Final Report

Autonomous Vehicle Project

EEC 195A/B

University of California, Davis

Spring 2016/Winter 2017

by

Hardik Patel

Fadi Abdulhameed

Casey Hood

March 20, 2017

TABLE OF CONTENTS

Overview

NATCAR, the design, build, test and race of an autonomous vehicle with the goal of traversing a track the fastest. Our approach was governed by the goal of traversing a track. We had no intentions of making the fastest car, we just wanted to get it to work first.

Throughout the course we performed numerous lab exercises as a group targeted at helping us gain the tools to build such an autonomous car, but these tools, while extremely valuable, only went so far. The car quickly transformed into something much more than any of us had initially imagined. Our approach still remained the same however, just get the car to work.

After many hours of work, we were successfully able to traverse the track at 9.8 ft/s.

Task breakdown:

Hardik Patel - Software, Algorithms, Motor control

Fadi Abdulhameed - Motor Control Circuit design , Car assembly , Control theory

Casey Hood - PCB, Control theory, Motor control

The design was a cumulative effort of every member of the team. We hereby agree on an even 33 % split.

X______________________	X______________________	X______________________
Hardik Patel	Fadi Abdulhameed	Casey Hood

Technical Report - by Hardik Patel

In this section I will describe the software algorithms and choices that we made for our car’s software portion. For the software, I wanted to have a minimalistic approach and not clutter up the design by having code that didn’t achieve much. We also left a lot to our camera angle, i.e, for our final design to work completely we need to have the camera angle to a certain position, or else, it will impede performance of the car and it will not be able to traverse crossings and steps correctly. The final code structure is as follows:

1. ADC interrupt handler - Fills up ping and pong buffers and set the constants to keep track of the buffers.
2. Timer Handler - Triggers the ADC conversion.
3. A few helper function for UART etc.
4. Initializing the clock, timer, ADC, and PWM to the correct specifications.
5. Initialize all the constants for use in the while loop.
6. While 1 loop, where we analyze the last filled buffer, find the current index, perform P and D control, and set the servo direction.

To analyze the camera data, we had two options, voltage threshold and slope threshold methods. We tried our basic algorithm with both the methods, however, with the voltage method, we encountered a few performance issues. In theory both methods had equal performance, however, the line was harder to capture with the voltage method. The reason I wanted to use the voltage method was because it would be easier to analyze camera data with an array of 1s and 0s. Nonetheless, this proved to be unnecessary, as with the right camera angle, the car and higher speeds the car was able to traverse through steps and pretzels on its own without any software interventions.

We started implementing our control algorithm with a simple Proportional control. Initially at about speeds of 4-5 ft/s, our car handled the track acceptably just with the proportional term. We tested the car with K_p terms ranging from 0.2 to 2, and unexpectedly, higher K_p, even as high as 1.3, our car still managed to traverse the track completely, at low speed. As we started increasing the speed, with these higher K_p values, our car started over correcting itself and started swinging along the track. The final servo direction that we chose to pass into the PWMPulseWidthSet() function was also made compatible to the proportional values, which meant, that while implementing the Integration and the Derivative term, we would have to scale up those terms up to the values of the Proportional term to see any kind of effect.

To fix this issue we decided to implement the Integration term (however, we ended up not utilizing that term in our final run).Once the Integration term was implemented, we did not experience the benefits of having that term, because the performance was very similar to only having the proportional term.

We decided to implement the Derivative term next. For the derivative term, we decided to use a buffer of 4 at first. This proved very helpful, as, the term allowed aur car to traverse the track considerable faster, at 7-8 ft/s. I also implemented a buffer of the past 16 indices, however, I was not able to use it as I would have had to scale it down by a large factor, which turned out to be unnecessary. We tested different values of K_d and K_p and after many trials we decided to settle on 0.7 and 0.4 values respectively. At this point, with no speed, control we were able to traverse the track at 7-8 ft/s, without our car going off track.

