EEC 195 Autonomous Vehicle Design Final Report

Christopher Chan
Jasper Siu
Elizabeth Ear
Jason Woo

I. Overview (Executive Summary)

For our autonomous vehicle, our car was designed so that it operated simply yet efficiently. We had to design, build and test speed control circuits, track sensing circuits, and a steering control with the implementation of edge detection cameras. When programming concepts, such as Proportional and Derivative control, we worked on making performance variables easily tunable when testing our car on the track. We chose a simpler design because we wanted to make modifications and debugging easier.

These were our primary emphases, but not restricted nor limited to these contributions. We all contributed in hardware, testing, troubleshooting, motor control design, report write-ups, designing the car, and the first quarter labs.

Breakdown of Contribution Percentage:

Team Member	Percentage	Contributions
Chris Chan	33%	Programming
Jasper Siu	25 %	Programming
Elizabeth Ear	21%	Camera Design
Jason Woo	21 %	PCB

Camera - Edge Detection (Elizabeth Ear)

For edge detection we interfaced to two linescan cameras using PIT and ADC interrupts. Because the processor only allows for 1 ADC conversion at a time, we stored data in two sets of ping-pong buffers and investigated track detection for an Edge Lane track. To fill the buffers with data from camera 1 and camera 2, we alternate and fill both up at the same time. So, we collect a value from camera 1, and put it in ping 1, and then we collect another value, and place it in ping 2, and so on. The SI is set high and then the ADC conversion CLK runs for 256 cycles, so that it can fill both ping 1 and ping 2. After 256, SI is set to high once again.

As for the hardware, we used the J3 header on the TFC shield board to interface a second linescan camera to the Freedom microcontroller board. From the TFC shield schematics, the KL25Z signals connected to the J3 header are as follows:

AO:	PTD6 (ADC0_SE7b)
SI	PTD7
CLK	PTE1

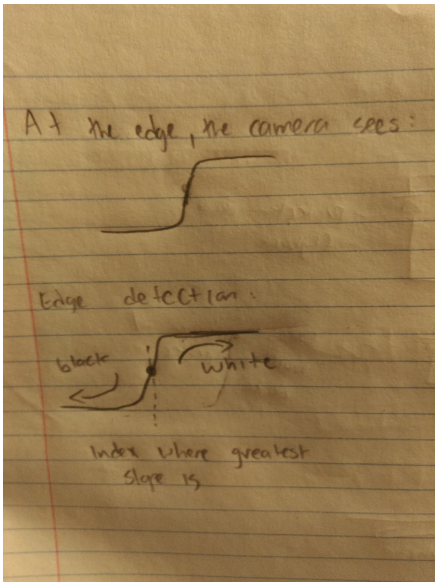
The same processor signals (PTD7 and PTE1) were used for the SI and CLK signals for both of the linescan cameras. Since both cameras have identical SI and CLK signals, the cameras will always have the same integration time. We used the KL25Z's on-chip ADC to do the A/D conversions for the two linescan cameras. The advantage of using two A/D converters on our microcontroller is that it processes faster and will most likely make our car more accurate and faster. This improves the interface of the two linescan cameras because we decided to use two

ping pong buffers. Since we utilized two ping-pong buffers (one for each camera) we needed a way to fill up the buffers just as fast or even faster. In order to do so, we use two A/D converters so that the buffers are able to convert and fill up simultaneously.

All of the A/D conversions for the camera pixel data were done using interrupts. We obtained 128-byte line of pixel data from each linescan camera for each PIT interrupt. Therefore, there was 256 ADC interrupts per PIT interrupt. The PIT interrupt handler started the first A/D conversion and then we used the A/D interrupt handler to start each successive A/D conversion until all 256 A/D conversions have been done. We used ping-pong buffering for both cameras, where each camera has its own set of ping-pong buffers in program memory (Ping, Ping_2, Pong and Pong_2).

After we used a GPIO output to measure the time to clock out the 128-bytes of pixel data from both cameras, we set the GPIO signal high in the PIT interrupt handler and set the signal low when the final A/D value is stored in a global variable. Our A/D conversion was about 1.05ms which made sense with the pulse width that we obtained.

The values we obtained in the ping pong buffers were used for edge detection of the track. We calculated all the slopes from all the buffer values using slope = (buffer[i-1] - buffer[i+1]) / 2 . Then, we looked for the index containing the greatest slope because we knew there would be a slope in between the white portion and the dark portion when the camera sees the edge of the track. The slope index would help us distinguish what portion of the track was white or black from our cameras. Any values past the index of where the greatest slope was considered white and any values behind the slope value were considered black.



Caption: Slope Threshold Method Visual

We tried three different camera heights of one inch, two inches, and three inches away from the surface at different angles (30, 45, 60 degrees). The configuration that we thought worked best was between three and four inches at an angle of 60 degrees to account for shadows. The cameras are highly sensitive to shadow. If the camera detects even the slightest of dark area, it will analyze it incorrectly and veer off the track.

When we started testing the car for the checkoffs and competition, we ran into a lot of camera problems. The first thing we did was mount the cameras on a plexiglass attached to a couple metal poles. We wanted the cameras to be adjustable in all directions so we screwed the cameras on hinges that would allow for the camera to look up and down. This was only sufficient for the first check off. After a while, the glass was not as sturdy as we had hoped. So, we decided to get rid of the glass and just stick with the 2 metal poles. The poles gave more leeway for the cameras to face outward and detect the edges better.

As you can see from the pictures in the Appendix, the cameras were able to move freely up and down and side to side. This method was sufficient for checkoff two, however, the hinges were too flimsy that the cameras would move or fall down every time they crashed.

So, we went back to ace hardware and bought a wooden dowel to widen the camera angles. Having the wooden stick allowed us to spread the cameras more wide and grab better data from the edges of the track. This seemed to work the best for us and gave us camera readings that were much more clear and precise. See appendix for pictures of various designs.

Servo Control (Jasper Siu)

Our car used a closed loop feedback system to control the car's servo. The values that we read through the camera are used to feedback information to the plant, in this case, our servo. The main value that controls our servo is the Pulse-Width (PW1) value. To control the servo, we first implemented a proportional controller.

After converting the camera readings into an array of 128 values classified as lights and darks, we compared the quantity between the two cameras. With calibrated camera positions, the amount of lights and darks at the center are equal on both cameras. As the car moves to the left, the left camera will see an increased number of dark values and the right will see the center of the track, hence more light values. Our algorithm for our servo control utilized the difference between the number of lights and darks from each camera. The difference was the error value should be around zero to straighten and center our car, otherwise our car would turn depending if the error was positive or negative. For the proportional controller, we took the error and multiplied it by Kp. The Kp value was set to a value that was high enough that enabled the car to reach hard lefts and rights 3000 and 6000 respectively. The servo is configured so that 4500 sets

the wheels straight. The error value also works in place so that if there is a discrepancy of whites that the two cameras see, the error value will be negative or positive. This works so that when the error multiplied by the Kp is added to the initial servo value of 4500 it will turn the servo left or right. This Kp value helped overall amplify the error which helped turn the car enough and even reach the max servo values if needed. The Kp value needed to be tuned according motor speeds, but for our competition 2 we set it to 12, which we derived through track testing and trial and error.

