EEC195A-B NATCAR 2018-2019

Final Report

Student Team: Date: 3/19/2019

Ryan Blanc SID: 913340679

Huangze Tang SID: 914294672

Eric Qing SID: 913298695

Table of Contents

I. Overview(Executive Summary)

For our project, we separated the work into three sections. The first part is the programming of our camera microcontroller, an OpenMV M7, that allows our car follow the track made of white tape on dark carpet. Our programs can accelerate on straight sections, decelerate and turn during corners and finally stop after it sees the finish line marker. The second section of work is the PCB board and signal analysis. Our PCB contains all necessary components to process signals from our OpenMV to control the servo and motor so the car can move as we wish. The final section of the work is to assemble the car together and design a mounting system to hold our circuit board and camera. We finely tuned our program and car so it can cope with all conditions that could occur on the track.

Our team assigned tasks as follow: Ryan Blanc takes the tasks of assembling the car, soldering PCB boards and designing and attaching a lightweight mounting system on the car. Eric Qing is in charge of designing PCB board, assembling car kit, designing the camera holder and troubleshooting of car's components during testing in the lab. Huangze Tang is responsible for writing the program and modify different parameters on testing tracks in the lab to make sure it stays on the track. The approximate contribution percentage for each team member is around:

Ryan Blanc: 32%, Huangze Tang: 37%, Eric Qing: 31%.

II. Detailed Technical Reports

Program - Huangze:

During this quarter, based on the programs we wrote last quarter in the labs, we started to tune the PID controller to write the real control algorithm for our Natcar. The program ran on the OpenMV microcontroller, we interfaced it to our PCB board to control our car.

The program used the while loop to capture the photo through the sensor at the frequency of around 80fps. Taking advantages of the API built in the OpenMV, we could easily deal with the photo we captured. We could draw the brightest part from the photo (usually the white line on the ground) to use as the direction for our car to trace while running. After analyzing the photo, we could use the PWM output on our microcontroller to adjust the servo input and the motor input so as to change the direction and speed of the car as needed. Finally, the car would stop automatically when it detected the finish line marker, which was three pieces of tape. To achieve the goal, I made a lot of design decisions to improve performance.

First, I only used one blob detection in my program but not two or three. I set the top portion of the photo to be the ROI we wanted to find the blobs in to decide the directions

where the car should go. This decision might lower the accuracy of how the car would follow the track because only one blob detection would be easily affected by the noise and make the process unstable. However, if I added more blob detection, the fps would drop from 80 to 40. I believed the fps would be extremely helpful for the car to follow the track. When the car was running fast, the fps also needed to be increased to fit the car. That's why I traded the accuracy for the fps.

Although our code utilizes only one blob detection, my stop line detection algorithm was efficient. When the program found there were three blobs in the ROI, and each of them had width smaller than 15 pixels, which proved they were not the crossover, the program would respond to stop the car automatically using the reverse motor spinning to slow down the car as soon as possible so it could stop in the desired area. Moreover, to avoid the program to detect the finish line when it starts before the line at the beginning, I used the Utime API to record the program running time. Only when the program has started 10s, the stop line detection algorithm would be activated to stop the car.

When I was tuning for the PID controller, as I wanted to speed up the car, I increased the value of Kp, and I found the oscillations of the car too extreme. It would oscillate and shake even when the car wasn't moving, so I tried a lot of ways to decrease the oscillation. First I deleted the integral term because the Ki term would cause some kind of

the oscillation. But this way was not effective, there was still far too much oscillation. Next, I set the dead-band for the current error. When the error was within some range of the straight forward direction, the car would run definitely straight forward instead of responding to the error. This way reduced the oscillation to some extent, but when I continued increasing Kp, the oscillation showed up again. In order to solve this problem, I added some algorithm to the error buffer. When the newest error was close to the previous one, the previous one would not be updated. It would remain the value until some error had a large difference from it. This strategy made a great improvement in the performance of our car to decrease the oscillation.

In the motor speed portion of the program, the code would adjust the pulse width percent for the motor based on the deflection angle. When the deflection angle is small, the motor should spin fast; while the deflection angle is large,such as when the car was turning, the motor should slow down. In another version of my code, I also added a drifting algorithm to boost up the car. I implemented the drifting algorithm with three new parameters: straight-counter, turn-counter, and a drift-wait term. The straight-counter was used to test how long the car had speeded up. The turn-counter was used to prevent drifting at a zigzag or slight turn on the track. The drift-wait term was used to count several frames that the car should stay in the drifting mode. When the car was running at a high speed, the straight-counter would count up. When the car met with a tight turn, it

would first slow down, then the turn counter would begin to count up. When the straight-counter and turn-counter both reached the defined threshold, the motor would have a reverse pwm to drift the car. And the drift-wait would let the motor reverse for several frames to drift the car over the tight turn. Even if this algorithm was not used in the time trial, I thought it was a creative design.

Another aspect of the coding was the Bluetooth Control algorithm, so that we could remotely control the car to prevent it from crashing with walls. This algorithm would also output the data from the trial runs to our phone, making testing easier easily. I also set the color tracking thresholds to (250, 255) to track the lines more precisely.