The equations we used for P and D term were as follows (in code) :

```
                diff_in_index[errorcount%4] = 64 - index[errorcount%2];
                   pterm = ptermconst * diff_in_index[errorcount%4];

dterm = dtermconst * ((diff_in_index[errorcount%4] - diff_in_index[(errorcount + 1)%4]) +
      3*(diff_in_index[(errorcount + 3)%4] - diff_in_index[(errorcount + 2)%4]));
                                 finaldterm = dterm;
```

To increase speed we decided to slow our car down considerably on only sharp turns. We tried different methods of slowing down our car, based on the current index, or the current servo direction and more, however, with all these methods, the car failed to regain its speed to its maximum after a sharp turn. The final method we settle on, was to slow it down only if the servo direction increased beyond its maximum and minimum, in both directions. Using this method we were able to speed up our car considerably, to reach our maximum speed in the time trial.

However, at this time we were faced with an unusual problem. Our car would traverse the trial track perfectly, but only in one direction. While going right, on sharp turns our car would suddenly turn left and leave the track completely. The last two weeks were solely spent on fixing this issue. Another interesting aspect of this issue was, that this issue would only be fixed if we were travelling at much slower speeds, 4-5 ft/s. We tried a variety of different ways to overcome this issue. The two main algorithms we used were:

1. If the index is showing a number greater than 122 and less than 128 and if the next index jumps to a random number like 40 or 50, the current index would be set to the last index. The rationale behind this approach was, the index would never jum 30 or 40 indices in one iteration as it is still following the white line which does not jump change that abruptly.

2. Another method, derived from the first method, was, if an index is seen at position greater than 122, the only next position acceptable would be a position between, 115 and 122. In this way we for the algorithm to bring down the indices slowly.
3. The last method, and the least effective method we tried as just moving our camera slightly to the right. While this had good performance, it was unreliable as it would cause problem while traversing the pretzels.

In our final run, we used method 1 to get the highest speed.

After the time trial however, we realized a possible cause for this problem, which was, the camera frequency was quite high and that may have been causing problems with the last few pixels of the data, hence we got garbage values. Nonetheless, this theory is still untested, as we were able to perform well in time trial 1.

Technical Report - by Fadi Abdulhameed

In my portion I will discuss the objectives for each of the implementations, and the approach I took to achieve them. Also, the reasons I decided to approach every task a certain way and the tradeoffs that came with each approach.

First of all, my main focus was on the analog part of the design (motor control circuit) and the car assembly. I was very involved in the circuit design. I also did some coding, but not as much as the rest of the parts. I attended the workshop for eagle and learned how to design the PCB, from laying out the schematics to routing the wires and establishing the main grounds and the most efficient components layout to reduce the noise.

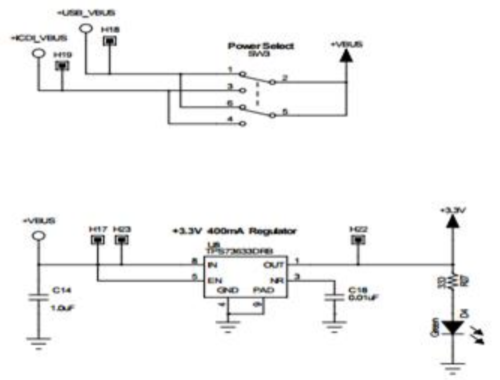
I assembled the car and built the outer body for it. In this process I focused on few factors such as the weight factor and the camera position to get the best performance possible. In addition, as far as the body of the car, I made sure not to use any tape in sensitive places to mount the main parts, and maintained a simplistic yet clean look of the body. I made sure to not use long jumper wires that could get in the way when the car runs at high speed. Also, made sure the PCB, and the battery are mounted very tightly, so they do not move when the car is running and turning which would cause it to lose power. The height of the camera was an important factor of the design since it is the main source the car gets its data from to analyze, so the position needed to be as accurate as possible. Since we had to run the car multiple times during the testing stage, the camera mount needed to be very sturdy so it doesn't fall off the car in case of a collision. The final product of the car is in Appendices Figure (1).

For the motor control design. We needed to make a choice to which circuit works most efficiently for our car and after Casey and I studied all the possibilities provided in class we decided to use the H-bridge since it allows the motor to run in both directions forward and reverse due to the fact that it has 4 transistors two of which control the forward flow of the current and the other two reverse the direction of the current which causes the motor to spin in both forward and reverse directions. This method adds more flexibility to the motor control and it is very efficient to implement. However, there were some tradeoffs to picking the H-bridge such as the complexity of the chip, and the fact that it has multiple inputs and outputs that we needed to keep track of.

