF capacitors were also added near the two terminals for the motors to act as decoupling capacitors and minimize the noise from the motors that would possibly affect the other components of the board.

The right half of the PCB contained all the components that are necessary for controlling the car and allowing it to run the code. It contains two pushbuttons, two potentiometers, two areas for speed sensors, resistors, capacitors, voltage, pads, and a large amount of holes on the upper portion of the board (Figure 1). These components were created on the schematic because these are the same components on the TFC shield and a goal for the custom PCB is to simulate the TFC shield (Figure 2). The pushbuttons and potentiometers were essential for this PCB and running the code. The way the code is implemented onto the car and allows the car to run is through the pushbuttons and potentiometers. These two main components helped control the camera and motors.

These TFC shield components were placed on their own half, so the noise of the motor control board wouldn't affect them. All the components made on the schematic were placed near each other in the effort to try to reduce the overall size of the board and help keep everything organized. The Freedom board that was required to be used with our custom PCB needed 5V in order to turn on, so a 5V voltage regulator, with a 1A rating, was placed on this half of the board to regulate the voltage from the battery and supply the 5V necessary to turn on the Freedom board. The potentiometers were included on the custom PCB for control of the car's speed, it allowed us to adjust the speed quicker without having to constantly go and change a set speed in the code. The pushbuttons were needed on the custom PCB in order to start the code, one pushbutton controlled when to start the motors and the other was used so the camera could start reading data on what was in front of it.

## Design and Performance Summary

The car’s paper design, which consisted of the code and PCB schematics, worked surprisingly well. The physical design of the car however, leaves much to be desired. The PCB was too crowded, and for competitions, needs to be redone so that everything has room and important power and ground wires don't have to be rerouted through a breadboard in order to work properly. The car’s performance has a similar story. It met the standards of the class, and completed both class competitions with a passing grade, but for the competitions in Spring Quarter with other schools, it will have to be optimized to perform at a much higher level in order to be competitive. All in all, it is merely a working, average car, but the design has the potential to be much faster and cleaner.

# Appendix

Figure 1

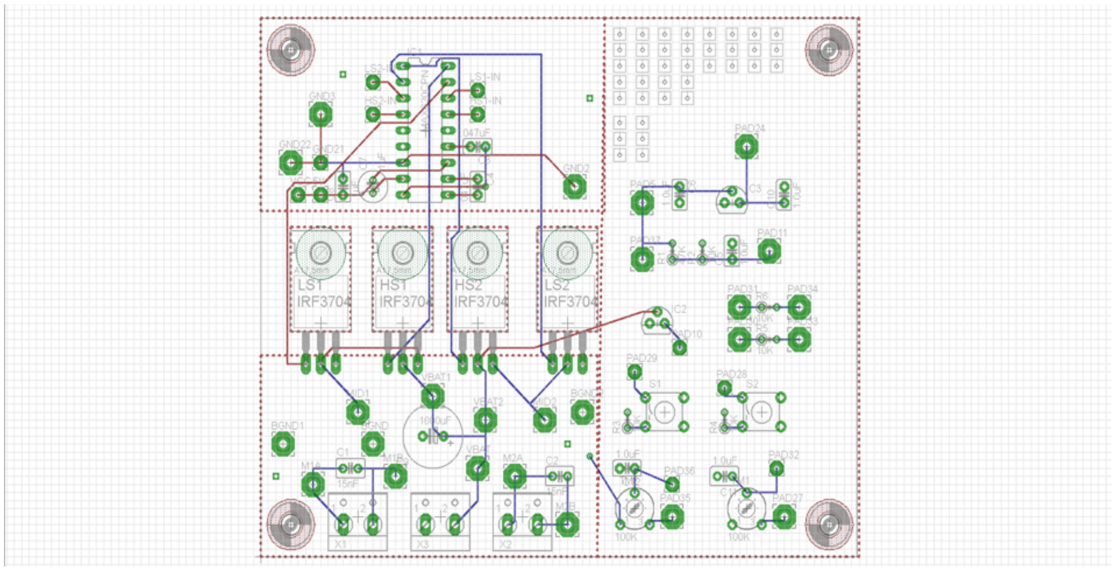


Figure 2