NATCAR Project Report

Fall 2014 – Winter 2014

Sijie Lin

Zhisen Qian

Ruixiang Li

Guangliang Wu

Project Overview

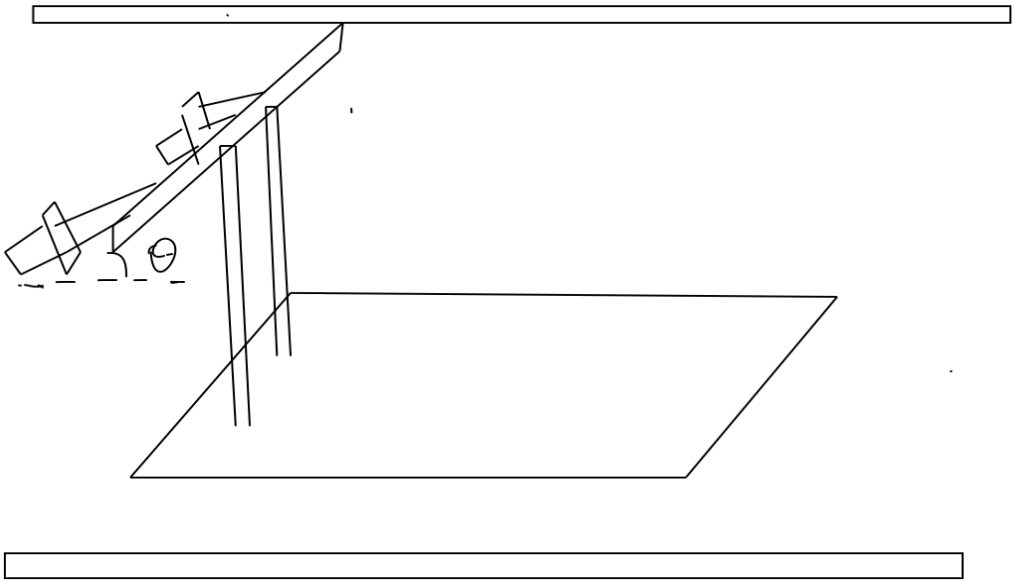
After numerous discussions, our team concluded that the key to have a good performance on the track, despite a well-designed steering algorithm, is to ensure that both cameras must be able to consistently collect good data in order to keep the car in the center of the track. Moreover, in order to ensure the car can react to the new servo PWM value on time with a faster motor speed, the cameras must see far enough so that it can balance out the servo response time. It's under this idea that we design the car as it is(put a picture maybe ?). Once the design of the car is settle, team members who are responsible for the programming would then carry on to write the program based on the received data and make any necessary suggestions to change the car design along the way in order to minimize the complexity of programming. As soon as we passed the second check point, we then started to design the custom motor control PCB for the car.
Below is a task break-down for each member:

Name	Task	% Effort	Signature
Guangliang Wu	Chassis Design Custom PCB	25%	
Zhisen Qian	Chassis Design Custom PCB	25%	
Sijie Lin	Program, Chassis Design	25%	
Runxiang Li	Program, PCB	25%	

Detailed Technical reports:

Classis Design: (Writer: Guangliang Wu)

At the beginning of this quarter, we need to set up our car first in order to do the following coding testing. The most important part of classis design is the camera setup. We decided that use two cameras in front of the car so that they can clearly detest two tracks. The beginning of our car design is shown below.



According to our tests, we figured out that the angle, length, and height of camera are mainly effected our car design. We changed our car design many times because of these three factors.

Before the checkpoint one, we tried to follow the competition rule. We used the length between the two cameras is 25cm, and height of cameras is 30cm, and θ is about 30 degrees. We wanted left camera control the right turn and right camera control the left turn. In the other words, as our car was closer to the left track, the servo of the car turned a larger angle to the right. Using this design, our car can run in the straight line very well, but it

