# NATCAR Final Report

Andrew Yunseok Chung, Samuel Dawson, Justin Laguardia

EEC 195B

March 19, 2015

## I. Overview

The objective of the NATCAR project was to design, program, and build a model car that could navigate a course without driving off the edge-laned track. The course consists of obstacles such as intersections, s-bends, chicanes, and hills. The Freedom KL25Z development board, interfaced with a TFC shield, was used to implement the design.

The car detects the track using data captured from a single line-scan camera. There were several reasons for using only one camera. The primary reason was to increase simplicity of the system. A two camera setup creates additional variables that must be tuned and accounted for. With a one camera setup, alignment is much simpler. Additionally, the one camera setup fits in with our edge detection and steering method. Rather than keeping an eye on both edges of the track at all times, we center the camera so that an edge is only detected once we begin to veer off course. Essentially, we treat edges of the track as something to avoid, rather than something to follow.

The camera data is processed to find an edge. The edge is then used by a PID controller to determine the steering angle by the servo. Motor control was implemented using a closed-loop PID, which takes as input speed data acquired from a rotary encoder attached to a motor gear. Closed-loop speed control is advantageous over open-loop control because the controller can compensate for changes acting on the car. For example, the car is able to apply additional power on the climb of a hill and reduce power coming down to maintain a constant speed. Closed-loop speed control can also be implemented using camera data, but it cannot respond to hills.

Some user-interface features included on-the-fly variable changing using potentiometers, and LED indicators for camera detection and acceleration/deceleration. On-the-fly variable changing This was implemented by taking advantage of the DIP switches included in the TFC shield. Depending on the configuration of the switches, important variables such as steering servo control PID variables can be changed using the potentiometer. For example, a setting of 1100 allows the user to change the P constant for the servo motor PID controller.

<u>Task breakdown:</u>

- Andrew Yunseok Chung Effort: 45%

  Tasks: Implementation of camera, servo motor and DC motor interfaces, PID controllers for servo motor and DC motor, user interface
- Samuel Dawson Effort: 30%

  Tasks: Safety code, Speed control, Tuning and adjustments, car dynamics (speed control, servo control, track behavior), Debugger
- Justin Laguardia Effort: 25%

  Tasks: Car structure and design, 3D printing, Motor control board, Tuning and adjustments

## II. (a) Technical Report: Andrew Yunseok Chung

There are several options for implementing speed control: hall effect sensors, IR emitter/receiver pairs, and rotary encoders. The main considerations when deciding between these options were time/cost, physical feasibility, and accuracy, from greatest importance to least. Time was a large factor because it would take three days to ship a part and two days minimum to research the part, test its electrical characteristics, write and debug code to test it, and attach it to the car. Paired with the fact that the part might not perform as we expect, the investment of both time and money to order a part made the task undesirable. Physical feasibility was the second largest concern. There were many questions related to how the part was going to be attached on the car when deciding between the three methods. First, where was the part going to go? A reading of the front wheels would be the most representative of the speed of the car, since it will not spin when off the ground and is less likely to lose traction compared to the rear wheels. However, how would the sensor be attached to the front wheels such that it would follow the wheel as it turned? Second, for each of the sensors, which would be easiest to physically attach to the car? The hall effect sensor method would require a magnet to be somewhere on the wheel and the sensor to be placed in very close proximity to it. The magnet would have to be small and lightweight otherwise it would interfere with the wheel. The IR transmitter/receiver would have to either be on either side of the wheel, or it would have to reflect off some surface on the wheel. In either case, the modules would need to be small to fit in the tight space between the motor cage and the wheel. For both the hall effect sensor and the IR LED, the sensors would have to be maintained in place somehow. The rotary encoder along with the gear on its shaft would need to fit in on the car. Third, the accuracy of the encoder would be greatest, whereas the accuracy of the IR emitter/receiver and the hall effect sensor would depend on the pattern of the encoding and the number of magnets on the wheel, respectively. Among these options, we ultimately decided on the rotary encoder because we found a pair on a car used in previous competitions. This solved our concerns of time/money, physical feasibility, and accuracy because we had the parts on hand, we knew how to attach it to the car, and the rotary encoder was the most accurate of the three methods. Unfortunately, only one of the encoders was functional and the second may have been destroyed in the process of figuring out how they worked. Since the parts cost upwards of $30, and they only shipped from China, we decided to use the single rotary encoder despite the risk that if the single wheel loses traction, the speed control would fail.