<u>The Power Inputs</u>

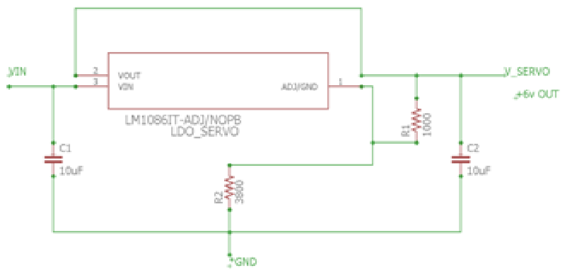
The main control unit in the car is he microcontroller which controls the motor, servo, and line-scan camera. The microcontroller is powered via VBUS with a 5 Volts signal that is connected in the bottom left corner of TIVA board. When the 5 Volts signal is generated, it is passed through a 3.3 Volts LDO to drop the voltage down to 3.3 Volts. The VBUS pins can be powered directly from the battery and is controlled with a switch to power the car on and off.

Microcontroller VBUS Power Schematics



For our design we specifically used the LDO (LM1086CSX) regulator to the servo to make sure that in case we encountered a voltage drop, the output would always be a constant at 6 Volts. In addition, the motive behind using this LDO is to regulates the output voltage and keeps constant voltage against temperature and current rapid changes, which makes it a more efficient choice than a normal voltage regulator for the servo because the servo will have faster and higher torque for higher VDD supply.

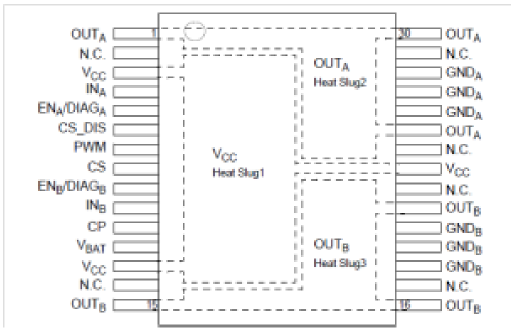
LDO Schematics



We chose the (LM1086CSX) LDO. This LDO choice was most efficient for few reasons. First, it has a 10 uf bypass capacitor that prevents ripples from being amplified as the output voltage is increased which is a good way of keeping the signal from getting too much noise because the higher the capacitance, the larger the available charge per demand. Second, from a conceptual approach in theory looking at it from the frequency domain perspective, the series resistance connected to ground introduces a zero within the loop which increases the phase margin and thus increases stability.

The control unit:

The main control unit in our PCB is the H-bridge chip (VNH5019A-E)



In this part of the design the objectives was to have as much of flexible control as possible. In other words, we wanted our motor to be able to run in both directions and have breaking mechanism available as well. We intended to achieve these goals by implementing this full H-bridge chip which has two high side drivers and two low side switches. Moreover, it has multiple inputs which can take signal inputs such as INA and INB which through the microcontroller can select the direction of motor rotation, and also breaking. We didn't implement the breaking mechanism so we set INB low and INA high at all times since we only did the clockwise operation mode. As well as ENA and ENB were set high as well. In other words, we only enabled one leg of the H-bridge chip. This chip also handles PWM input signal up to 20 KHz which can drive the motor in all possible speeds besides breaking and coasting options.

<u>Coding portion for data processing through the line-scan camera</u>

We used the line-scan camera to grab the data necessary for the car to run. The camera grabs 128 pixels and stores them in a circular buffer (ping-pong) and every segment of 128 pixels has one SI pulse which controls the integration time. The CLK signal works as a shift register which shifts each pixel voltages to the AO output.

We used the Slope method in analyzing our data. We basically calculated the slope at each pixel (current value of pixel) minus (past value of pixel) divided by two which is equivalent to taking the derivative of the slope at each pixel location. Since the track is white it peaks at positive slope and slopes down at a negative slope forming a spike shaped graph. Even though we were successful in implementing this method it has some tradeoffs since the working mechanism is based on searching for the mins and maxs peaks then the main peak of the white pixels data could be mistaken with the peak of the noise and be assigned as the max if too much noise is introduced.