Next, we implemented a derivative controller. The derivative controller is used to pre-empt what will happen next and help anticipate hard turns. To use derivative control, we looked at the previous error values and calculated the slope. During hard turns, there would be an increasing error due to the change in light and dark values and thus, the slope of the errors would change as well. Our algorithm would store the previous error values in a moving array called "ErrArr" and we use the error values to calculate a slope. This slope value would then be multiplied by a Kd value. The slope equation we used is,

$$D(n) = k_d[E(n) + 3E(n-1) - 3E(n-2) - E(n-3))/6T]$$

, where our sample speed (T) was .04 seconds. Our k_d value needed to be adjusted according to different servo speeds but the number it is set to is 1, through knowing our T value as well as from trial and error on the track. The entire derivative calculation which we named DerivPortion was then also added to the neutral servo position. Our final equation was

$$PW1 = ((lightcount1 - lightcount2) * 12) + 4500 + DerivPortion.$$

The derivative portion helped with sharp turns and would help the car overturn slightly which helped the car steer back to the middle of the track when coming too fast into a turn and position

the cameras to read the next future values. In addition to the sharp turns it helped on the hills. Our car would be coming out very fast off of the hills, but the derivative allowed the car to still make the turn at high speeds..

To prevent the car from being too sensitive to the slightest of turns, we also implemented a dead-band region. In the dead-band region, which the physical region was a few centimeters to the left and right of the car when placed dead center, we set the servo to hold at a constant value. We found the dead-band region mathematically by saying any difference of our light and dark values were very small, (in our case 10), then it would leave the car straight.

We had considered implementing integral controller. However, integral if not tuned precisely could lead to oscillations, which was already a problem we had not completely eliminated from our existing car, and we did not want to provoke it further.

Custom PCB and Competition II Setup (Jason Woo)

For our custom PCB, we first started by watching multiple online tutorials to get acquainted with how to use Eagle Cadsoft. Then we started by looking at the FRDM-TFC Schematic that was provided. From there we started to use Eagle to put the different parts on a board. The main components included were the line scan camera interfaces, potentiometers, switches, battery, and the MAX620 with the H-bridge. The schematic closely follows the schematic that was provided in Smartsite. We separated each part and labeled individual headers and pins so that we could see them in layout mode. It was important to use components in the schematic that would give us what we wanted in layout form. For instance many of the switches, potentiometers, and passive components (capacitor, resistors) had many different options. We had to choose the one that would give us the best fit in the layout mode.

In layout mode, we tried to keep the pin-outs (TFC Shield) that were going to be used as far as possible from the motor control portion. In our schematic we put these pin-outs at the "top" of the board while the IC/diodes were at the "bottom" of the board. We placed the passive components on the "sides" of the board and tried to configure the board kind of like our original TFC shield with the potentiometers and switches in the middle of the board. The most useful function was the "ratnest" function and autorouter. Since we were using a free student version the autorouter wasn't as nice as the other versions. However, it still set traces nicely and the ratnest function was very helpful. We set the grounds to a common ground in layout. Once connected, the V-Battery would also be the common ground for our whole car setup.

The most challenging aspect of using Eagle was getting use to the software initially. For most of us, it was the first time using a PCB design software. Originally, we tried twice to draw the headers using the regular draw rectangle function that was under the draw tab. It wasn't until we re-watched tutorials and asked the professor that we realized you had to use the libraries and look up individually the passive and active components. The hardest components to find were some of the active ones such as the MAX620 IC as well as some of the connectors. You had to type out the whole component name that was part of the schematic or else it wouldn't show up in the libraries provided by Eagle. We tried to find some of the parts online in forums that had perhaps made the components that we were looking for, such as the battery header or the IC we were looking for, but those were not very helpful. We eventually just resorted to looking back into the libraries provided and found all the parts we needed.

For competition 2, we were not able to get our design to the PCB manufacturers in time. Therefore, we decided to implement our design using a small breadboard, the lab 8 motor control

board, and the freedom board. We looked up the same schematic (TFC shield schematic) to wire the breadboard as an interface for switches and potentiometers. Originally we used our old lab 8 motor control board but when we tested it the first time we fried a trace line on both the TFC shield and motor control board. That's when we realized it was because we were not using a voltage regulator that was needed to supply 5V to both devices correctly. We originally thought we could just use the 5V that was supplied through one of the pins on the freedom board. After that we decided that instead of using a TFC shield we would use the breadboard so that we could also implement a 5 V regulator. Refer to the appendix for the PCB schematic and layouts.

The hardest part was the actual wiring of these three boards together. There were many complications and it took three times to finally wire it correctly. Originally, we used old wires from old classes but found out these connections were terrible. The car would sometimes respond well but it was very inconsistent. We switched from these wires to the wires that were in the 195 lab room (thick, insulated wires) and this change made the car work exactly as it did in competition 1. We also found that our cameras were switched and the motor made the wheels run backwards but were prepared for that difficulty and knew that you just had to switch the leads. Testing the car was more difficult with this setup compared to competition 1 because the track was more challenging. The new setup took up a lot more space and had many wires that could jostle when loose and thus took a longer time to test for a combination that ran through the whole track.

Speed Control (Christopher Chan)

Speed Control is very important because it will vary the car's speed at different portions of the track to minimize the overall time it takes for the car to complete race. We used potentiometers that will set a pulse width to drive the DC motors. This determined the speed of the car. During the first competition, we left our car at a constant speed. However, we realized we needed to make our car faster without it steering off the track during a turn or after a hill. As a result, we implemented speed control and we did so by increasing or decreasing the pulse width driving the DC motors. Overall, our design uses the braking of one wheel and the increase of the pulse width of the other DC motor to effectively make sharp turns while maintaining a high speed during straight paths.

We controlled our car using a unipolar PWM, driving one side of the H-Bridge with a PWM signal and the other with a GPIO output set at logic high or low. Although we knew we could have controlled our DC motors with bipolar PWM, we decided not to ensure that our car would not go reverse during any part of the race. The tracks had many sharp turns and one difficulty we ran into was going into a ninety degree turn after coming down from a hill, especially at higher speeds. In order to solve this issue, we implemented torque vectoring. When the cameras detected a sharp turn, we turned off one of our DC motors, thus braking a wheel and reducing the current going through one of the DC motors. We made it so that only the outermost wheel would still be moving and the car would pivot on the stopped wheel, allowing our car to make a sharper turn. At even higher speeds, we would increase the speed of the DC motor that is still on to make even sharper turns. The speed of both the wheels would return after the cameras detected a straight portion of the track and the servo straightened out.