The device used to implement speed control was the Nemicon OME-N 100 rotary encoder. Inside the rotary coder, there is an LED emitter and a phototransistor. As the shaft turns, the LED turns on and off and the transistor produces an output according to the amount of light it sees. The strength of the LED and therefore the range of the output voltage can be configured by controlling the current going through it. There were three steps in implementing the rotary encoder. First, the LED must be turned on and we must read the sine-wave voltage output from the photo-transistor. Second, we must parse the sine-wave output to get a square-wave digital result. Finally, the pulses must be analyzed using a timer to determine the rate of pulses, which corresponds to how fast the shaft is rotating.

The encoder was configured with the help of the data sheet, which provides an example circuit that suggests a current of 10 mA through the LED, a supply of 5V to the receiver, and an emitter resistor of 1k Ohms. The voltage drop across the LED was measured to be 1.1 V. Therefore, the LED current is determined from the equation: $i = \frac{VDD - 1.1V}{R}$. The encoder would be powered from the KL25Z, so VDD = 3.3V. Solving for i = 10 mA, we find that R = 220 ohms. One deviation from the sample circuit was the supply to the phototransistor, which was changed from 5V to 3.3V. The effects of this is a lower current, but same voltage at the output. Testing the recommended circuit resulted in a sine wave peak-to-peak output voltage of only 1.2V. Ideally, we want pulses of 0 or 3.3V so that the output of the encoder can be fed into a pin on the KL25Z and analyzed using a timer to determine the period between the pulses. This can be achieved by feeding the sine-wave into a comparator circuit configured such that, for example, if Vi < 1.2/2 V, Vo is deasserted, and if Vi >= 1.2/2 V, Vo is asserted, where Vi is the output of the phototransistor and Vo is the output of the comparator. The KL25Z has a comparator peripheral, which was used in conjunction with a DAC voltage reference set at 0.61875V. Using the comparator that comes in the KL25Z board spares extra circuitry from hanging off our board. In the future, however, we may want to have an external comparator circuit on a PCB so that the comparator on the KL25Z can be used by another peripheral on our car (2D camera comes to mind). The comparator also has a hysteresis option, which is used to give a buffer around the switching point. For example, if the input voltage changes from 0.618V to 0.619V, the comparator will toggle from a logic low to logic high. However, if the input voltage fluctuates due to noise between 0.618V and 0.619V, the comparator will toggle rapidly, which is not reflective of the switching desired from a sine wave. With hysteresis, a buffer zone in the switching point prevents such unwanted rapid toggling due to noise in the input. In our case, the hysteresis level was set to maximum, which is approximately 38mV according to the KL25Z datasheet.

Finally, the pulses from the comparator must be analyzed to get a value for speed. This can be done in the frequency domain or in the time domain. In the frequency domain implementation, the number of transitions in a certain interval of time would be counted. The more transitions, the faster the speed of the car. In the time domain implementation, the period between transitions would be measured. The smaller the period, the faster the speed of the car. In general, it is better to find the frequency in an interval of time if the pulse frequency is high and better to measure the period between pulses if the pulse frequency is low. Since the rotary encoder used in our car has 100 pulses per revolution, the number of pulses were recorded in a period of time. This was implemented using an up counter timer module set to reload every 18 milliseconds. The result is a speed value that ranges from 0 to 400.

## II. (b) Technical Report: Samuel Dawson

Our car is a low slung, single camera design. By keeping all components as low as possible (with the exception of the single stick and camera atop it), the car is more stable and has a shorter lever arm for the wheels to act against laterally in turns. Additionally, we place most of the weight in the rear. Keeping weight off the front of the car allows the car to yaw with less resistance. This counteracts our car's propensity to understeer.