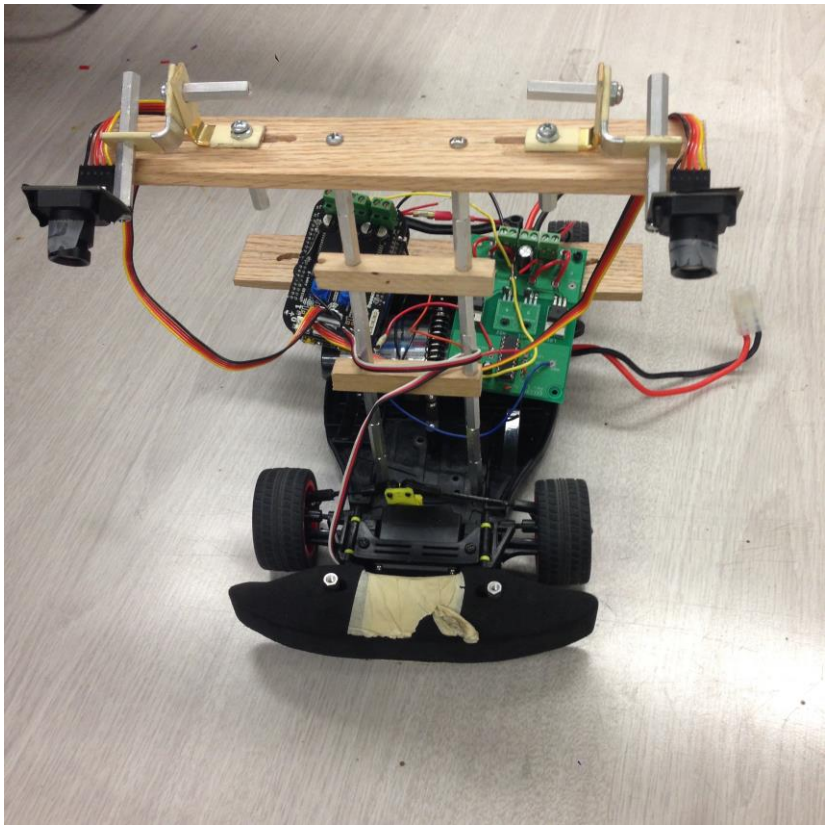
cannot make a turn. After our tests and discussion, we found out the reason why our car cannot make a turn. When the car was making a turn, one camera detested two tracks at the same time, so the car goes in a straight line instead of a turn. To fix this problem, we decided to make two cameras more width so that one camera cannot detest two tracks at the same time. Therefore, the car successfully passed the checkpoint one.

Before the checkpoint two, we tried to make the car run faster. Two hills are added to the track of the checkpoint two. When we used the same thing in the checkpoint one, the car cannot pass the hills. The servo made a turn all the way to the right or left. However, we found that the car at the top of the hill detested more forward than we expected. We fixed this problem in two ways. First, we moved the cameras from the front of the car to the middle of the car. Second, we decreased the angle θof the cameras. After we finished these two steps, the car also successfully passed the checkpoint two.

Before the competition one, although our car can make a turn with a slow speed, it cannot stably make a turn with a high speed. To fix this problem, we tried many different values of these three factors. When the angle of θ is increased, the servo will not turn enough so that it runs away the track. When the angle of θ is decreased, the car will not pass the hill. So we realize that we need to fix this problem from coding. At last, our car still cannot stably make a turn after downhill in order to our car cannot pass the competition one.

To prepare the competition two, we decided that the design of our car would not change any more. We chose the angle θ is 30 degrees, the width of the cameras is 23cm, and the height of the cameras is 28cm. After improving our coding, our car can finish the competition two with 34 seconds.

Beside the cameras setup, we put a bumper in front of the car so that it is more safety. We set the servo pwm is 4500 as middle position and make sure the front wheels is straight as same as the back wheels. Due to we burned out two lz25 boards, we put the pcb boards above the mood. In brief, our car design is base on our coding. The final car design is shown below.



Program Description:

Track Detection: (Writer: Runxiang Li)

The objectives of this section is to use a line detection algorithm to detect the edges of the track with the data from each camera. There is no denying that it is very important to analysis those data from each camera because that algorithm have a significant impact on our Steering Control. In our design, we use voltage method and slope method in the same time.

In general, the slope method is the best way to detect the edges of the track, because, in comparison to voltage threshold method, the data collected via slope method does not get distorted easily due to the light intensity of the environment. This advantage allows us to avoid the necessity to adjust the threshold reference values too frequently. Moreover, using the slope method would allow us to turn down the PIT frequency a lot, as long as it satisfies the minimum time value. To implement the slope method, for every buffer that is filled with 128 newly collected pixel data, the void Slope() would do:

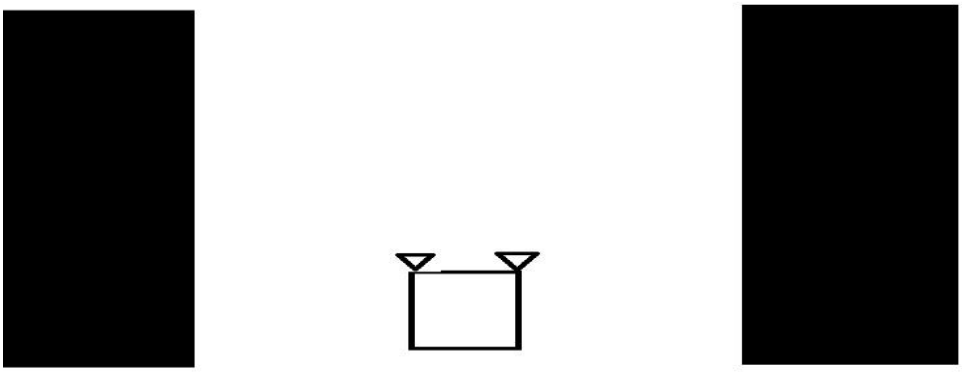
$$slope = buffer[i+1] – buffer[i-1] / 2$$

The result would then be compared with a slope threshold value which we measured to be 0x0D as the maximum threshold value. After the comparison, a new buffer track[] would be reconstructed with 0’s and 1’s, where 0’s should represent the position of the track.

In fact, based on the environment in which we use in competition, we still need to know the environment nearby the track: where the color is black or white because this can let us know the intersection track easily. In the intersection track, the slope method can only tell us that there is no track, but when the car is closing to the turn track, the

slope method still tell us there is no track. Therefore, we need to use the voltage method to determine the track is turn (all back) or intersection (all white). To implement the voltage method, we only need to focus on when the slop function cannot find the slope, then we only chose one pixel data to compare the voltage threshold.

Another critical information we can gather from using slope method and slope is that whether the track position is detected from a black surface to white surface, from white to black, all white surface, or all black surface. As you can see on the figure below:



While on the straight lane, the left track is detected from a black surface to a white surface, the program can recognize this via a flag to tell it what situation it is dealing with. If it's from black to white, the slope would be positive, we then set the flag CAM0_slope_sign to equal to '1'; if it's from white to black, for example the right camera in this figure, it will pick up a negative slope because the line, thus the correspondent flag CAM1_slope_sign will be set to '2'. And if cameras see nothing, flags will be set to 0's. Therefore, with this information, we can systematically establish all the cases that we may encounter on the track inside our steering algorithm. Theoretically, we

will have 9 cases in total, and they are: CAM0_slope_sign =’1’ and CAM1_slope_sign = ‘0’,’1’,’2’; CAM0_slope_sign = ‘2’ and CAM1_slope_sign = ‘0’,’1’,’2’; CAM0_slope_sign = ‘0’ and CAM1_slope_sign = ‘0’,’1’,’2’. However, among these cases, when CAM0_slope_sign = ‘2’ and CAM1_slope_sign = ‘1’ is practically impossible. Therefore, we eventually have 8 cases in total.

Steering Control: (Writer: Sijie Lin)

For steering control, I implemented proportional and differential control so that the vehicle can adjust smoothly compare to using proportional control only. The general function for servo pwm are:

if only one camera collects data:

*1 servo_pwm = 4500 +/- pGain * (w1 or w2) + dGain * (avg1 or avg2)

if both camera collects data:

*2 servo_pwm = 4500 + (w1 – w2) * pGain * 0.5

Where 4500 represents the position when the servo is turning straight, avg1 and avg2 are differential values obtained from function void Differential(unsigned int* left[], unsigned in* right[]) which calculate the difference of the previous error and current error value for each camera, and then return it to avg1 and avg2. The pGain and dGain for the final version of the code were adjusted to pGain = 11 and dGain = 48 which we observed to have the least oscillation when the car was running on the straight lane. These gains will have to increase along with the increase of the speed.

As mentioned early, the detection method I chose will generate 8 cases total when the car is running on the track.

Case 1: CAM0_slope_sign = 1, which means that the left camera sees the left track. As for right camera, we then have the following three sub-cases:

Case1: CAM1_slope_sign = 1, where both cameras see the left track and it indicates that the car may be too close to the left, so it should turn right with a relatively large angle.

Case2: CAM1_slope_sign = 2, where right camera sees the right track. In this case, the car is most likely running on a straight lane, can thus equation *2 will be applied.

Case3: CAM1_slope_sign = 0, where right camera sees nothing. In this case, the car is most likely undergoing a right turn situation, and therefore, *1 is applied.

Case 2: CAM0_slope_sign = 2, which means that the left camera sees the right track and it indicates that the car may be too close to the right, and thus it's better to turn left with a relatively large angle.

As for right camera, we then have the following two sub-cases:

Case1: CAM1_slope_sign = 2, the right camera sees right track also, but since the car is likely being too close to the right track, we applied *1 here with the error data from w2.

Case2: CAM1_slope_sign=0, this case indicates that we lost the footage from right camera, therefore it's better to directly assign a large angle to servo pwm so that it can turn left.