I wrote the functions for finding the maxs and mins using pointers.

```
int findMax(volatile int *arrayPtr)
{
    int Maximum = arrayPtr[0];
    int i =0;
    for(i =1; i <= 127; i++)
    {
        if(arrayPtr[i] > Maximum)
            Maximum = arrayPtr[i];
    }

    return Maximum;
}


int findMin(volatile int *arrayPtr)
{
    int Min = arrayPtr[0];
    int i =0;
    for(i =1; i <= 127; i++)
    {
        if(arrayPtr[i] < Min)
            Min = arrayPtr[i];
    }

    return Min;
}
```

PID stability control methods

I have taken few courses in control systems and I have good understanding on the topic. The motive was to get the car as stable as possible and reduce overshoot and oscillation when the car goes off the track and when it encounters sharp turns.

I recommended using the PD controller because in theory adding a PD doesn’t change the system type (it only adds a zero which in theory makes the system faster) and improves the damping (e.g. reduces overshoot, improves PM and GM) , so it seemed to be a more efficient choice. I didn’t recommend using the PI controller even though it works great in theory in which it improves damping as well, but it has some tradeoffs some of which is the integrator windup which can cause the system to oscillate. So in theory PD seemed more appealing to our design. However, the real life implementation didn't work as close as the theoretical part suggests due to the fact that we don't have a real mathematical representation of our system (NO Transfer Function) to work with, so the ideas still applied, but it wasn't as precise as the theory suggests. Therefore, in practice, we did implement the PD and PI and tried all different kinds of combination that could possibly work best, and we did a number of runs with trial and error while changing the constants values every run to figure out the most stabilized method. The final, product resulted in a very stable system.

Technical Report - by Casey Hood

The design of the PCB was a challenge in that it involved determining components and developing an overall layout of a board that none of us had ever built. I initially knew that I wanted to future-proof our design so that we would never max out our board's capabilities. With that in mind, I will begin by discussing the headers, as they are the most basic, but essential building blocks of the board.

Beyond the necessary headers such as Tiva headers, linescan camera header and the servo header, we opted to include an optional header for a Bluetooth module. Including this header in the design would allow us the capability of monitoring our car's data, changing constants and having a safety shutoff button. One thing we neglected on including in our design was adding an additional header for another linescan camera. This would allow us to further revise the way we analyzed our data and determine how the car should react upon certain situations.

Another unusual header we added was our Tiva power supply header. The Tiva itself can be powered directly by its VBUS pin, but we opted to solder a header directly to the bottom of the Tiva board and one to the PCB. The idea behind this is slightly flawed in that our initial goal was to prevent our Tiva board from being powered from two sources at once. There is the possibility for shorting the USB power connection when powering the the Tiva board with the PCB and the USB cord. We could avoid this problem altogether by simply disconnecting the jumper wire that powers our Tiva board from the PCB before we connected the Tiva via USB. However, the shorting of the USB power connection could still happen if we forgot to disconnect the wire. A simpler fix for this issue is to simply switch the power selection switch on Tiva to device mode while being powered via the PCB.

After determining what we wanted to be able to input and output from our board, we then began to decide which type of motor control to use. Fadi and I went through the pros and cons of each as follows. The single DMOS circuit was very simple and would be able to provide us with a very stable motor controller that would be easy to implement. We already knew we weren't going to implement braking in our software so the single DMOS plus braking wasn't a consideration for us. I argued that although the H-Bridge was quite complex and most likely overkill for our needs, it would provide us with a future-proof system. It would allow us to have forward, reverse, braking, and free-wheeling. This was much more than we ended up using in the end, since our speed control was done by simply varying the PWM sent to the H-Bridge if we detected a turn of certain radius. In the end, we opted to use the VNH5019 H-Bridge for our motor control. Any high current drawing elements are placed on one side of the PCB and are separated by the use of two ground planes connected through a small channel to prevent noise interference to camera data.