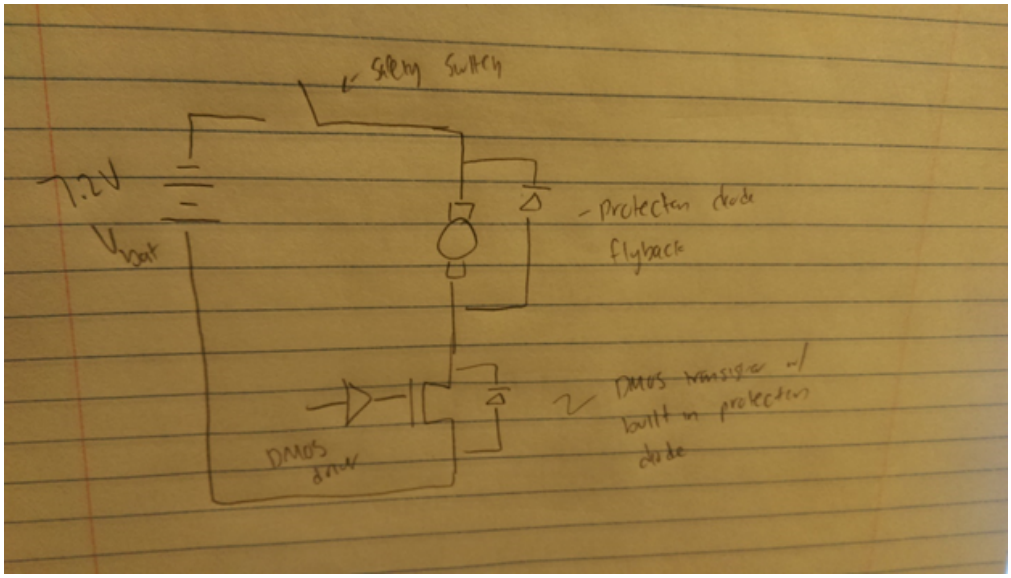
```
if(PW1 > 4700){
  turnspeed = ReadValue2 + 86;
  ReadValue2 = turnspeed;
  ReadValue = 0;

}
else if(PW1 < 4300){
  turnspeed = ReadValue + 86;
  ReadValue = turnspeed;
  ReadValue2 = 0;

 if((lightcount1 - lightcount2) <= 45 && (lightcount1 - lightcount2) >= -45){
   ReadValue = holda;
   ReadValue2 = holda;
 }

void TPM0_IRQHandler(void){ // dc motor
//clear pending IRQ
   NVIC_ClearPendingIRQ(TPM0_IRQn);
  if(TPM0->CONTROLS[0].CnSC & TPM_CnSC_CHF_MASK)
    TPM0->CONTROLS[0].CnSC |= TPM_CnSC_CHF_MASK;
  if(TPM0->CONTROLS[2].CnSC & TPM_CnSC_CHF_MASK)
    TPM0->CONTROLS[2].CnSC |= TPM_CnSC_CHF_MASK;

    TPM0->CONTROLS[0].CnV = ReadValue;
    TPM0->CONTROLS[2].CnV = ReadValue2;

    if(darkcountflag == 1){  // turn off motors when see all black
      TPM0->CONTROLS[0].CnV = 0;
      TPM0->CONTROLS[2].CnV = 0;
    }
}
```

Caption: This code is an example of us increasing the speed of one DC motor while shutting off the other DC motor (Torque Vectoring). As the car straightens back out after a turn by analyzing our error, we return the speed originally set by our potentiometer. These values are updated in the TPM0_IRQHandler. We chose to add the number 86 to our Pulse Width value through trial and error as it seemed to work best after the hill.

However, that came with a trade off because occasionally our servo would oversteer, causing us to occasionally go into the next part of the track crooked. Sometimes it would still

continue in the correct portion of the track while other times it would steering off to another portion of the track. But this wasn't as big of an issue for us as it happened infrequently. Also, this made our turning mechanism a bit choppier and going into a straight path our car would oscillate for a meters until eventually the PD controller was able to mostly straighten out the position of our car with a slight oscillation. However, these were disadvantages were eventually improved through implementing a better derivative control. It helped us come into and out of turns better which worked well in accordance to our speed control.

This torque vectoring technique worked very well with the TFC shield, but we were not able to perfect it in time for competition one. However, for the Freescale Cup we will be sure to implement this technique in our car. When installed the custom motor PCB on our wheel did not brake as quickly as we intended it to do so. A speculation we had for this issue was because the custom motor PCB consisted of a flyback diode in parallel with the DC motor. The flyback diode allowed the current to dissipate but since it was in parallel with the DC motor, it would take time for some of the current to dissipate due to the motor inductance. This caused the motor to gradually lose speed for a second or two before fully coming to a stop. The TFC shield had a transistor in parallel with the DC motor which current flowed through. Having this allowed the DC motor to brake faster on the TFC shield. For now, this is our own speculation. For the future competitions, we will try to prove our hypothesis and use our custom PCB that we designed. With it, hopefully we can get our DC motors to brake faster.



Caption: Single DMOS DC-Motor Control Circuit. DC motor with flyback diode.

III. Design and Performance Summary

For the first competition, we were only allowed three tries to get around the track successfully. The first complete time was then recorded. Our main objective was to consistently design the car to run around the course completely. The weekend prior to competition, we had our car running very smoothly but had no implementation of any other controller, besides proportional. Since it was running consistently, we decided to implement a derivative controller and torque vectoring. When we tried implementing these new concepts, our car became decreasingly reliable. It would consistently veer off the track or while crossing an intersection turn unexpectedly. So, after countless hours of testing, we decided to revert back to our original code that lacked the derivative and torque vectoring controller just so we could .

After the first competition, we once again began to implement the derivative in our controller. As we worked on it, we saw our car improve vastly. We began to implement torque vectoring into our car as well. Things went smoothly until we removed the TFC Shield and installed our motor PCB. Response times were slower and torque vectoring did not work as well as we hoped. The week before the competition, we tried to work with our adjusted circumstances. Unfortunately, come Competition 2, we were not able to tune our car to a fast speed. There were concepts that we wanted to implement but the probability of failure was too high. During competition day we had to revert to a "safer" code and turn our motor speed down. In the end, we did not perform as well as we hoped during the competition, but we have acquired knowledge that we can use in future Freescale competitions.

After our two quarters of design and testing, we found some things that we could have changed. We realized our torque vectoring technique was not effective as we hoped it to be during competition two. This is because we used the custom motor PCB instead of the TFC shield and upon testing, we realized that the DC motor controlling our wheel did not shut off as fast as we expected it to shut off. Therefore, this significantly slowed down the speed at which our car could go because torque vectoring helped us make sharper turns at faster speeds. Next time, we would make our own motor control PCB similar to the TFC shield uses so our motors could shut off faster and so we would not have to compromise any speed for the Natcar competition. We hope the PCB we made will do work as planned in future competitions. Also, we would use the NTSC cameras because the imaging for the NTSC is much more clear than the line scan camera. This would help us get more distinct value readings, making our edge detection

method more accurate. This would also clean up all the wiring from our car since we used the custom motor control from Lab 8.

IV. Safety

During the construction of our car, we tried to follow safe lab practices. When we were operating the drills and soldering iron we made sure to use goggles and to turn off the iron when we were done using it. Other things we did to practice safety is to cleanly strip our wires so that not a lot of live wire showed for the second competition when we had to use a small breadboard for our switches, potentiometers, and the voltage regulator. Originally we had used thin wires from a past lab but decided to switch these for the thick red insulated wires that were provided in lab for noise control. We connected all the ground pins to the common ground of the battery. The last thing we did was to ask for help when we needed materials that we didn't know how to handle. For instance, when we needed plexiglass we made trips to ACE Hardware to ask them to cut pieces that we needed. Our car design changed as the quarter went on, as we found better camera placements and sturdier frames, so we made extra trips to ACE to ask them to cut us material.

On the car itself, we made sure the add a bumper that was a thick sponge so that when we hit things the car's camera placements and frame wouldn't be too damaged. The sponge was thick enough to absorb the collision. When we were testing the car, we tried to follow safe practices by making sure that we didn't run the car in the opposite direction as the other people that were testing their cars and only tried another direction when the track was free. We also made sure to follow the car in case another car was going in the opposite direction so that we

could catch ours and avoid unnecessary collisions. The last thing we did was to completely stop our car after it sees a prolong duration of black so that the car wouldn't run off the track incessantly. If both cameras saw all black we turned off the DC motors.