PCB Board - Eric:

Our PCB board is designed to read the signal from the camera and send them to motor and servo to drive our car, using the power from the battery. We started our design by drawing the schematic that contains all the recommended components. In the schematic, we designed 2 rows of header pins. One row, MVLEFT, contains header pins for motor and servo controlling signals and also Bluetooth control signal. These pins are connected to the OpenMV through jump wires to receive signals. The other row, MVRIGHT, is designed for testing. These two rows are connected directly so we can monitor the signals when troubleshooting problems. For the rest of the schematic, we followed the lab

manual requirements and technical documents, such as connecting signals to VNH5019A-E chip and take its output to a terminal block to connect to the motor. The chip behaved as a half bridge, which allowed us to give motor different modes of operations such as: speed up, brake, and neutral. I also designed a power system two LDOs, one LDO for the servo, and another LDO for the rest of our car. Next we routed the control signals, like the PWMs for motor and servo control, from our microcontroller on our PCB. We carefully added decoupling capacitors to all vulnerable areas of the circuit to reduce electronic noise.

After we finished our schematic, we were able to begin working on designing our two layer PCB. For the first trial, I put all components in order so that all of them have fewest of traces crossed. However, by using the star ground method, I needed to separate components into different sections, one for power and motor controlling components, one servo controlling components, and one for logical signals. The best solution was to put the ground pour into three sections and overlap them at the center of the board. At this point in the center, I put a through-hole header that connects the top and bottom ground pours. An additional result of changing the grounding was the number of crossed air-wires increased, which means that vias number and the complexity of the routing increased. To solve this problem, I used through hole “vias” to make connect the top and bottom layers, but keep the number of through holes at the minimum to reduce delay and

complexity of tracing signals during troubleshooting. The result of my first trail was a board of 5x5 inches, which was fairly large, so I decided to shrink the size of the board in order to get better performance.

For shrinking the size of the board, I organized the components closer together. I first put the VNH5019-A chip to in one corner of the board, and placed resistors next to it because many resistors are connected to this chip. Then, I adjusted the thickness of the traces for high power signals such as for VCC and for the traces for the motor signals.

Then I grouped the header pins together so the connections and signals will be easier to test. Next, I put the terminals for the battery and motor grouped together at the edge of the board for easy access. The final part was the servo controller section of the circuit. I tried several configurations to make it simple and compact, the result of the new arrangement was the board's size became 2.5x2.9 inches. However, when Ryan soldered the components, he found the arrangement was too compact and it was difficult to solder, especially for replacing parts. At the same time, we met with a problem that our chip burnt out. The result was the outputs from the chip became quite strange and could not drive the motor. Since the chip was surface mounted so the re-soldering process was very hard, attempts to resolder the board did not work either. Therefore, I asked TA for some advice on how to fix the problems.

According to TA's advice and responses from my teammate, I redesigned our board and add more spaces for components so it could be modified if needed. I added more resistors, a terminal for the switch and changed the positions of the decoupling capacitors, between chip and motor. Another thing I fixed was to make the power traces thicker than before, to make sure it could handle the higher power signals. I also spread the components out in a larger area, the new chip's size is 4x3.5 inches. We also figured out one of the problems with our output signals. In order to have the proper PWM signal, we have to connect our board to the motor. If we did not connect the motor and only looked at the signals on an oscilloscope, we could not get the correct output. This version did well on the time trial one.

Mounting Hardware - Ryan:

Initially, I designed that instead of having one large platform which would hold the PCB and camera stand there should be two separate platforms which minimize the size and weight of the additional parts. At first I considered 3D printing more of the mounting hardware (see figure 9), but due to limited printer availability I found the parts I made without 3d printing worked quite well. The tradeoffs for almost all the hardware modifications was stability vs. weight. There were modifications that would make our car

even lighter, such as using balsa wood, but we wanted to keep our car intact in case of crashes.

At first, we considered having two poles that stick straight up, or a tripod to hold the camera up. We wanted something that was more adjustable, I saw another group had successfully shortened a selfie stick. I thought about doing that at first but had never taken a telescoping mechanism apart. Now that I knew that it could work, I bought a selfie stick and shortened it for our car.

I have a drill press, so I took measurements from our car and determined what size platform to make. To do this, I made a template tracing an extra car chassis onto a piece of plywood for future reference. Initially, I tried connecting our second layer above the chassis (the parts that hold the PCB and Camera holder) to the plastic rods that came with the car for mounting the plastic car body to the car frame. These plastic sticks did not seem sturdy (see figure 1 and figure 2), so I decided to replace them with wooden dowels (seen in figure 3).

I got some scrap clear acrylic plastic and used it to make a small platform for holding our selfie stick camera holder. Initially, I cut too big of a hole for the selfie stick, so I had to buy more drill bits and make another one which was more accurate. I went to the

hardware store several times to buy parts such as small screws for mounting, a metal pipe adapter which firmly holds the selfie stick, and a few odds and ends (see figure 4).

While I was making the wooden dowels support columns I decided to try drilling a few extra holes in the car chassis for mounting support. After we tested the car and the holes did not cause an issue, I realized the chassis could withstand losing more mass without losing the structural strength we needed. I needed to make a few changes such as trimming the stock parts so our camera holder would fit, as well as drilling more holes for future additions, so I dismantled almost everything from the chassis. From this point, I marked numerous points where to drill holes of different sizes, some holes where dowels for support could be placed, others just where material could be trimmed to reduce weight. After making the adjustments I reassembled the car and recalibrated the wheel height.

While Huangze was making changes to our code, I made a laser cut speed sensor wheel and began testing places and ways to mount it (see figure 6). I also bought 2 kinds of speed sensor boards to accurately determine car speed. Ultimately, we did not end up using a speed sensor nor did we use current sense in our code.