The actual wiring of the H-Bridge quickly became a huge challenge due to the issue that there is not a library with the CAD project of this H-Bridge available for Eagle yet. I was able to use a sister board schematic (VNH3SP30-E) that luckily had the same pin configuration but different pinout. I was then able to mirror our design of the schematic posted on Canvas with some slight modifications. The two enable pins, ENA/DIAGA and ENB/DIAGB, were driven high so that the full H-bridge was always active, since we wanted to ensure that we could still use the full bridge. Pins INA and INB are connected to two separate GPIO pins, PF2 and PF3, on the Tiva board. If we wanted the car to only move forward, clockwise direction, we could drive INA high and INB low as seen from the truth table below in figure 1. The PWM pin is driven by a single GPIO PWM enabled pin, PC4. All pins labeled OUTA and OUTB are then connected together accordingly and connect directly to the motor. A small capacitor of about 10nF is connected in parallel to help alleviate some of the commutation noise produced by the DC motor. The CS_DIS pin was driven to ground as we are not using current sense feature and as a result the CS pin is left floating as we are not using it. The pins labeled NC are not connected and the pin VBAT is connected to a 10A toggle switch that is then in turn connected directly to the battery with a large capacitor in parallel, ~500□F. We also chose to take the risk of not including a reverse battery protection diode. We figured that once everything was soldered on correctly, the battery connector already contains a key that prevents the user from reversing the terminals accidentally.

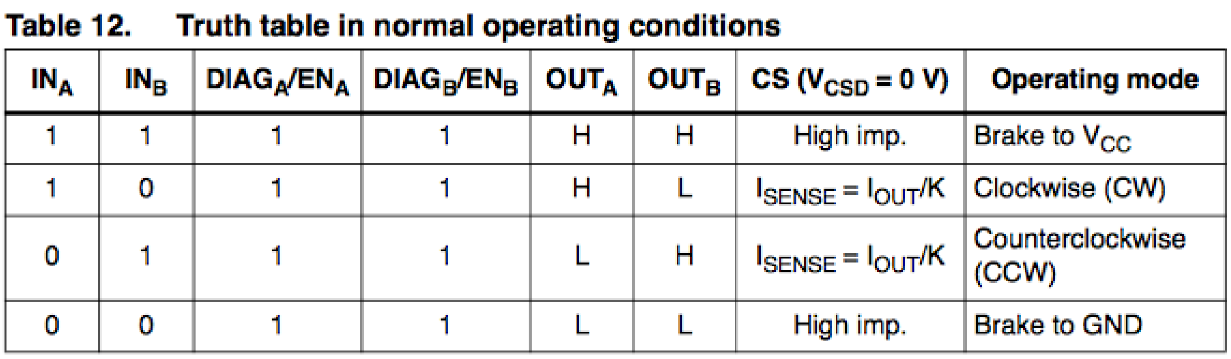


Table 12. Truth table in normal operating conditions

IN_A	IN_B	$DIAG_A/EN_A$	$DIAG_B/EN_B$	OUT_A	OUT_B	CS (V_{CSD} = 0 V)	Operating mode
1	1	1	1	H	H	High imp.	Brake to V_{CC}
1	0	1	1	H	L	$I_{SENSE} = I_{OUT}/K$	Clockwise (CW)
0	1	1	1	L	H	$I_{SENSE} = I_{OUT}/K$	Counterclockwise (CCW)
0	0	1	1	L	L	High imp.	Brake to GND

Figure 1: Truth Table for VNH5019

We opted to use two adjustable LDOs (LM1086A), low dropout voltage regulators. The reasoning behind two separate LDOs is because these LDOs can only supply up to 1.5A out, we did not want to risk the servo from pulling too much current such that we lost power momentarily to other crucial components such as the Tiva itself. With that said, the first LDO is adjusted to 5V to power our Tiva board, Bluetooth module, and the DMOS driver for the servo input. The second LDO is adjusted to 6V to power only our servo. The corresponding circuit (figure 2) to the 5V and 6V LDOs was then computed from the schematic in the LM1086A data sheet.