The single camera was chosen primarily to reduce complexity. There are of course advantages to a two camera design. Two cameras allow for more total resolution when viewing the track, and can each be trained on one of the track edges. However, the alignment issues associated with having two cameras are very prominent. In a two camera design, the cameras must have perfectly mirrored positions and view angles, or the car will not react the same to left edges versus right edges. A crash or accidental bump can throw off the alignment and be a pain to fix. On the other hand, the single camera design is physically more tolerant. The camera simply needs to be pointed along the long axis of the car and have the correct pitch. Furthermore, the single camera design allows for simpler programming and faster processing - only half as many data values need to be acquired from the ADC.

Our algorithm for controlling the car relies on the single camera having a field of view just slightly smaller than the width of the track. In the totally ideal, straight track case, the camera detects no line, and the car proceeds straight ahead. As soon as the car begins to drift off-center or the track turns, an edge comes into view and the car steers away from it. Steering is done entirely based on current error, there are no derivative or integral terms. However, to increase stability, this P term is reduced for very small errors.

Therefore, turning is based on P * (distance from end pixel of camera to detected edge), with P divided by 3 for error < ERROR_THRESHOLD or divided by 2 for error < ERROR_THRESHOLD * 2 (as seen in updateServoPID in servo.c).

We do have software support for full PID for servo control, however at the moment we find that our car behaves well with I and D set to zero (as can be seen in the ****configure variables**** section of main.c)

Most of the complexity of our control algorithm lies in the speed domain. We split speed into four domains based on the calculated error in edge position. The following four domains (described in order of decreasing speed) can be seen in the section labelled ***speed ctrl*** in main.c:

--In the ideal, no edge or small edge-error case, the car will increase speed (starting from a base speed determined by POT1) every time through the control loop, up to a maximum defined as NEVER_EXCEED_SPEED.

--As soon as edge-error increases past a certain threshold, we enter the second domain. Here, speed is set to the base speed. This speed is considered appropriate for very mild turns and correcting orientation on straightaways and through wiggly pieces of track.

--The third domain is for traversing regular turns. This is currently set to about half of the base speed. In this domain, the car is applying nearly full turn to the front wheels, and may understeer slightly.

--The final domain is our emergency turn. When edge-error becomes very large, we must cut speed immediately to prevent plowing through a sharp turn. This domain is usually entered when a turn is encountered at the end of a long straightaway, when the car is travelling very fast. The previous three domains used our closed loop speed control to set velocity. However, in this domain we don’t use it for two reasons. The first is that closed loop speed control has a bit of lag, particularly for very low desired speeds. The second is that in the emergency turn domain, we use a bit of differential thrust to help the turn. This can be seen in our direct call to setMotorPWM(). The inside wheel on the turn is set to zero speed (zero applied voltage), and the outside wheel is set to a slow speed (low duty cycle PWM). Since the rotary encoder is only coupled to one wheel, true closed loop speed control is impossible without assuming the two wheels are travelling the same speed, which is not the case in this domain. Coupled with full turn-in from the servo, this ends up manifesting as initial understeer (sliding out) when first responding to the turn, transitioning into slow coasting with a full turn until the the edge-error is detected to be within the limits of one of the faster domains.

The current domain is not dependent upon the previous domain, and is determined based on the most recent edge-error every time through the main control loop.

## II. (c) Technical Report: Justin Laguardia

In our final design for attaching the line-scan camera to the car chassis we 3D-printed parts to better fit the camera and to add firmness to the pole that holds the camera. The three different printed parts were the rod base (figure 3c.1 in appendix), the camera hinge (figure 3c.2 in appendix), and the camera mount (figure 3c.2 in appendix). To modify and print the parts, the files were edited using Autodesk’s AutoCAD and TinkerCAD, and KISSlicer. The parts work such that the line-scan camera is attached to the mount by screws, which is then in the attached to the hinge. The hinge in then placed on a pole and can be adjusted using a nail, screw or tape. Lastly, the pole is connected to the base, which again can have a nail or screw to hold it. As a side note, currently, a balsa wood stick is used as the pole because of its small weight. Together these parts made the car more durable and made recognizing the track edges easier because the base is very sturdy and does not shake as much as the other designs did.