V. Appendixes

NATCAR.c*

```
int w1, w2;

unsigned int turnspeed, turnspeed2;

//volatile unsigned short PW1 = 0;        //change to 4500 to have at 1.5ms
volatile unsigned int PW1;
volatile unsigned int PW = 0;
//volatile unsigned short PW;
int uart0_clk_khz; //uart clock
volatile unsigned int ReadValue, ReadValue2, holda, holdb;
int SW2;
int Switch;
volatile unsigned int ReadValue3, ReadValue4;

void put(char *ptr_str)
{
  while(*ptr_str)
    uart0_putchar(*ptr_str++);
}

void LED_Initialize(void) {

  SIM->SCGC5    |= (SIM_SCGC5_PORTB_MASK
                  | SIM_SCGC5_PORTD_MASK);           /* Enable Clock to Port B & D */
  PORTB->PCR[18] = (1UL <<  8);                     /* Pin PTB18 is GPIO */
  PORTB->PCR[19] = (1UL <<  8);                     /* Pin PTB19 is GPIO */
  PORTD->PCR[1]  = (1UL <<  8);                     /* Pin PTD1  is GPIO */
  PORTD->PCR[7]  = (1UL <<  8); //pin ptd7 is GPIO
  PORTE->PCR[1]  = (1UL <<  8);

  PORTC ->PCR[4] = (1UL << 8); // turns transistors on for H-Bridge
  PORTC ->PCR[2] = (1UL << 8);

  FPTC->PDOR = (1UL << 4 | 1UL << 2);
  FPTC->PDDR |= (1UL << 4 | 1UL << 2);
  FPTC->PCOR = (1UL << 4);
  FPTC->PCOR = (1UL << 2);


  FPTB->PDOR = (led_mask[0] | led_mask[1] );          /* switch Red/Green LED off  */
  FPTB->PDDR |= (led_mask[0] | led_mask[1] );          /* enable PTB18/19 as Output */
  FPTD->PDDR |= (outport[0]); //enable PTD7 as an output
  FPTE->PDOR &= ~2;
  FPTE->PDDR |= 1UL << 1;
```

NATCAR.c*

```
135   FPTD->PDOR = led_mask[2];            /* switch Blue LED off  */
136   FPTD->PDDR |= led_mask[2];           /* enable PTD1 as Output */
137  }
138
139
140 void Init_ADC(void) {
141
142   init_ADC0();      // initialize and calibrate ADC0
143   ADC0->CFG1 = (ADLPC_NORMAL | ADIV_1 | ADLSMP_LONG | MODE_8 | ADICLK_BUS_2); // 8 bit, Bus clock/2 = 12 MHz
144   //ADC0->CFG2 |= ADC_CFG2_MUXSEL_MASK;
145   ADC0->SC2 = 0;    // ADTRG=0 (software trigger mode)
146  }
147
148
149 unsigned int Read_ADC (void) {
150   volatile unsigned int res=0;
151
152   while (!(ADC0->SC1[0] & ADC_SC1_COCO_MASK)) { ; }   // wait for conversion to complete (polling)
153
154   res = ADC0->R[0];       // read result register
155   return res;
156  }
157
158
159 void ADC0_IRQHandler() {
160   ADCCOUNT++;
161   if (ADCCOUNT%2==0){
162     FPTE->PSOR = 1UL << 1;              // setting CLK PTE1 high
163     CLKcounter++;
164   }
165   NVIC_ClearPendingIRQ(ADC0_IRQn); //clear IRQ for ADC
166
167   res = ADC0->R[0];       // read result register (read A/D value??
168
169
170   // Ping Pong Buffer
171   if (pp==0){                          //pp alternates between ping and pong
172     if (ADCCOUNT==256) {
173       //ADCCOUNT=0;
174        pingflag = 5;
175       pp = 1; //done FLAG
176       p = 0;
177       q = 0;
178      printflag1 = 1; // when ping1 and ping2 are full
```

NATCAR.c*

```
181     else if(pingflag==0){       //pingflag alternates between ping and ping2
182       printflag1 = 0;
183       ping[p] = res; // stores all 128 bytes in a circular buffer
184       p++;
185     }
186     else if(pingflag==1){
187       ping2[q]=res;
188       q++;
189     }
190
191     if(pingflag==0)
192       pingflag=1;
193     else
194       pingflag=0;
195     }
196
197   else if (pp==1){
198     if (ADCCOUNT==256) {
199       //ADCCOUNT=0;
200       pongflag = 5;
201       pp = 0; //done FLAG
202       p = 0;
203       q = 0;
204       printflag2 = 1;
205     }
206
207     else if(pongflag==0){
208       printflag2 = 0;
209       pong[p]= res; // stores all 128 bytes in a circular buffer
210       p++;
211     }
212     else if(pongflag==1){
213       pong2[q] = res;
214       q++;
215     }
216
217     if(pongflag==0)
218       pongflag=1;
219     else
220       pongflag=0;                     //ping/pongflag, pp, printflag
221   }
222
223
```

NATCAR.c*

```
224   if (CLKcounter < 129){
225     if(ADCCOUNT%2==0){
226       ADC0 -> CFG2 |= ADC_CFG2_MUXSEL_MASK;                       // select b channel
227       ADC0->SC1[0] = AIEN_ON | DIFF_SINGLE |ADC_SC1_ADCH(6);      // start conversion (software trigger) last one might be 9
228     }
229     else{
230       ADC0 -> CFG2 |= ADC_CFG2_MUXSEL_MASK;                       // select b channel
231       ADC0->SC1[0] = AIEN_ON | DIFF_SINGLE |ADC_SC1_ADCH(7);      // start conversion (software trigger) last one might be 9
232     }
233   }
234   else{
235     FPTB->PCOR = 1UL << 0; //deassert GPIO signal for measuring conversion time of the entire line
236     if (ADCCOUNT == 256){
237       ADC0 -> CFG2 &= ~ADC_CFG2_MUXSEL_MASK;                        // select a channel
238       ADC0->SC1[0] = AIEN_ON | DIFF_SINGLE |ADC_SC1_ADCH(7);      // start conversion (software trigger) last one might be 9
239     }
240   }
241
242     if (ADCCOUNT == 257){
243       ReadValue3 = res;       // read result register                          // Feedback A
244       ADC0 -> CFG2 &= ~ADC_CFG2_MUXSEL_MASK;                      // select a channel
245       ADC0->SC1[0] = AIEN_ON | DIFF_SINGLE |ADC_SC1_ADCH(3);      // start conversion (software trigger) last one might be 9
246     }
247     if (ADCCOUNT == 258){
248       ReadValue4 = res;       // read result register                          // Feedback B
249       pingflag = 0;
250       pongflag = 0;
251     }
252
253   FPTE->PCOR = 1UL << 1;      //deassert CLK
254 }
255
256 void TPM1_IRQHandler(void){
257 //clear pending IRQ
258   NVIC_ClearPendingIRQ(TPM1_IRQn);
259   TPM1->CONTROLS[0].CnSC |= (1UL << 7);
260
261   TPM1->CONTROLS[0].CnV = PW1;      //Hold Servo Constant
262 }
263
264 void TPM0_IRQHandler(void){ // dc motor
265 //clear pending IRQ
266    NVIC_ClearPendingIRQ(TPM0_IRQn);
267   if(TPM0->CONTROLS[0].CnSC & TPM_CnSC_CHF_MASK)
```