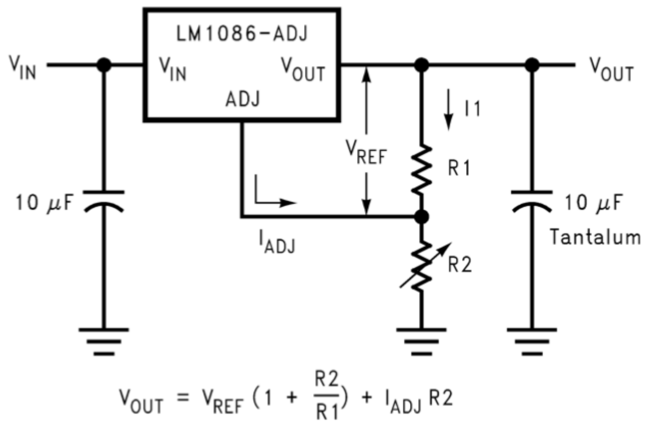
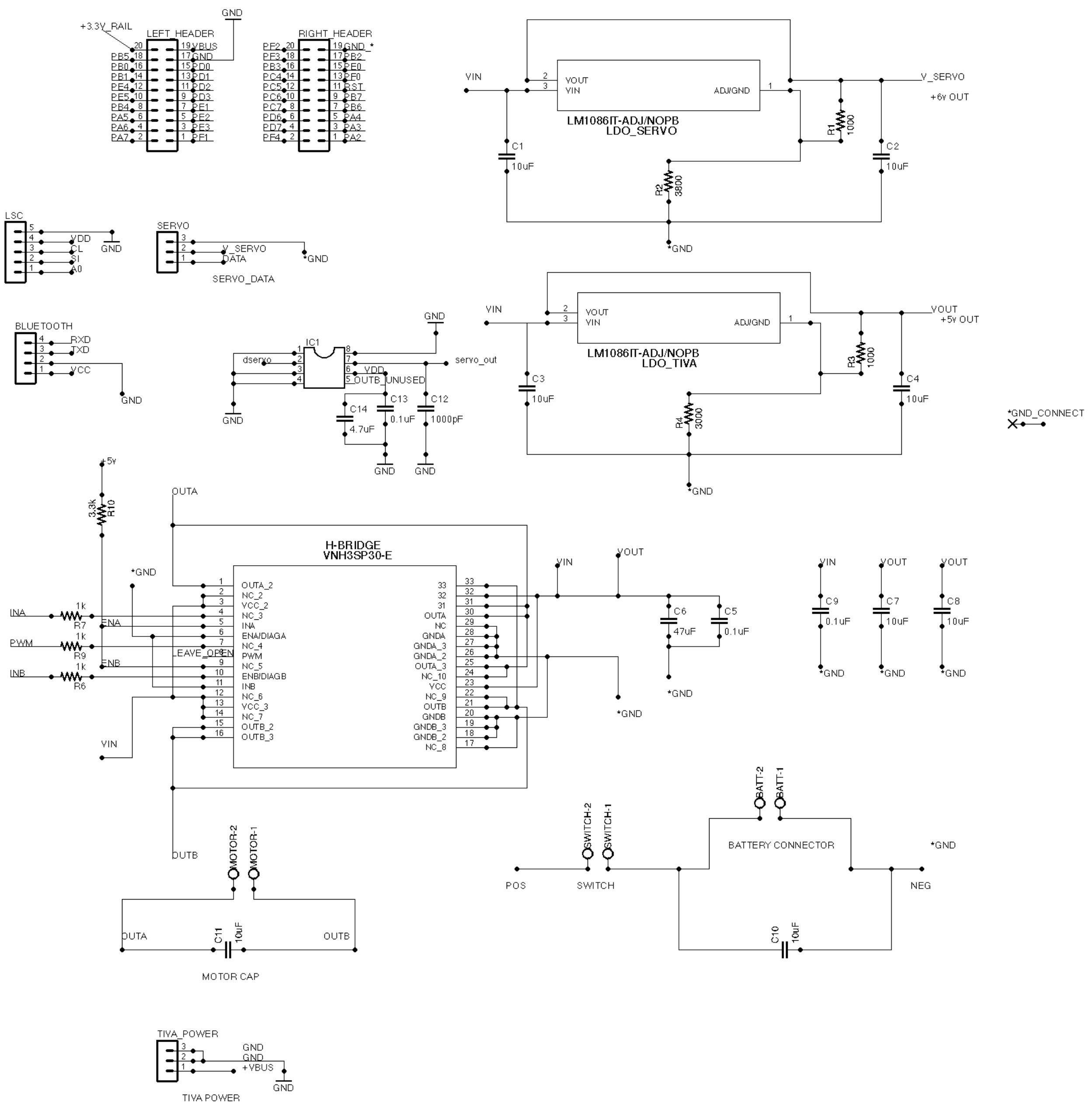