Case 3: CAM0_slope_sign = 0, the left camera lost its footage.

As for right camera, we then have the following three sub-cases:

Case1: CAM1_slope_sign = 1, the right camera sees the left track which indicates that the car is too close to the left track, and therefore it must turn right with a large angle.

Case2: CAM1_slope_sign = 2, the right camera sees the right track which indicates that the car is most likely undergoing a left turn, and therefore we applied *1 here.

Case3: CAM1_slope_sign = 0, both cameras lost its footages, and therefore if the car is on the white surface, the program then set the servo straight.

Among these eight cases, where *1 is applied, in between pixel 35 and 115, I divided it into 3 zooms each with pGain increased by 10%. The idea was that I want the servo to turn more as soon as the car gets too close to the edge during a turn. Whereas, on the straight lane, the pGain value will maintain at default setting.

Circuits Design and PCB Layout (Writer : Zhisen Qian)

In our Natcar team, my responsibility was to design relevant PCB such as motor control circuit and linear scan cameras circuit in eagle.

The Design of the Custom Motor Control PCB Including Servo

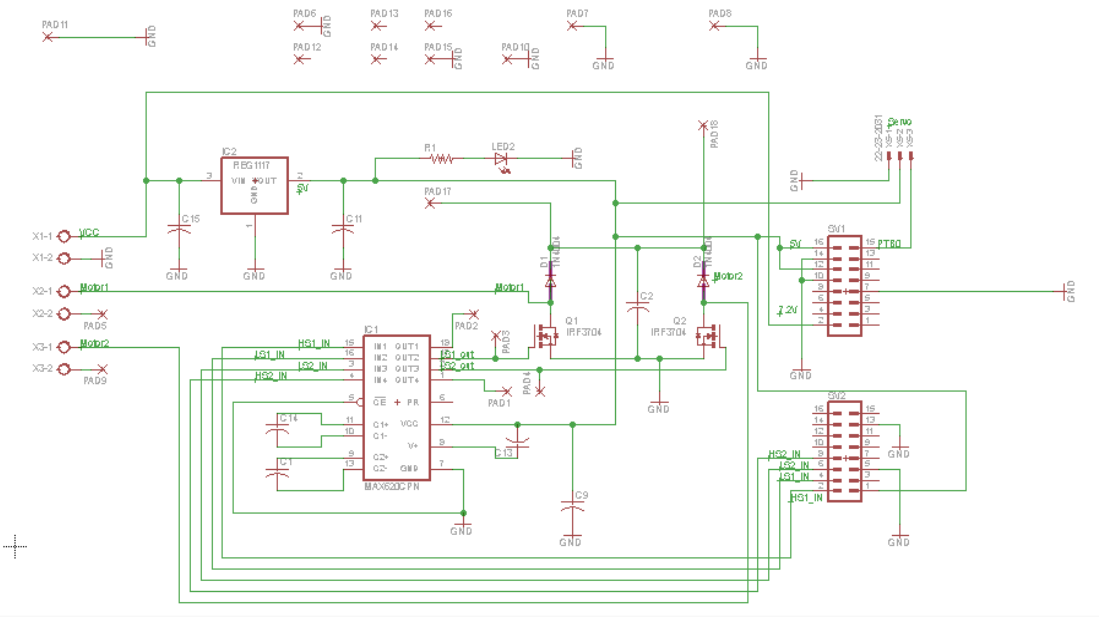


Figure 1.

The figure 1 is my motor circuit schematic. In my motor circuit design, I chose a MAX620 as our MOSFET driver and used an 800mA and 1A Low Dropout (LDO) 5V Positive Regulator (REG1117) to convert the 7.2V battery voltage to a 5 V. The MAX620 safety range of supply voltage is from 4.5V to 16.5V. I decided to supply the MAX620 a 5V supply voltage through the 5V regulator.

I followed the lab 8 example circuit to make my own design. Because we only use low side of MAX620 and set high side equal to 0, I used two simple diode 1N4001 instead of the two high side N-Channel Logic Level PWM Optimized Power MOSFETs.

These two diodes, 1N4001, were in parallel with the two motors so that to protect the motors.

In addition, I put 3 pins connectors to connect the servo on this PCB. Because the servo safety voltage range is 6.0 – 4.8V, I gave the servo a 5V power supply by my regulator. I used two 16 pins connectors to connect our KL25Z. For example: the pin 2 on SV1 was a 7.2V power supply to give the KL25Z a 7.2 VCC. I also put some led to test if the regulator and the circuit worked.