NATCAR.c*

```
267   if(TPM0->CONTROLS[0].CnSC & TPM_CnSC_CHF_MASK)
268     TPM0->CONTROLS[0].CnSC |= TPM_CnSC_CHF_MASK;
269   if(TPM0->CONTROLS[2].CnSC & TPM_CnSC_CHF_MASK)
270     TPM0->CONTROLS[2].CnSC |= TPM_CnSC_CHF_MASK;
271
272     TPM0->CONTROLS[0].CnV = ReadValue;
273     TPM0->CONTROLS[2].CnV = ReadValue2;
274
275     if(darkcountflag == 1){  // turn off motors when see all black
276       TPM0->CONTROLS[0].CnV = 0;
277       TPM0->CONTROLS[2].CnV = 0;
278     }
279 }
280
281 void Init_PWM(void) {
282 // Set up the clock source for MCGPLLCLK/2.
283 // See p. 124 and 195-196 of the KL25 Sub-Family Reference Manual, Rev. 3, Sept 2012
284 // TPM clock will be 48.0 MHz if CLOCK_SETUP is 1 in system_MKL25Z4.c.
285
286   SIM-> SOPT2 |= (SIM_SOPT2_TPMSRC(1) | SIM_SOPT2_PLLFLLSEL_MASK);
287
288 // See p. 207 of the KL25 Sub-Family Reference Manual, Rev. 3, Sept 2012
289
290   SIM->SCGC6 |= SIM_SCGC6_TPM0_MASK | SIM_SCGC6_TPM1_MASK;; // Turn on clock to TPM0
291
292 // See p. 163 and p. 183-184 of the KL25 Sub-Family Reference Manual, Rev. 3, Sept 2012
293
294 //  PORTB->PCR[1] = PORT_PCR_MUX(3); // Configure PTB1 as TPM1_CH1
295 //  FPTB->PDDR |= 1UL << 1;
296
297 // Configuring PTB0 as a GPIO and Ou
298
299   PORTB->PCR[0] = PORT_PCR_MUX(3); // Configure PTB0 as TPM1_CH0
300   //Added:
301
302 // Set channel TPM1_CH1 to edge-aligned, high-true PWM
303
304   TPM1->CONTROLS[0].CnSC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK| TPM_CnSC_CHIE_MASK ;   // Generate interrupts
305
306 //Clear Channel Flag
307   //TPM1->CONTROLS[0].CnSC |= (1UL << 7);
308
309 // Set period and pulse widths
310
```

NATCAR.c*

```
311    TPM1->MOD = 60000-1;     // Freq. = (48 MHz / 16) / 60000 = 50 Hz
312    TPM1->CONTROLS[0].CnV = PW;
313
314  // set TPM1 to up-counter, divide by 16 prescaler and clock mode
315
316    TPM1->SC = ( TPM_SC_CMOD(1) | TPM_SC_PS(4));
317    if (TPM1->CONTROLS[0].CnSC & TPM_CnSC_CHIE_MASK) TPM1->CONTROLS[0].CnSC |= TPM_CnSC_CHIE_MASK;
318
319  // See p. 163 and p. 183-184 of the KL25 Sub-Family Reference Manual, Rev. 3, Sept 2012
320
321  // Configuring PTB0 as a GPIO and Output
322
323    //PORTB->PCR[0] = PORT_PCR_MUX(3); // Configure PTB0 as TPM1_CH0  SERVO
324    PORTC->PCR[1] = PORT_PCR_MUX(4); // Configure PTC1 as TPM0_CH0
325    PORTC->PCR[3] = PORT_PCR_MUX(4); // Configure PTC1 as TPM0_CH2
326    //PORTC->PCR[2] = PORT_PCR_MUX(1);
327    //PORTC->PCR[4] = PORT_PCR_MUX(1);
328    // Set channel TPM1_CH1 to edge-aligned, high-true PWM
329
330    TPM0->CONTROLS[0].CnSC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK | 1UL<<6;   // Generate interrupts
331    TPM0->CONTROLS[2].CnSC = TPM_CnSC_MSB_MASK | TPM_CnSC_ELSB_MASK | 1UL<<6;   // Generate interrupts
332
333  //Clear Channel Flag
334    TPM0->CONTROLS[0].CnSC |= (1UL << 7);
335    TPM0->CONTROLS[0].CnSC |= (1UL << 7);
336
337  // Set period and pulse widths
338
339    TPM0->MOD = 2805;   // Freq. = (48 MHz / 16) / 60000 = 50 Hz   (48 MHz/ 16) / 3000 = 1069Hz
340    TPM0->CONTROLS[0].CnV = 0;   // i don't think we need this anymore?
341    TPM0->CONTROLS[2].CnV = 0;
342
343  // set TPM1 to up-counter, divide by 16 prescaler and clock mode
344
345  TPM0->SC = (TPM_SC_CMOD(1) | TPM_SC_PS(4));
346
347  // Enable Interrupts
348
349    NVIC_SetPriority(TPM1_IRQn, 128); // 0, 64, 128 or 192
350    NVIC_ClearPendingIRQ(TPM1_IRQn);
351    NVIC_EnableIRQ(TPM1_IRQn);
352
353    NVIC_SetPriority(TPM0_IRQn, 128); // 0, 64, 128 or 192
354    NVIC_ClearPendingIRQ(TPM0_IRQn);
```

NATCAR.c*

```
354    NVIC_ClearPendingIRQ(TPM0_IRQn);
355    NVIC_EnableIRQ(TPM0_IRQn);
356  }
357
358  void hexconvu(int input)
359  {
360    int MSB1, LSB1;
361    MSB1 = (input/16) & 0xF;
362    LSB1 = input & 0xF;
363    if(MSB1 > 9)
364      uart0_putchar(MSB1+55);
365    else
366      uart0_putchar(MSB1+48);
367    if(LSB1 > 9)
368      uart0_putchar(LSB1+55);
369    else
370      uart0_putchar(LSB1+48);
371  }
372
373  void hexconvs(int input)
374  {
375    int MSB1, LSB1;
376    MSB1 = (input/16) & 0xF;
377    LSB1 = input & 0xF;
378    if(MSB1 > 9)
379       uart0_putchar(MSB1+55);
380    else
381       uart0_putchar(MSB1+48);
382    if(LSB1 > 9)
383       uart0_putchar(LSB1+55);
384    else
385       uart0_putchar(LSB1+48);
386  }
387
388  void darklight(unsigned char input)
389  {
390    if (input==0)          //camera 2 ping 1 - working ish
391      uart0_putchar('0');
392    else
393      uart0_putchar('1');
394  }
395
```