These three parts were printed out multiple times. Each time we tried printing the set of parts, we varied the amount of hollowness to see how hollow we can make the parts so that they still function correctly, aren't too fragile, and have the least amount of weight as possible. All printed parts were chosen to be printed using PLA plastic, which is pretty lightweight and sturdy to begin with so the main concern became more focused on how lightweight they can be made. Printing parts that are not hollow, on other hand, can significantly increase the weight, also the time to print can easily change from half an hour to hours. The range of varied fullest percents was recommended and chosen to be between 20 to 15, where 100 percent is completely solid. In the end, only two sets of parts were completely printed because below 18 percent, the parts failed to print correctly on the printer. The two sets were 20 percent full and 18 percent full.

For class competition II, the additional requirement was to have the motors controlled by another board other than the TFC shield. A custom printed circuit board was drawn up using Eagle software. The motor control board created in Eagle was designed to be similar to the one used in Lab 8 from Fall quarter. The board also uses a MAX620 to process the H-bridge controls and two ISL9N312AP3 N-channel MOSFETs. The differences in the two boards were that it replaces the two high side H-bridge transistors with 1N4001 diodes and added extra connections for having multiple potentiometers on the board to easily control and adjust variables. Thus, this motor board would only focus on just the low side H-bridge because reversing is not needed for our car and that lowside was recommended for its better control. Please refer to figure 3c.4 and figure 3c.5 in the appendix for the schematic and the airwave layout respectively. After the schematic was created, a simple airwave layout was made. However, at this step it was realized that for competition II it would be easier and faster to use the motor control board from lab 8 because we could still use the TFC for other controls. Thus, the layout has been put on hold so we can further develop it instead of rushing it just for competition II.

In competition II, the PCB from lab 8 was used. To control the motors, M1A was connected to MID1, M1B was connected to VBAT1, M2A was connected to MID2, and M2B was connected to VBAT2. Wires were also soldered from the TFC shield onto the motor control board such that the PTC1 and PTC3 pins on the TFC shield were connected to the LS1_IN and LS2_IN pins respectively. This produced a low side configuration. The two ground plated were also connected together and then connected to the TFC shield's ground. Although a 5 volt regulator, the LM2940 low dropout regulator, was recommend for this motor, we unable to add the regulator due to insufficient capacitors. To work around this, we observed that the regulator was unnecessary for us since the supply voltage for the MAX620 has a range of 4.5 to 16.5 volts, which meant that connecting it to the battery pack would be within the range to power the device.

## III. Design and Performance Summary

Many objectives were accomplished throughout this quarter. We were able to successfully move through the first track prior to the first milestone checkoff by implementing the proportional control will using two line-scan cameras. Shortly after we were then able to successfully switch to using only one line-scan camera which then surpassed the top speed of our two camera design. Our next objective of having speed control to address the track's hills and overall car performance was achieved by using a rotary encoder, which was mentioned above in the technical report (a). Our last, and main goal, was to become the fastest the car in the class. From class competitions, we received the second fastest time for Competition I and fastest time for Competition II, 5.6 ft/s and 5.8 ft/s respectively.

With our many accomplishments, we still have a few things that we would have done differently. First, we would have liked to have started this quarter with using a one line-scan camera car because it would have saved us a lot of time, since we began this quarter using two line-scan cameras, one on the left and one on the right. Thus, our time that we put into designing and testing the car with two line-scan cameras could have been put into the time we needed to test and adjust the one camera method. Another task that we wish we could have done was to have tried using one 2D camera because of its huge potential on being able to see a lot more of the track and hopefully having a better to predict the track.