III. Safety:

We used the recommended safety features such as Bluetooth control to turn the car on and off, with this we can remotely disconnect the motor when we finish testing so our car will not run if we do not intend to. To make our car safer, we mounted the battery low on the car chassis and taped it in a place where if the car crashed it would not be damaged. Another precaution we implemented was that our board used an IRLU8743 Mosfet, this protects the board if the battery is connected with the wrong polarity. When we tried to fix our car we removed it, so it would be one less thing to troubleshoot. After we removed the MOSFET we soldered the two correct legs together and marked our battery packs to prevent us from accidentally connecting it the wrong way. Doing this fixed our car for the time being.

Another of the main ways we kept our car safe was keeping the impact foam on our car. By designing our car to survive potential crashes by keeping critical components away from the front, as well as minimizing weight and keeping the center of gravity low, our car was designed to minimize damage to itself and to whatever it crashed into during an accident. We made sure the gears on our car were covered by a piece of wood so no cables would get caught in them, this also minimized the chances of someone getting their fingers or clothes caught in the gears. On this same piece of wood we mounted a

large toggle switch in order to connect the motor, for the times when our car did run off course before we stopped it via bluetooth on our phones.

IV. Design and Performance Summary

If we were to do this over again we would spend more time on our PCB, making sure we labeled every net, making all traces far thicker than recommended, and spaced everything out to make troubleshooting as easy as possible. We ended up making 4 different PCBs for our car to troubleshoot and try to fix it, even then we could not get our car working for the second time trial. Because we had so many issues with our board, making more of the parts modular and removable would have been ideal for troubleshooting, such as a second PCB which only had out surface mount H-Bridge IC, or use simpler solutions like using diode to get rid of the chip for stability.

The board we used for time trial one suddenly broke when we were testing for the second time trial. We first thought it was the contact between chip and board was not secure so there was no output, but it turns out that was not the case. We tested the input signals INA and INB on the chip. It turned out the input did reach the chip, but the output was zero all the time. We tried using a hot air gun to repair connections on the board to no avail. We could not figure out what caused this problem, so we tried replacing the surface mount IC. We ended up having 4 different circuit boards (see figure 5), but

despite multiple attempts to check the connections and replace parts, we could not get our car to run for time trial two.

V. Appendices

First Revision of Car:

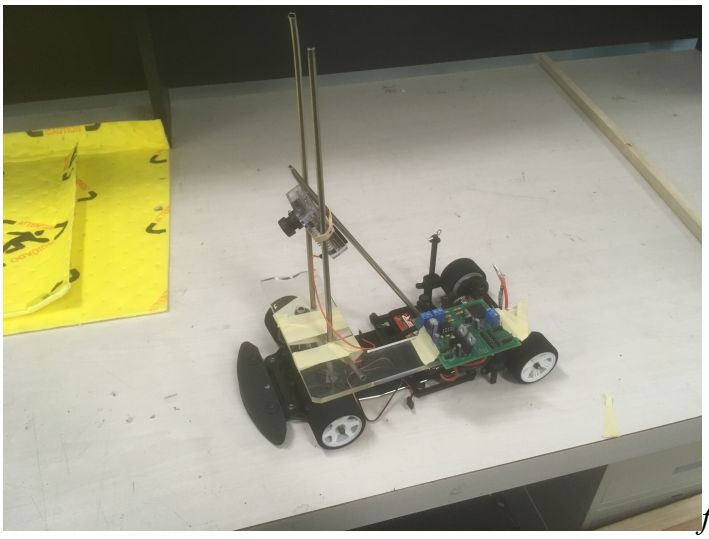

figure 1

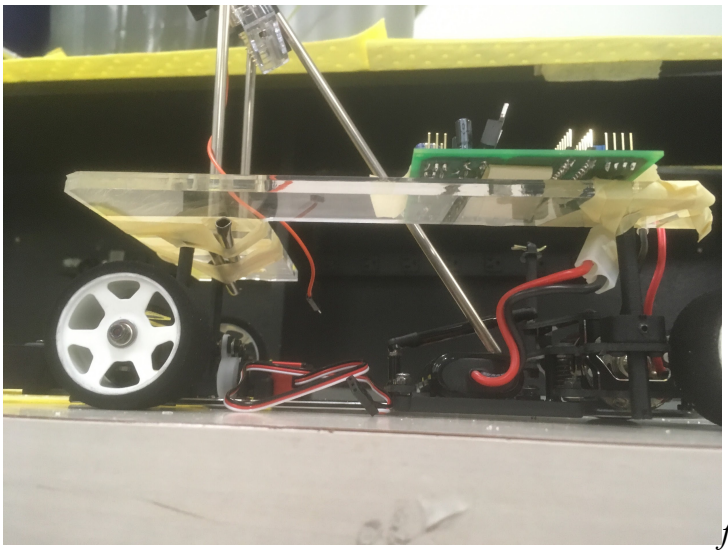

figure 2

Second Revision of Car:

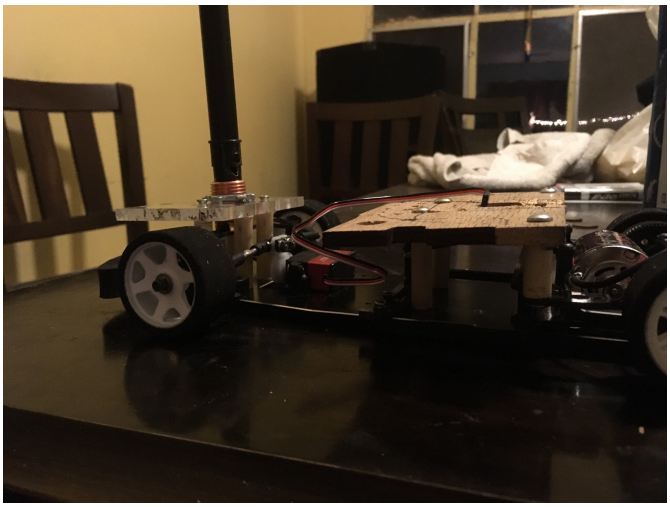

figure 3

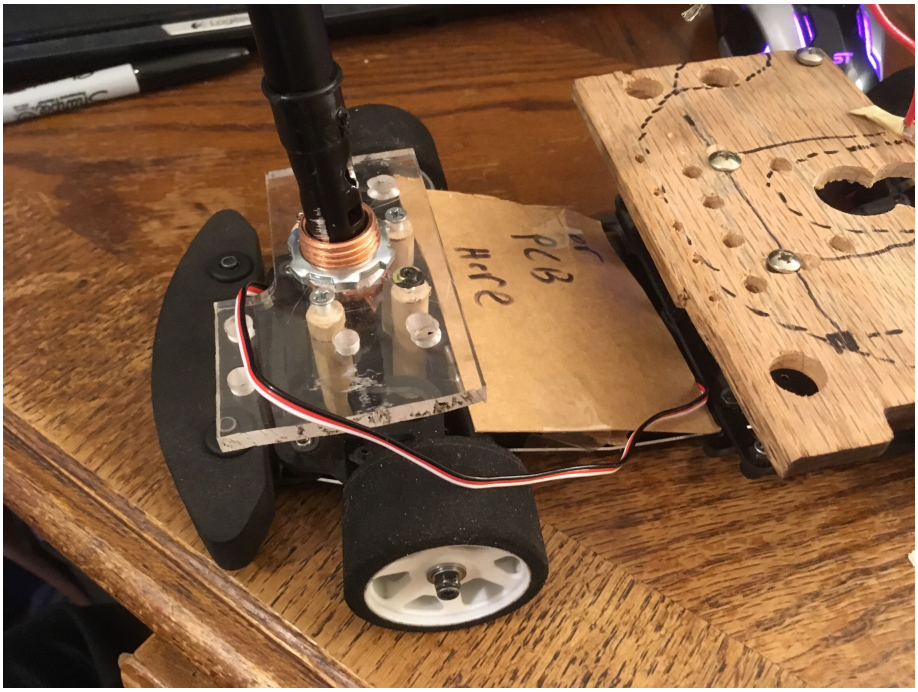

figure 4

Last Revision of Car:

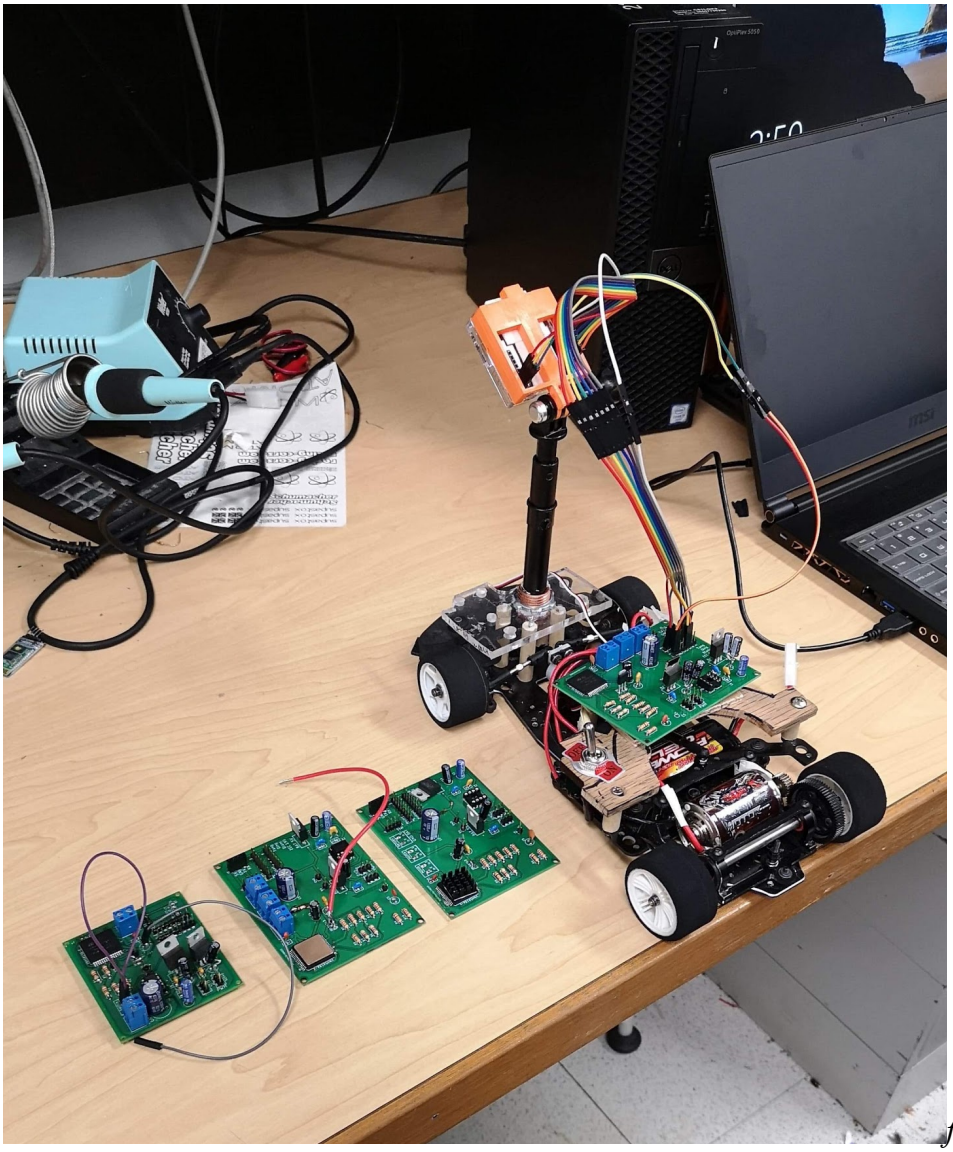

figure 5

Attempting to Mount Speed Sensor Wheel:

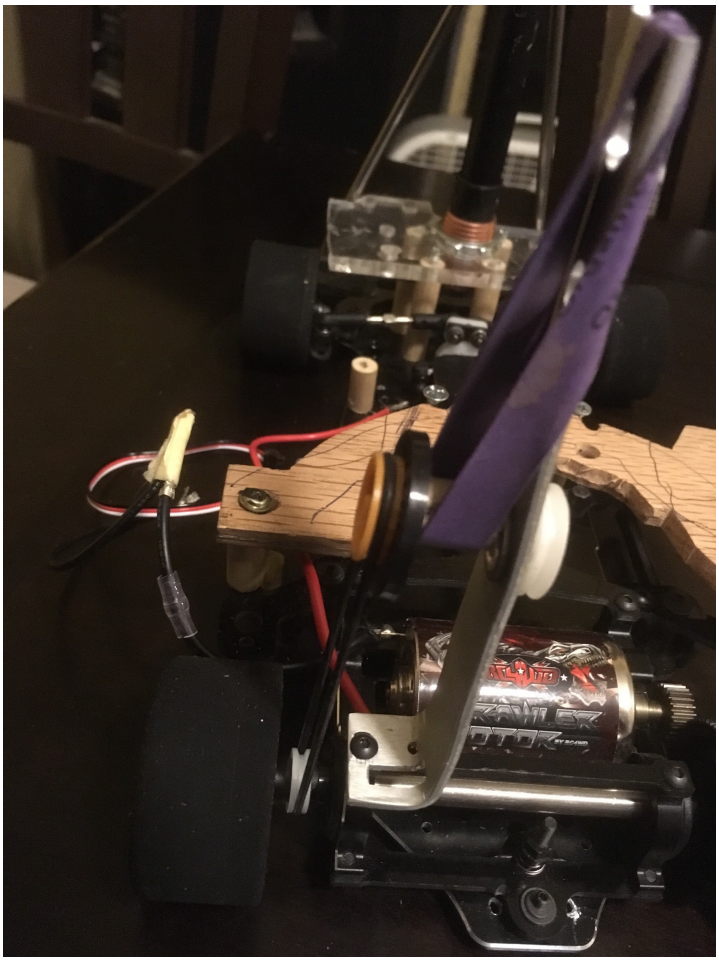

figure 6

PCB Version 1

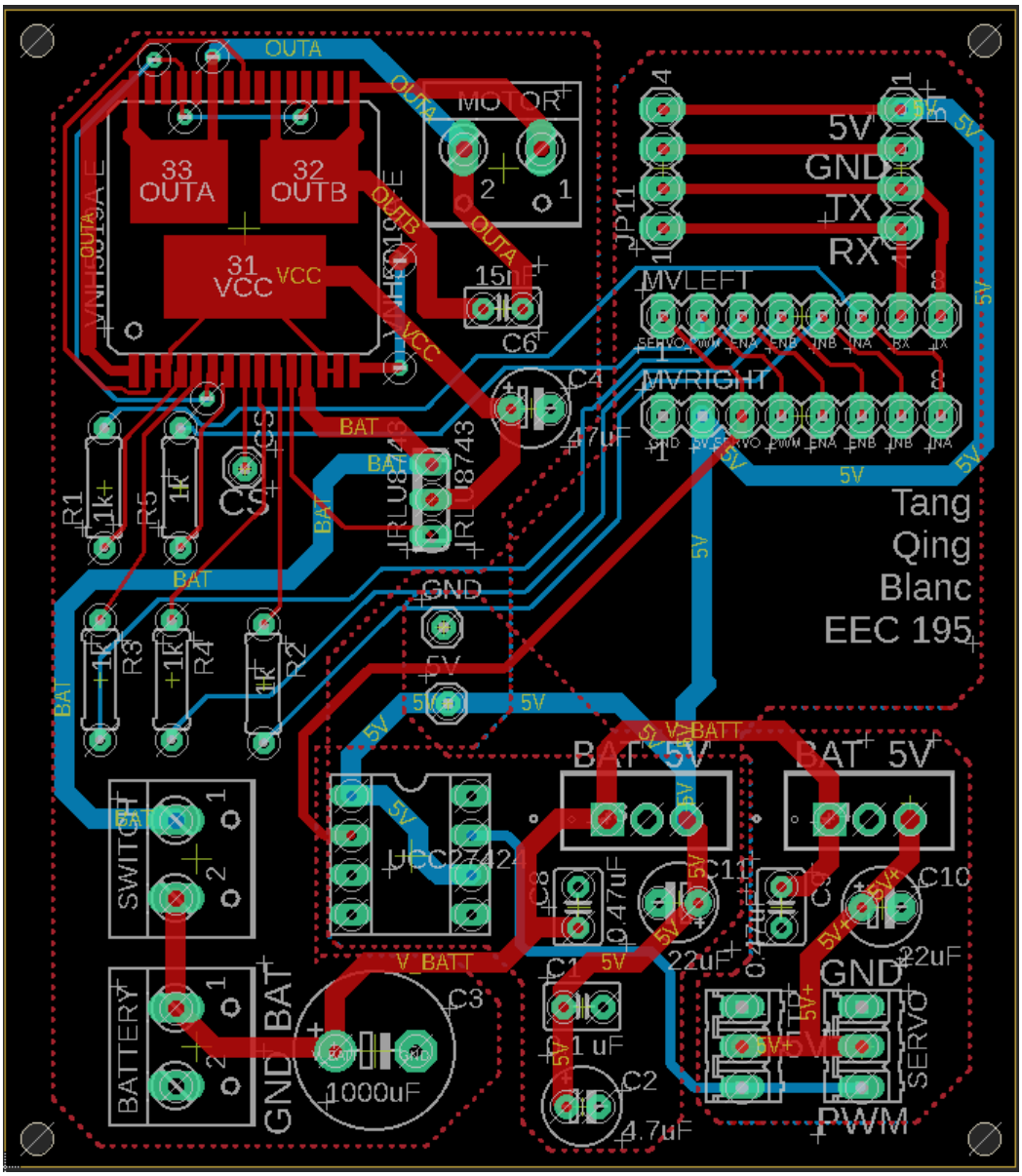