NATCAR.c*

```
396  void analyzeServoFromLinescan(unsigned char buffer[128])
397  {
398    int j, slope;
399    int high = 0;
400    int low = 255;
401    int max;
402    int min;
403    //int PW;
404    for(j = 17; j <= 107; j++){
405      slope = (int)(buffer[j+1] - buffer[j-1])/2;
406      if(slope > high){
407        high = slope;
408        max = j;
409      }
410      if(slope<low){
411        low = slope;
412        min = j;
413      }
414    }
415    buffer = 0;
416    PW = (max+min)/2;
417  }
418
419  void slopecalc1 (unsigned char buffer1[128], unsigned char buffer2[128]){
420    int maxslope = 0;
421    int slope, index = 0, i;
422
423    for(t = 0; t<= 127; t++){
424      if(buffer1[t] < 127){
425        darkcount++;
426      }
427    }
428
429    if(darkcount > 120){
430      darkcountflag = 1;
431    }
432    darkcount = 0;
433
434    for(i = 1; i <= 126; i++){
435      slope = (int)(buffer1[i-1] - buffer1[i+1])/2;
436      if(slope > maxslope){
437        maxslope = slope;
438        index = i;
439      }
```

NATCAR.c*

```
443    for(i = 0; i <= 127; i++){
444      if(i < index){
445        buffer2[i] = 1;
446        uart0_putchar('1');
447      }
448      else{
449        buffer2[i] = 0;
450        uart0_putchar('0');
451      }
452    }
453  }
454
455  void slopecalc2 (unsigned char buffer1[128], unsigned char buffer2[128]){
456    int maxslope = 0;
457    int slope, index = 0, i;
458
459    for(t = 0; t<= 127; t++){
460      if(buffer1[t] < 127){
461        darkcount++;
462      }
463    }
464
465    if(darkcount > 120){
466      darkcountflag = 1;
467    }
468    darkcount = 0;
469
470    for(i = 1; i <= 126; i++){
471      slope = (int)(buffer1[i-1] - buffer1[i+1])/2;
472      if(slope > maxslope){
473        maxslope = slope;
474        index = i;
475      }
476    }
477
478    // right camera pong 1 & ping 1
479    for(i = 0; i <= 127; i++){
480      if(i < index){
481        buffer2[i] = 1;
482        uart0_putchar('1');
483      }
484      else{
485        buffer2[i] = 0;
486        uart0_putchar('0');
```

NATCAR.c*

```
/*----------------------------------------------------------------------------
  MAIN function
 *----------------------------------------------------------------------------*/
int main (void) {

  //initialize uart
  /* Enable the pins for the selected UART */
  /* Enable the UART_TXD function on PTA1 */

  SIM->SCGC5 |= (SIM_SCGC5_PORTA_MASK
                | SIM_SCGC5_PORTB_MASK
                | SIM_SCGC5_PORTC_MASK
                | SIM_SCGC5_PORTD_MASK
                | SIM_SCGC5_PORTE_MASK );
  SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK; // set PLLFLLSEL to select the PLL for this clock source
  SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1); // select the PLLFLLCLK as UART0 clock source

  SIM->SCGC5    |= (1UL <<  10) | (1UL <<  12);      /* Enable Clock to Port B & D */                    /* Pin PTD1  is GPIO */
  PORTB->PCR[1]  = (1UL <<  8);                      /* Pin PTB1  is GPIO */ /*For #5 we enabled PTB1 as a GPIO Signal and configured
  PORTE->PCR[21] = (1UL << 8);                       /* Pin PTE21 is GPIO Enabling H-bridge*/

  PORTC->PCR[13]= PORT_PCR_MUX(1);          //switch 1
  FPTC->PDDR&=~(1UL<<13);

  PORTC->PCR[17]= PORT_PCR_MUX(1);          //switch 2
  FPTC->PDDR&=~(1UL<<17);

  PORTA->PCR[1] = PORT_PCR_MUX(0x2);    // Enable the UART0_RX function on PTA1
  PORTA->PCR[2] = PORT_PCR_MUX(0x2);    // Enable the UART0_TX function on PTA2

  uart0_clk_khz = (48000000 / 1000); // UART0 clock frequency will equal half the PLL frequency
  uart0_init (uart0_clk_khz, TERMINAL_BAUD);
  //end initialize uart
  //unsigned char val;
  // PIT main start

    SIM->SCGC5    |= (SIM_SCGC5_PORTB_MASK
                  | SIM_SCGC5_PORTD_MASK);          // Enable Clock to Port B & D
  PORTB->PCR[0] = (1UL << 8);                       // Pin PTB0 is GPIO
  FPTB->PDOR |= 1;                                  // initialize PTB0
  FPTB->PDDR |= 1;                                  // configure PTB0 as output

  Init_PWM();
  LED_Initialize();
```

NATCAR.c*

```
  Init_ADC();
    /* Enable Interrupts */
  NVIC_SetPriority(ADC0_IRQn, 128); // 0, 64, 128 or 192
  NVIC_ClearPendingIRQ(ADC0_IRQn);
  NVIC_EnableIRQ(ADC0_IRQn);

  TPM0->CONTROLS[0].CnV = 0;   // i don't think we need this anymore?
  TPM0->CONTROLS[2].CnV = 0;


  threshold4[0]= -1;
  threshold3[0]= 10;   //10
  threshold[0] = 127; //127                        //print out threshold value
  threshold2[0] = -10; // -10

  temp = 0xFF;

  FPTE->PCOR = (1UL << 21);  // Disable H-Bridge?

    // Reading Ping , analyzing pong
  put("\rPlease turn on power supply and press SW2\n");
    //if(SW2 == 1) // SW2 will be a pin
    while(1){
      if (FPTC->PDIR&(1UL<<17)){
        Switch = 1;
        break;
      }
    }
      FPTE->PSOR = (1UL << 21); // enables H-Bridge
      put("\rSet duty cycles using POT1 and POT2\n\r");
      while(1){
        ADC0 -> CFG2 &= ~ADC_CFG2_MUXSEL_MASK;                    // select a channel
        ADC0->SC1[0] = DIFF_SINGLE |ADC_SC1_ADCH(13);       // start conversion (software trigger) last one might be 9
        ReadValue = Read_ADC();
        ADC0 -> CFG2 &= ~ADC_CFG2_MUXSEL_MASK;                    // select a channel
        ADC0->SC1[0] = DIFF_SINGLE |ADC_SC1_ADCH(12);       // start conversion (software trigger) last one might be 9
        ReadValue2 = Read_ADC(); // need to change prescale value in InitPWM, 255 will be your MAX now and getting 255 will give you

        ReadValue= ReadValue * 11;
        ReadValue2= ReadValue2 * 11;

        TPM0->CONTROLS[0].CnV = ReadValue; // probe PTC1 to find the duty cycle
        TPM0->CONTROLS[2].CnV = ReadValue; // POT1 & POT2
```

NATCAR.c*

```
580         holda = ReadValue;
581         holdb = ReadValue;
582
583         PW1 = 4500;
584 //        TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
585
586         if (FPTC->PDIR&(1UL<<13))
587           break;
588     }
589       put("done");
590
591   Init_PIT(40000);                                  // count-down period = 100 us
592
593   Start_PIT();
594   //PIT main end
595
596   //calculations
597   while(1) {
598     // Reading Ping , analyzing pong
599     if (printflag2 == 1){                 //pong buffers are filled up
600       printflag2 = 0;
601       if(uart0_getchar_present()) {
602         key = uart0_getchar();
603           if(key == 'p'){
604             Stop_PIT();
605             put("\r\n\r\nPong Buffer:\r\n");
606            //printing the last buffer
607            for(j=0; j<128; j++){
608             hexconvu(pong[j]);
609             uart0_putchar(' ');
610            }  //for
611           put("\r\n\n");
612
613            put("\r\n\r\nPong2 Buffer:\r\n");
614            for(j=0; j<128; j++){
615              hexconvu(pong2[j]);
616              uart0_putchar(' ');
617          }  //for
618          put("\r\n\n");
619
620       //slope values
621       put("\r\n\r\nPong Slope Buffer:\r\n");
622       for(l=1;l<127;l++){                          //printing out slope values
623         hexconvs(slopepo[l]);
```