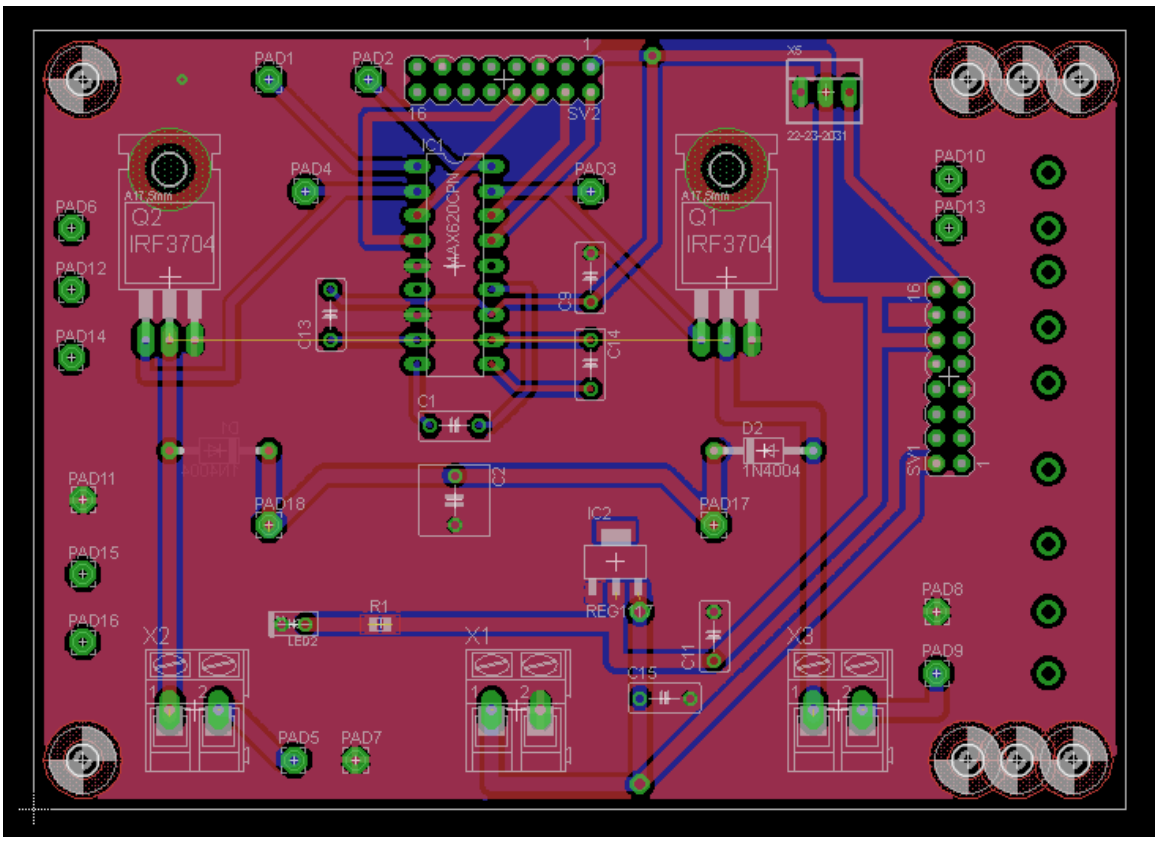


Figure 2

Figure 2 is my motor circuit PCB layout. I felt that the difficult part for this design was making the width of the wire wider. Firstly, I made the width of the wire was 16mil but I found the wire was too tiny to work well because the motors need to a large current to drive. Therefore, I used the wire was 60mil, and it worked well.