## IV. Safety

Multiple steps were taken in the project design and construction to ensure that the car was not a hazard. The first step after the car chassis was built was the addition of a front bumper. The initial bumper was built from styrofoam packaging. After realizing that the styrofoam bumper was too stiff and brittle, a new bumper was built using a sponge foam material that was given with the Freescale Cup Starter Kit.  Towards the end of the quarter, the big bumper was removed so the car would stop getting stuck on the track's crack on the downside of the hill and to decrease the total weight of the car.  The car chassis bumper was also flipped so that is was rested on top of the chassis which avoided the issue of hitting the track on the downside of the hill.

Quickly after the bumper was attached, Sam used his knowledge and experience in EEC 172 and came up with the amazing idea to add a safety stop (for testing purposes only of course) by using a programmable launchpad and an IR remote to remotely stop the car if something went wrong.  Thus, an IR signal from a button press on the remote would turn off the H-bridge enabler and almost instantly make the car stop moving.  This was extremely useful in cases when there was another car on the track and collision was about to happen and when the car went off track. This step was later removed after the last safety step was implemented.

For a more legal safety step, code was added to stop the car, by turning off power to the motors, if the linescan camera picked up the carpet (only black, no white track) for a certain amount of time.  Please refer to appendix figure 4.1 for the code.  When the car goes off track it will stop the motors by setting PTE21, which is the H-Bridge enabler, to low.