NATCAR.c*

```
624         uart0_putchar(' ');
625       }
626         put("\r\n\n");
627       put("\r\n\r\nPong2 Slope Buffer:\r\n");
628       for(l=1;l<127;l++){                          //printing out slope values
629         hexconvs(slopepo2[l]);
630         uart0_putchar(' ');
631       }
632         put("\r\n\n");
633         // Lab 3 ends
634
635       hexconvu(ReadValue3);
636       put(" ");
637       hexconvu(ReadValue4);
638       put(" ");
639       put("\r\n\n");
640
641       put("Enter 'c' to continue or 'q' to quit\r\n");
642       key = uart0_getchar();
643       if (key == 'c') {
644         Start_PIT();
645       }
646
647       else if (key == 'q') {
648         put("\r\n");
649         break;
650       }
651     }
652   }
653     else{                                                //calculating slope and printing them out as 1 or 0
654       slopecalc1(pong,linescanpong);
655         put(" ");
656
657       slopecalc2(pong2,linescanpong2);
658
659       // more servo stuff
660
661
662       for(i=5; i<=121;i++){
663         if(linescanpong[i] == 1){ // maybe need to detect if has more 1s
664           lightcount1++; // counts how many indexes are light
665         }
666         if(linescanpong2[i] == 1){
667           lightcount2++;
```

NATCAR.c*

```
667           lightcount2++;
668         }
669       }
670
671       if(Ecount <= 3){
672         ErrArr1[Ecount] = lightcount1 - lightcount2;
673         Ecount++;
674       }
675
676       if((lightcount1 - lightcount2) <= 10 && (lightcount1 - lightcount2) >= -10){
677         //PW1 = 4500;
678         PW1 = ((lightcount1 - lightcount2)*12) + 4500+ DerivPortion ; // set duty servo direction
679         TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
680       }
681       else{
682         //if(lightcount1<lightcount2){
683         PW1 = ((lightcount1 - lightcount2)*12) + 4500 + DerivPortion; // set duty servo direction
684         //TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
685       }
686
687         if(PW1 > 4700){
688           turnspeed = ReadValue2 + 86;
689           ReadValue2 = turnspeed;
690           ReadValue = 0;
691         }
692         else if(PW1 < 4300){
693           turnspeed = ReadValue + 86;
694           ReadValue = turnspeed;
695           ReadValue2 = 0;
696           //PW1 = 3000;
697           //PW1 = ((lightcount1 - lightcount2)*20) + 4500 + DerivPortion; // set duty servo direction
698           //TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
699         }
700
701       if((lightcount1 - lightcount2) <= 45 && (lightcount1 - lightcount2) >= -45){
702         ReadValue = holda;
703         ReadValue2 = holda;
704       }
705
706       //TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
707
708       if(Ecount>3){ // only works for the first condition
709         ErrArr1[0] = ErrArr1[1];
710         ErrArr1[1] = ErrArr1[2];
```

NATCAR.c*

```
710         ErrArr1[1] = ErrArr1[2];
711         ErrArr1[2] = ErrArr1[3];
712         ErrArr1[3] = lightcount1-lightcount2;
713         DerivPortion = 4.167*((ErrArr1[0] - 3*ErrArr1[2]) + (3*ErrArr1[1] - ErrArr1[3]));
714       }
715
716       lightcount1 = 0; // reset light counters
717       lightcount2 = 0;
718       put("\n\r");
719     }
720   }
721
722     ////////////////////////////////////////////////////////////////////////////////////////////
723
724   else if (printflag1 == 1){
725       printflag1 = 0;
726       if(uart0_getchar_present()) {
727         key = uart0_getchar();
728           if(key == 'p'){
729             Stop_PIT();
730             put("\r\n\r\nPing Buffer:\r\n");
731            //printing the last buffer
732            for(j=0; j<128; j++){
733             hexconvu(ping[j]);
734             uart0_putchar(' ');
735            }//for
736           put("\r\n\n");
737
738            put("\r\n\r\nPing2 Buffer:\r\n");
739            for(j=0; j<128; j++){
740              hexconvu(ping2[j]);
741              uart0_putchar(' ');
742           }//for
743           put("\r\n\n");
744
745       //slope values
746       put("\r\n\r\nPing Slope Buffer:\r\n");
747       for(l=1;l<127;l++){                          //printing out slope values
748         hexconvs(slopepi[l]);
749         uart0_putchar(' ');
750       }
751         put("\r\n\n");
752       put("\r\n\r\nPing2 Slope Buffer:\r\n");
753       for(l=1;l<127;l++){                          //printing out slope values
```

NATCAR.c*

```
752       put("\r\n\r\nPing2 Slope Buffer:\r\n");
753       for(l=1;l<127;l++){                          //printing out slope values
754         hexconvs(slopepi2[l]);
755         uart0_putchar(' ');
756       }
757         put("\r\n\n");
758         // Lab 3 ends
759
760       hexconvu(ReadValue3);
761       put(" ");
762       hexconvu(ReadValue4);
763       put(" ");
764       put("\r\n\n");
765
766       put("Enter 'c' to continue or 'q' to quit\r\n");
767       key = uart0_getchar();
768       if (key == 'c') {
769         Start_PIT();
770       }
771
772       else if (key == 'q') {
773         put("\r\n");
774         break;
775       }
776     }
777   }
778     else{
779       slopecalc1(ping,linescanping);
780       put(" ");
781
782       slopecalc2(ping2,linescanping2);
783
784       put("\r\n");
785
786      //SERVO STUFF
787
788       for(i=5; i<=121;i++){
789         if(linescanping[i] == 1){ // maybe need to detect if has more 1s
790           lightcount1++; // counts how many indexes are light
791         }
792         if(linescanping2[i] == 1){
793           lightcount2++;
794         }
795       }
```