figure 7

Version 2 of our PCB:

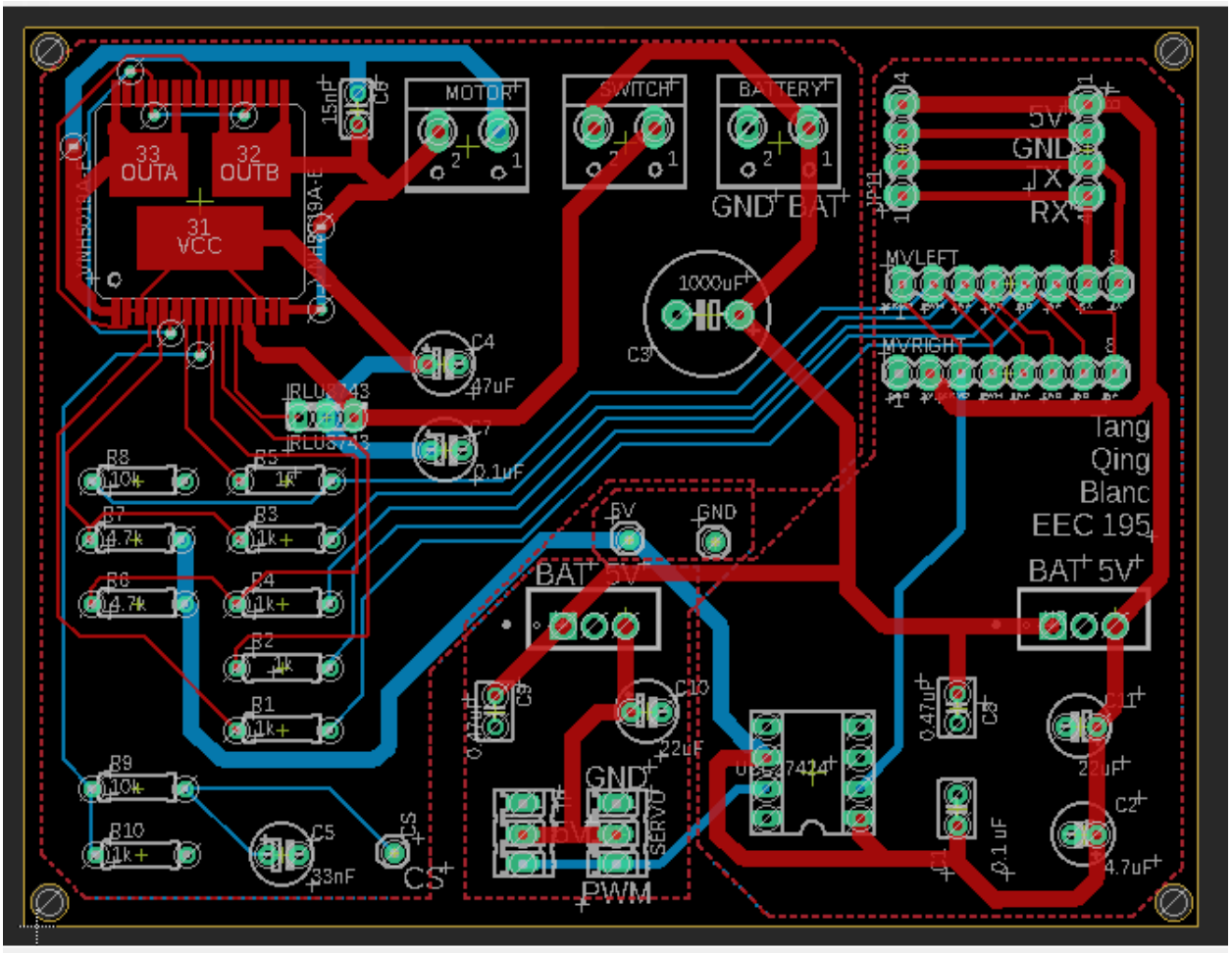

figure 8

Schematic:

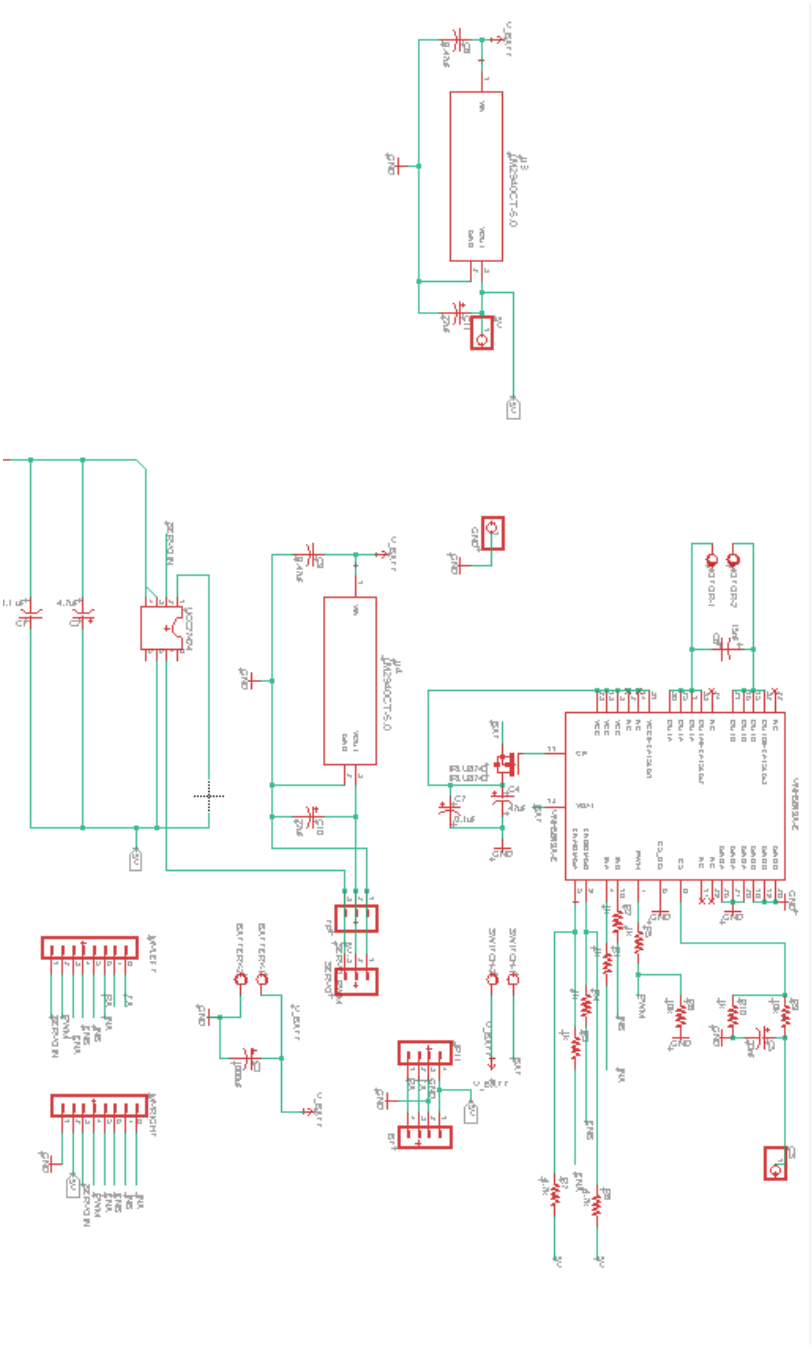

figure 9

Camera Holder First Design:

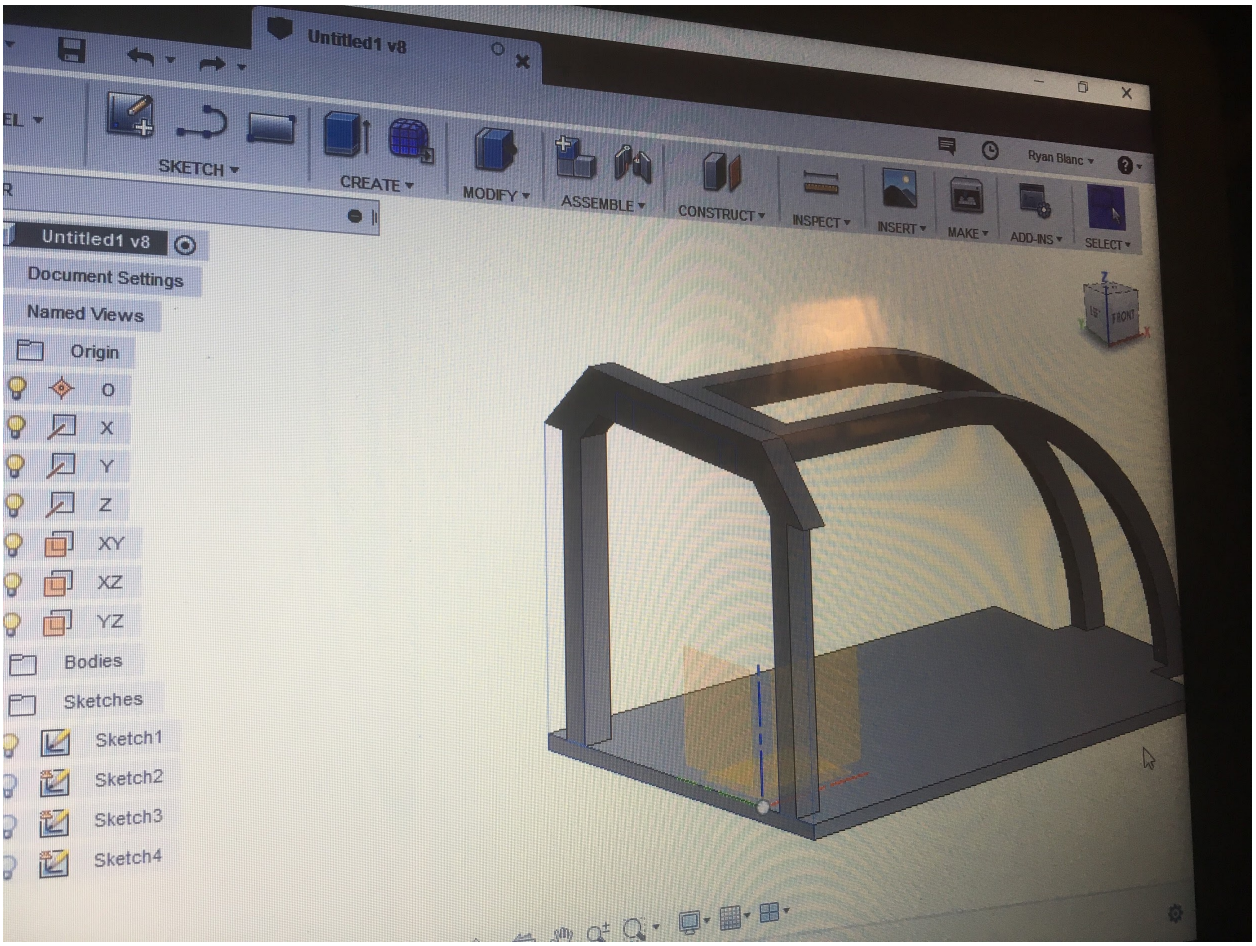

Figure 10

Camera Holder Final Design:

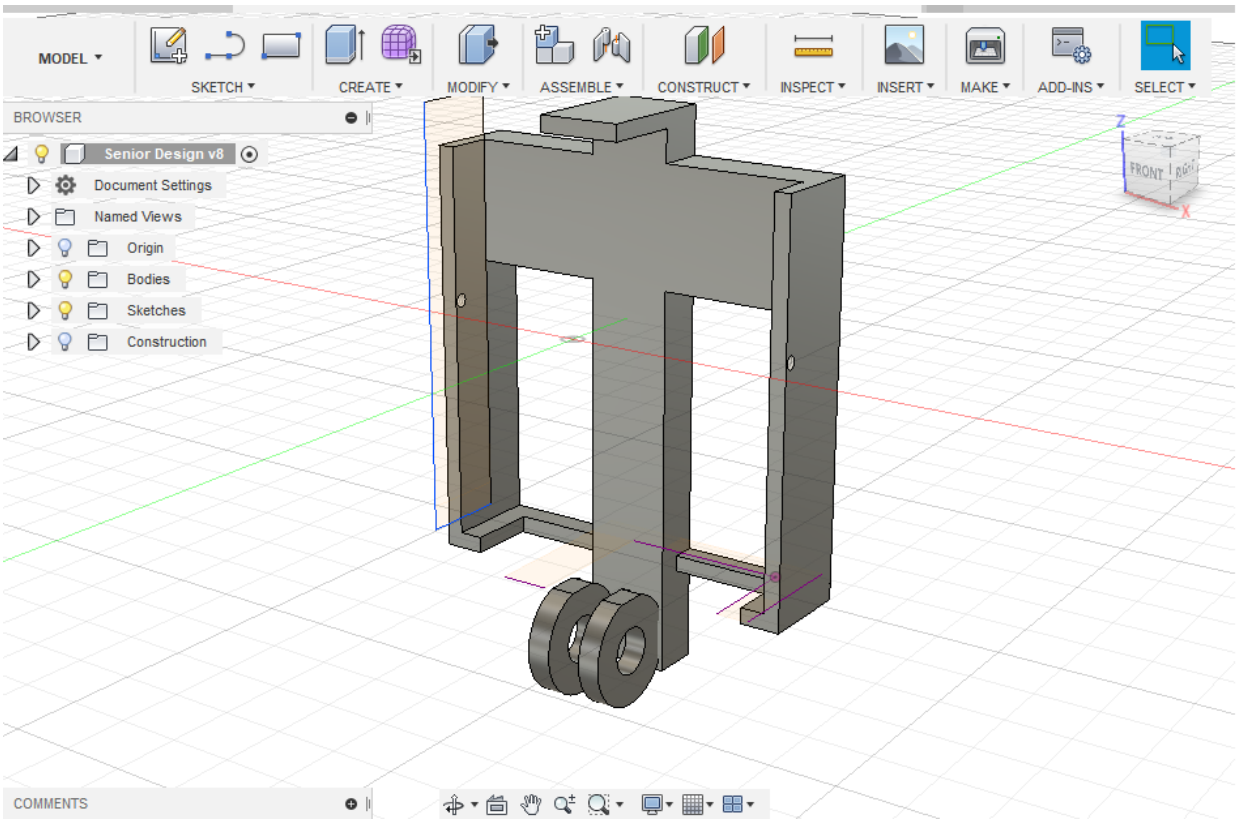
MODEL
SKETCH
CREATE
MODIFY
ASSEMBLE
CONSTRUCT
INSPECT
INSERT
MAKE
ADD-INS
SELECT
BROWSER
Senior Design v8
Document Settings
Named Views
Origin
Bodies
Sketches
Construction
COMMENTS

Figure 11