The Design of Camera PCB Including Switch and Potentiometer

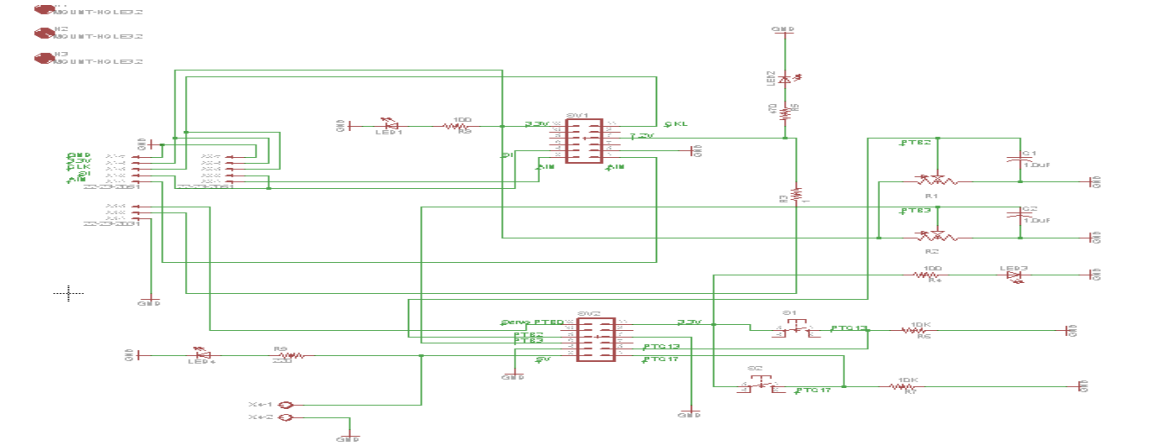


Figure 3

The figure 3 is my camera circuit schematic. I used both of motor circuit PCB and camera circuit PCB instead of the TFC shield. Because of the cameras needed a 3.3V power supply that came from the KL25Z, I used two 12 pins pin headers to connect the KL25Z. Then, using some led tested the circuit working.

The difficult part for this PCB was to find some right chips from eagle library. For example, the pin header to connect the camera was called “Center Header 5-pins”. It took me a lot of time to find the correct one.

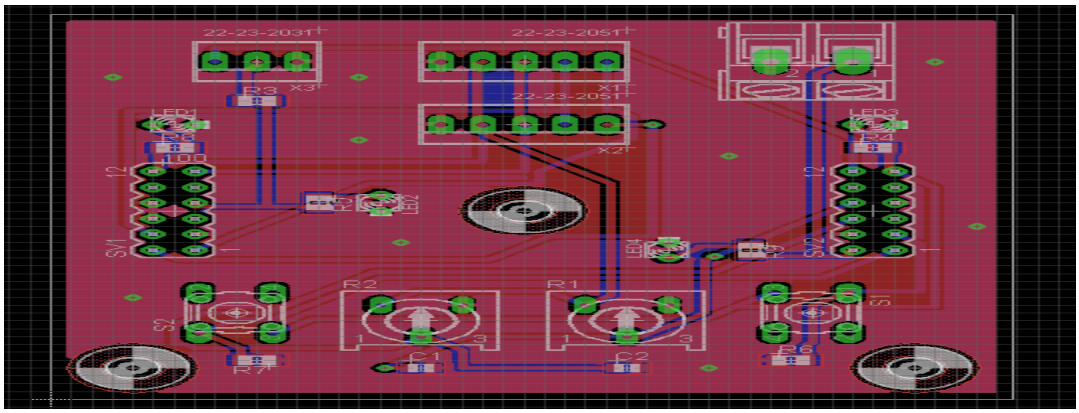


Figure 4

The objective of my design PCB was to replace the whole TFC shield. In fact, our two PCB worked well. After comparing the MAX620 MOSFET high side drivers and the MC33887 H-Bridge, I decided to use MAX620 because the MAX620 is easier to set and do not need to consider the issue of emit heat. The MC33887 is more complex because I need to consider the heat issue and non-inverting buffer, but its function is more comprehensive than the MAX 620.

The reason I did not put the camera PCB and the motor PCB together is the limit size of a PCB in Eagle is 100mm x 80 mm, and one board is always easier to make errors. Making two separate PCBs can easier to find errors and debug. To more effectively debug my PCB, I also made some holes on these board so that I could connect some pins in wires.

2D Camera PCB Design

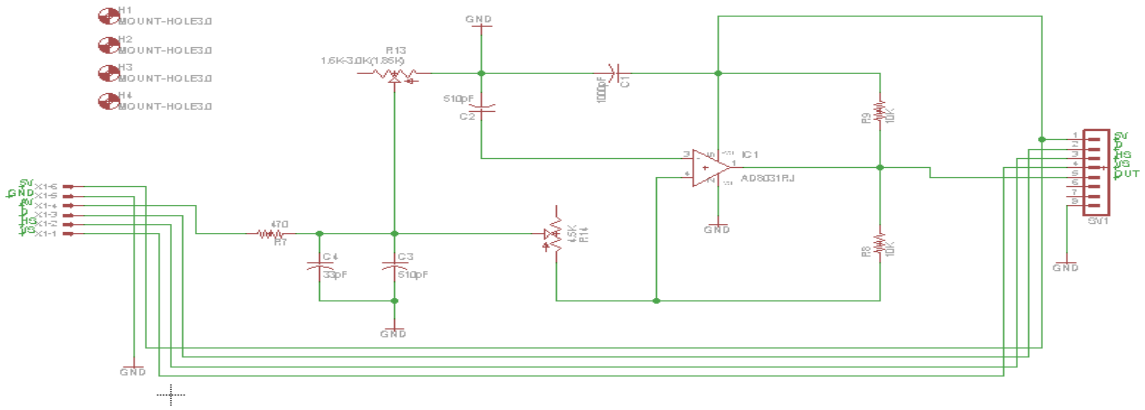


Figure 5

The figure 5 is the 2D camera schematic. In the spring quarter, we will try to use 2D camera instead of the linear scan camera, but there are a different principle between the 2D camera and the linear camera. Therefore, making a 2D camera PCB is very