## V. Appendixes

Rotary Encoder Datasheet: http://www.nemicon.com/pdf/ome-n.pdf

Figure 3c.1 - Camera Rod Base

Figure 3c.2 - Camera Hinge

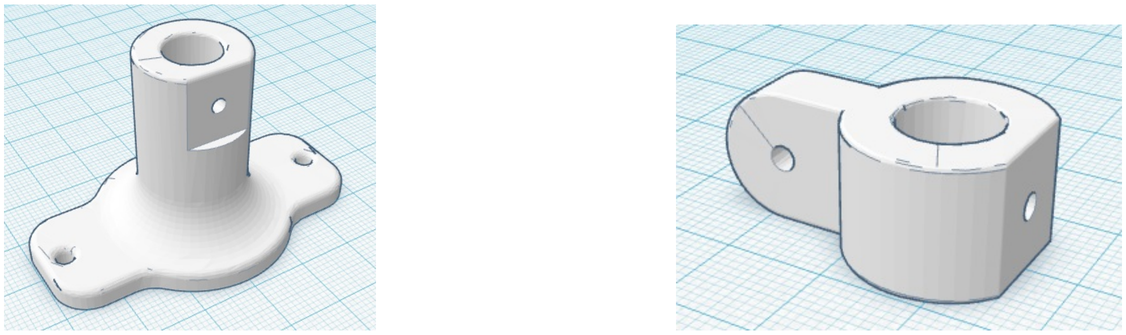

Figure 3c.3 - Camera Mount

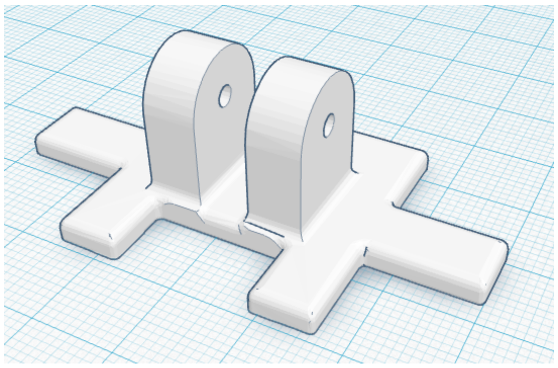

Figure 3c.4 - Eagle Motor Control Schematic

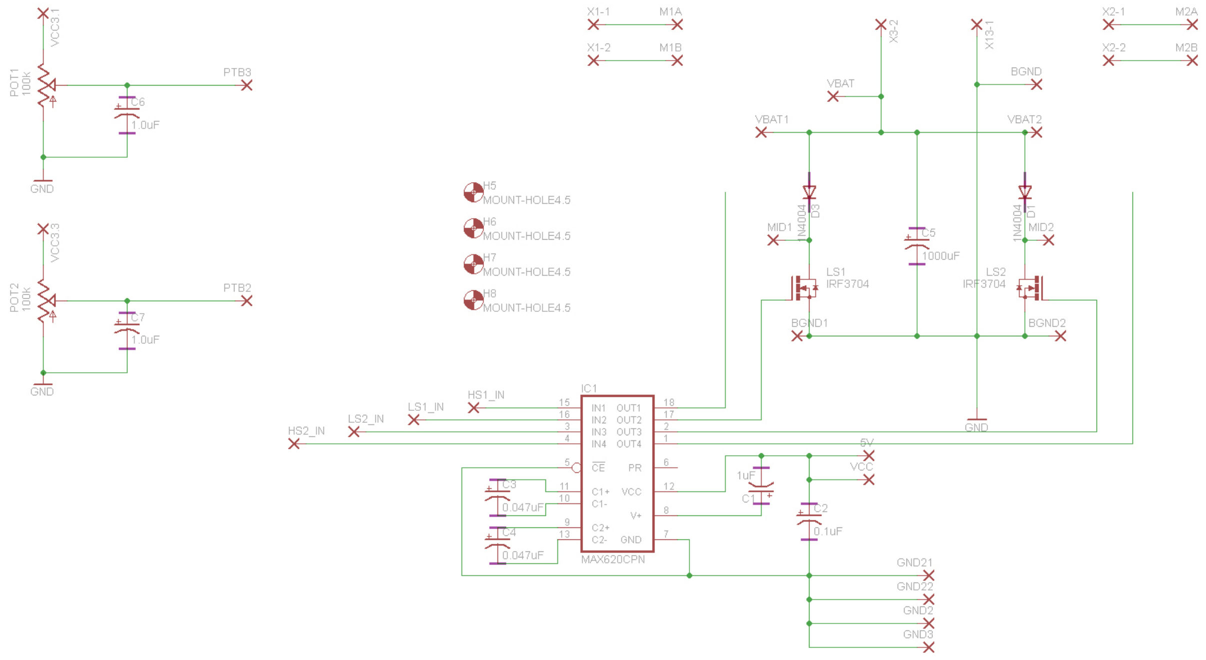

Figure 3c.5 - Eagle Airwave Layout

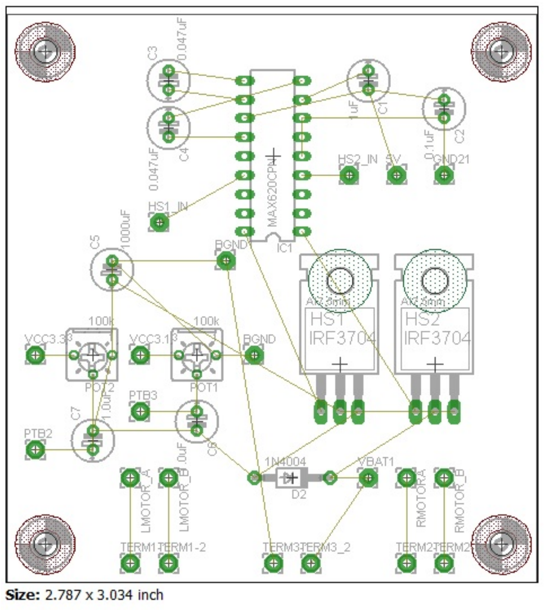

Figure 4.1 - Safety Code

<u>"main.c"</u>

```
if (blindCount > BLIND_THRESHOLD)
                        { disableMotor(); }
```

<u>"linedetect.c"</u>

```
int isBlind(unsigned char buffer[BUFFER_SIZE]) {
        int i, count;
        count = 0;
        for (i = LEFT_EDGE_BOUND; i < BUFFER_SIZE - RIGHT_EDGE_BOUND; i++) {
                if (buffer[i] < 0x40) {
                        count++;
                }
        }
        if (count > 95)
                return 1;
        else
                return 0;
}
```