Figure 2:Resistor Calculation LDO LM1086

After receiving our PCB back from Bay Area Circuits, I then soldered on all the components and began testing. It quickly became apparent that we were having power issues with our board. When powering via the battery, our Tiva board would shut off and on; this was apparent due to the green LED flickering. However, when powered via a bench power supply, the power supply would jump into constant current mode if we lowered the current knob too low. We were able to resolve this by changing our resistor values we initially calculated for the 5V LDO. In the datasheet for the LM1086A, it requires that resistor R1 be around 100 Ω. We had initially used a value on the order of 1KΩ. After changing these values out with the correct values, we received much better results in a constant voltage. At the same time as the power issue, we developed an issue where the H-Bridge was getting very warm when it was connected to a power supply, however, the H-Bridge was still performing as we intended. After reading the datasheet for the VNH5019 H-Bridge, it became apparent that we are actually enabling current sensing by driving to ground instead of disabling like originally intended. Although this is most likely not the root cause of our heating issue, it is something that we would like to change on a revision.

To the utmost surprise, there were not as many issues as I had initially planned for, however, with that being said there are a few things I would revise. When plugging in the USB connection to TIVA, the connector rests on top of one of the LDOs. This causes there to be a lot of pressure applied to the Tiva board connector and as a result breaking connectors. I would also use a 5V LDO next time instead of an adjustable LDO. This makes things much simpler with fewer components needed on the board.



GND
+3.3V_RAIL
LEFT_HEADER
VBUS
GND
PB5
PB0
PB1
PE4
PE5
PB4
PA5
PA6
PA7
PD0
PD1
PD2
PD3
PE1
PE2
PE3
PF1
RIGHT_HEADER
PF2
PF3
PB3
PC4
PC5
PC6
PC7
PD6
PD7
PF4
GND_*
PB2
PE0
PF0
RST
PB7
PB6
PA4
PA3
PA2
VIN
VOUT
ADJ/GND
LM1086IT-ADJ/NOPB
LDO_SERVO
V_SERVO
+6v OUT
R1
1000
R2
3800
C1
10uF
C2
10uF
*GND
LSC
VDD
CL
SI
A0
GND
SERVO
V_SERVO
DATA
*GND
SERVO_DATA
BLUETOOTH
RXD
TXD
VCC
GND
IC1
dservo
servo_out
VDD
OUTB_UNUSED
C14
4.7uF
C13
0.1uF
C12
1000pF
GND
LM1086IT-ADJ/NOPB
LDO_TIVA
VOUT
+5v OUT
R3
1000
R4
3000
C3
10uF
C4
10uF
*GND
*GND_CONNECT
+5v
3.3k
R10
OUTA
H-BRIDGE
VNH3SP30-E
*GND
INA
PWM
INB
1k
R7
R9
R6
ENA
ENB
LEAVE_OPEN
VIN
OUTA_2
NC_2
VCC_2
NC_3
INA
ENA/DIAGA
NC_4
PWM
NC_5
ENB/DIAGB
INB
NC_6
VCC_3
NC_7
OUTB_2
OUTB_3
OUTA
NC
GNDA
GNDA_3
GNDA_2
OUTA_3
NC_10
VCC
NC_9
OUTB
GNDB
GNDB_3
GNDB_2
NC_8
VIN
VOUT
C6
47uF
C5
0.1uF
*GND
VIN
C9
0.1uF
VOUT
C7
10uF
VOUT
C8
10uF
*GND
OUTB
MOTOR-2
MOTOR-1
OUTA
C11
10uF
OUTB
MOTOR CAP
BATT-2
BATT-1
BATTERY CONNECTOR
SWITCH-2
SWITCH-1
*GND
POS
SWITCH
NEG
C10
10uF
TIVA_POWER
GND
GND
+VBUS
GND
TIVA POWER

Safety

We had a few issues with the car that may be deemed unsafe. Even though we had a bluetooth compatible PCB, we did not implement that module. In all our testing, we only had a handful of instances where the car went off track and crashed. Most times we were able to catch it and turn it off.

The main safety issue we had was with our H-Bridge getting too hot. Although we had an idea of why the chip was getting hot, we were unable to pinpoint the exact cause.

Another issue was the batter getting warm, however, it may have been the long continued used of the battery that was warming it up, rather than an issue with the battery itself.

As far as safety with the TIVA board. It has a main power select switch, SW3, that selects the source of the power supply, which could be either through a USB input or a 5 Volts power source such a battery which connects through the VBUS line. The switch provides safety to the microcontroller in which it prevents the TIVA board from being powered by both USB and VBUS at the same time.