important to us. Firstly, the 2D camera can load a 240 x 320 pixels. We need to use a 5-lead small outline transistor to separate the HS, VS and output signal. Even this is a simple circuit, we also need to test the value of resistors and capacitor. Figure 6 is the 2D camera PCB layout.

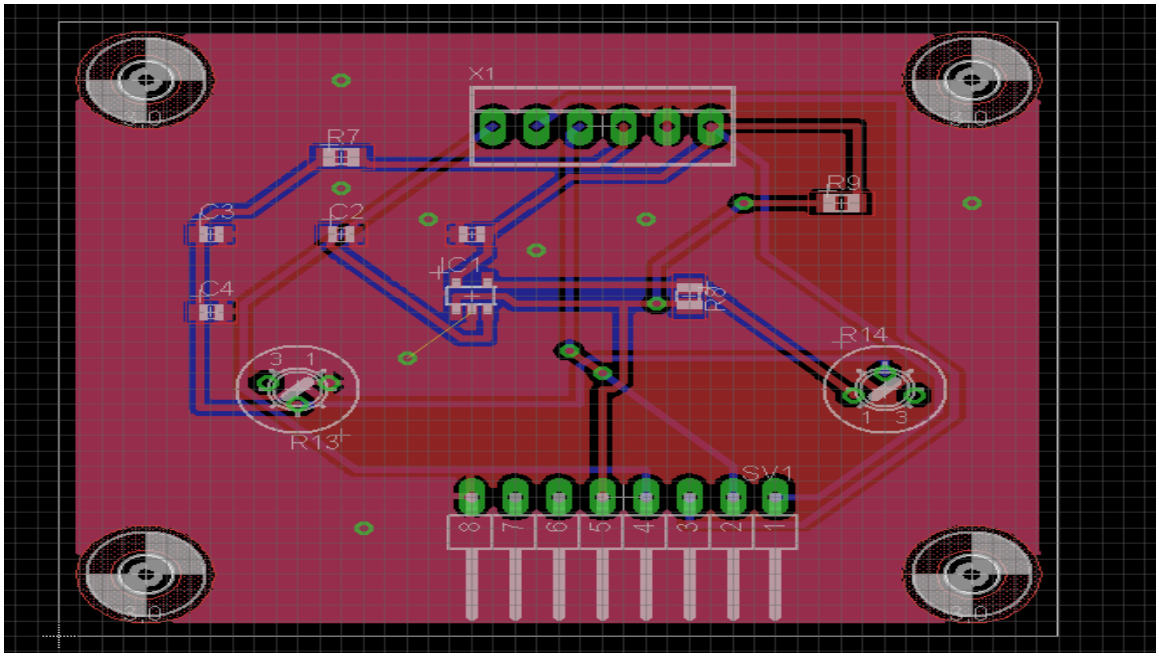


Figure 6

Design and Performance Summary

Even though out group did not pass the first competition, we finished the second competition in 34 seconds. In this project, we was accomplished to analyze the data from linear camera and make the car run through scan the two tracks. If you had it to do over again, we will try to learn the PID control system clearly and set an accelerating installation to make our car faster.

Safety Summary

There are a lot of ways to ensure that our car would never be a hazard. For example: we would set a bumper at front of car to reduce the impulse force when the car loses the track; we would set a normal speed so that It is not acceptable for it to take off across the room at full speed! Here is a list for the car safety summary.

1. Set the width of the wire in the motor PCB as width as possible to protect the PCB from burning.
2. Using two potentiometers controls the speed of motors.
3. Never making two pins short because short circuit can burn the device.
4. Care about the power supply, never making the voltage higher than 7.2V.
5. Do not making the car too low, the bottom of the car will touch the floor as the car go through a hill.