Instead of implementing the bluetooth module, we used a mechanical switch with a flag attached to shut the power off to the car instantly.

Design and Performance Summary

Time Trial I: 9.8 ft/s (18.8s)

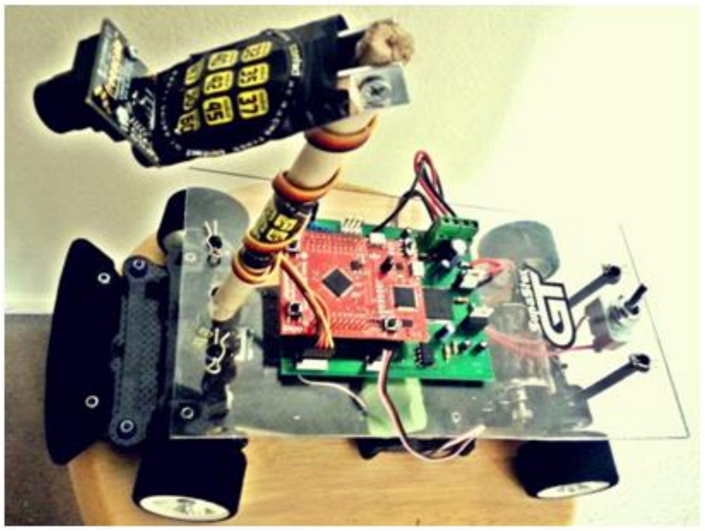
We were able to get very close to our goal of 10 ft/s, which is what we set out to accomplish. We learned the use of various tools, such as Code Composer Studio and EAGLE cad softwares, and the use of embedded systems and the integration of hardware and software.

If we were to have the chance of doing our senior design project again, we should learn the different properties of the tools and parts used by us in greater depth. If we had done this, we could have avoided problems like the high frequency of the camera.

The H-bridge choice worked great for our design, but if we were to redo this design , I would suggest enabling more features since it is capable of handling more than what we needed to use. However, the basic mechanism of the H-bridge allowed our whole system to have more robust and precise outputs.

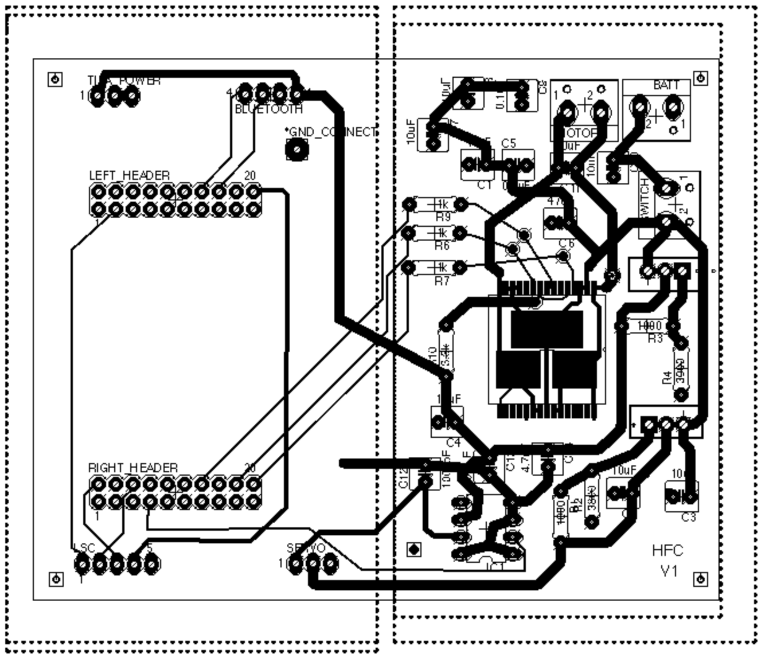
Appendices

The Final Product - The Car aka HFC



<u>Figure (1)</u>

The Final Product -PCB board



TIVA_POWER
BLUETOOTH
*GND_CONNECT
LEFT_HEADER
RIGHT_HEADER
LSC
SERVO
BATT
MOTOR
SWITCH
R9
R6
R7
R3
R4
C1
C3
C4
C5
C6
C12
10uF
HFC
V1