Appendix A

Detection Functions

```
int Slope( int camNum, unsigned int input[]){  // Convert ping-pong buffer in to 1's and
0's
        unsigned int tep;
        int begin = 0;
        int end=0;
        int btw = 0;  //blk to white
        int wtb = 0;  //white to blk
        int slope_sign = 0;  //2 negative, 1 positive, 0 nothing
          for (i=0;i<128;i++)
                        track[i] = 1;
        for(i=3;i<123;i++){
                if(input[i+1] > input[i-1]) {
                        tep = (input[i+1] - input[i-1])/2 ;
                        btw = 1;
                        wtb = 0;
                }
                else if (input[i+1] < input[i-1]) {
                        tep= (input[i-1] - input[i+1])/2;
                        btw = 0;
                        wtb = 1;
                }
```

```
                else
                        tep=1;
// white to black
// negative slope
        if((tep > hexdiff) && !begin && wtb == 1) {
                            begin = 1;
                         slope_sign = 2;
                        }
                //black to white
                //positive slope
                else if ((tep > hexdiff) && !begin && btw == 1){
                         begin = 1;
                         slope_sign = 1;
                }
                if(begin) { //marking position of track
                                track[i] = 0;
                                track[i+1] = 0;
                                track[i-1] = 0;
                        }
        if((tep > hexdiff) & begin) {
                                begin = 0;
                                    track[i] = 0;
                                track[i+1] = 0;
```

```
                    track[i-1] = 0;
                }
            return slope_sign;
}
```

Appendix B

Steering Function

```
void steering (int w1, int w2, int avg1, int avg2)
//  (left track pos, right track pos, l_safe_zoom, l_end_zoom, r safe zoom, r end zoom
//  safe_zoom: where car runs in a straight line

{
  int w_diff=0;
        int temp =0;
  int w =0;
        int w1_diff= 0;
  int w2_diff = 0;
        int L_missing = 0;
        int R_missing = 0;
        //pGain = 9, dGain = 20
  //pGain, dGain are defined in main.h
           //5,120,8,115
        //low to high = blk to white slope is neg
```

```
          //high to low = white to blk  slope is pos
          //CAM0_slope_sign 0 -> sees nothing, 2 is negative , 1 is postive
  //CAM1_slope_sign
// CAM0 positive slope
if(CAM0_slope_sign == 1) {
// CAM1 positive slope, CAM0 still sees data

          if(CAM1_slope_sign == 1) {
//CAM0 sees left, CAM1 detects left too;

          if (w1 >110 )
             servo_pwm = 5600;

          else

          servo_pwm = 4500 + pGain * w1 * 1.1 + avg1 * dGain;

                                                                                    }
        // CAM1 reads no data <--> CAM0 takes the control
          else if (CAM1_slope_sign == 0){
         if (w1<30)
         servo_pwm = 4500 + pGain * w1 * 1 + avg1 * dGain;
         else if (w1<50)
```

```
        servo_pwm = 4500 + pGain * w1 * 1.2 + avg1 * dGain ;
        else if (w1<80)
        servo_pwm = 4500 + pGain * w1 * 1.4  + avg1 * dGain ;
        else
        servo_pwm = 4500 + pGain * w1 * 1.4 + avg1 * dGain ;
         }
        //CMA1 negative slope
    else if (CAM1_slope_sign == 2) {
    if(w2 < 35 && w1 < 35)
    servo_pwm = 4500;
    else
    servo_pwm = 4500 + (w1 - w2)*pGain*0.4 - 8 + (avg1 - avg2)*dGain;
        }
 } //End CAM1_SLOPE_SIGN = 1
//CAM0 negative slop            left cam sees right track
else if (CAM0_slope_sign == 2) {
   //case1 CAM1 still collects data
if(CAM1_slope_sign == 2) {
    if(w2>100) {
         servo_pwm = 3600;
}
else
 servo_pwm = 4500 - w2 * pGain  - avg2 * dGain;
```

```
 }
//if CAM1 can't see data, indicates that car is very close to right track
else if (CAM1_slope_sign == 0){
servo_pwm = 3400;
}
 }
    else if (CAM0_slope_sign == 0) {
        if(CAM1_slope_sign == 1) {
                        servo_pwm = 5400;
                                }
                        else if (CAM1_slope_sign == 2) {
                                if(w2<30)

                        servo_pwm = 4500 - w2 * pGain  - avg2 * dGain;
                                else if (w2 <50){

           servo_pwm = 4500 - w2 * pGain*1.2  - avg2 * dGain;
                        }

        else if (w2 <80){
                servo_pwm = 4500 - w2 * pGain * 1.4 - avg2 * dGain ;
        }
        else{
```

```
        servo_pwm = 4500 - w2 * pGain * 1.4 - avg2 * dGain ;
                }
        else if (CAM1_slope_sign == 0) {

        if(ALL_white1 && ALL_white2) {

                servo_pwm = 4500;

        }
                }
                        }
        NVIC_EnableIRQ(TPM1_IRQn);

}
```