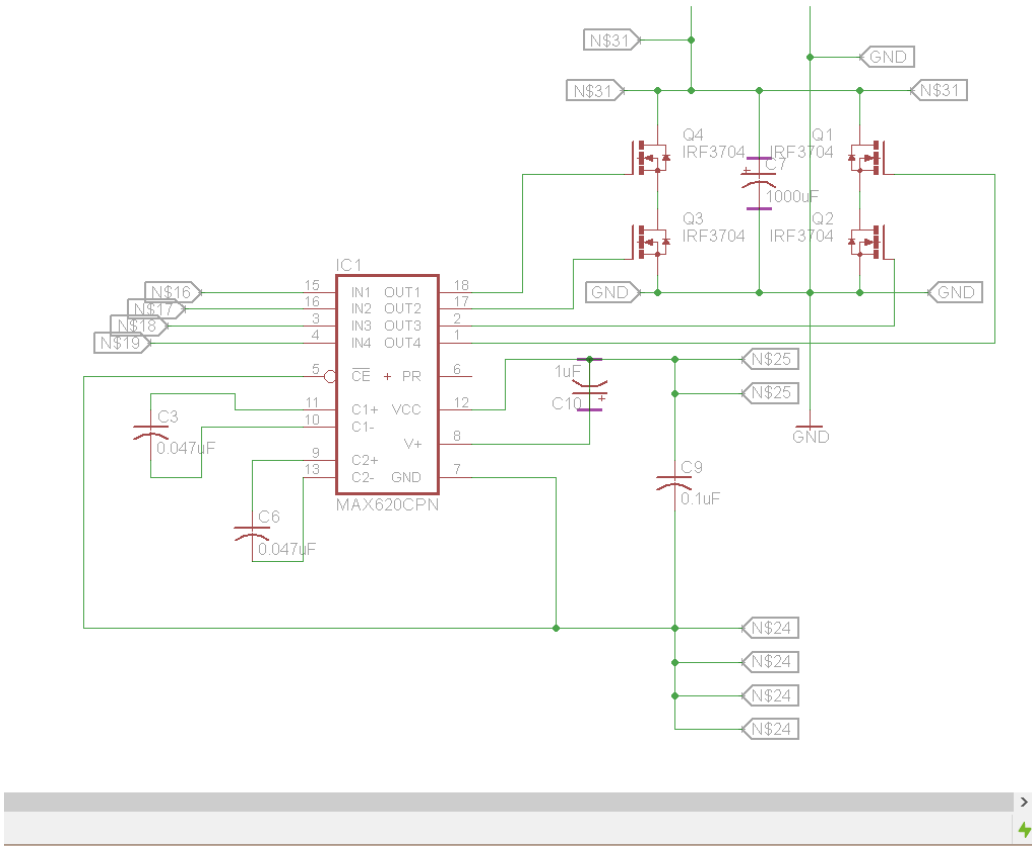
NATCAR.c*

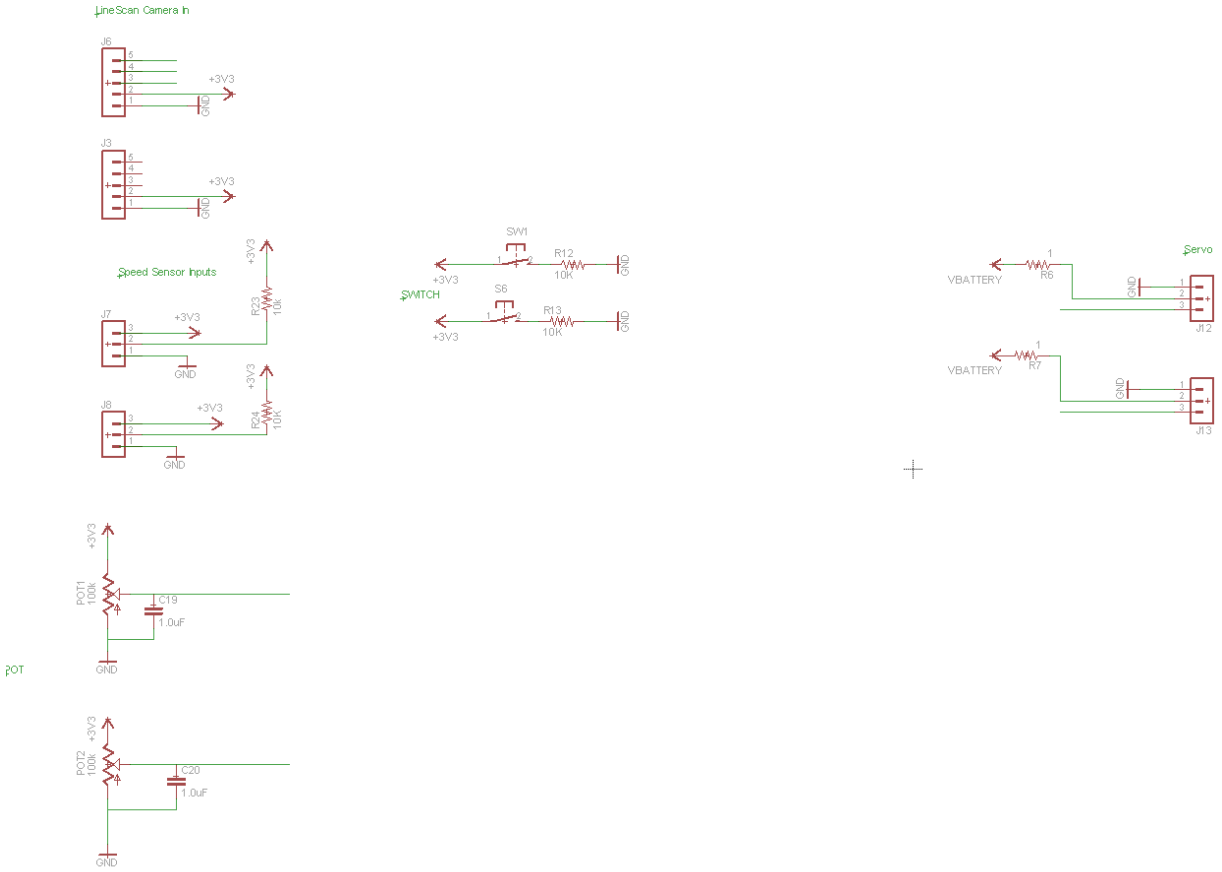
```
796
797       if(Ecount <= 3){
798         ErrArr1[Ecount] = lightcount1 - lightcount2;
799         Ecount++;
800       }
801
802       if((lightcount1 - lightcount2) <= 10 && (lightcount1 - lightcount2) >= -10){
803         //PW1 = 4500;
804         PW1 = ((lightcount1 - lightcount2)*12) + 4500+ DerivPortion ; // set duty servo direction
805         TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
806       }
807
808       else{
809         //if(lightcount1<lightcount2){
810         PW1 = ((lightcount1 - lightcount2)*12) + 4500 + DerivPortion; // set duty servo direction
811         //TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
812       }
813
814         //}
815         //else{
816         //PW1 = ((lightcount1 - lightcount2)*10) + 4500 - DerivPortion; // set duty servo direction
817         //}
818         if(PW1 >4700){
819           turnspeed = ReadValue2 + 86;
820           ReadValue2 = turnspeed;
821           ReadValue = 0;
822           //PW1 = 6000;
823           //PW1 = ((lightcount1 - lightcount2)*20) + 4500 + DerivPortion; // set duty servo direction
824           //TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
825         }
826         else if(PW1 < 4300){
827           turnspeed = ReadValue + 86;
828           ReadValue = turnspeed;
829           ReadValue2 = 0;
830           //PW1 = 3000;
831           //PW1 = ((lightcount1 - lightcount2)*20) + 4500 + DerivPortion; // set duty servo direction
832           //TPM1->CONTROLS[0].CnV = PW1;      // hopefully this works!
833         }
834
835       if((lightcount1 - lightcount2) <= 45 && (lightcount1 - lightcount2) >= -45){
836         ReadValue = holda;
837         ReadValue2 = holda;
838       }
839
```

```
840
841
842       if(Ecount>3){ // only works for the first condition
843         ErrArr1[0] = ErrArr1[1];
844         ErrArr1[1] = ErrArr1[2];
845         ErrArr1[2] = ErrArr1[3];
846         ErrArr1[3] = lightcount1 - lightcount2;
847         DerivPortion = 4.167*((ErrArr1[0] - 3*ErrArr1[2]) + (3*ErrArr1[1] - ErrArr1[3]));
848       }
849
850       lightcount1 = 0; // reset light counters
851       lightcount2 = 0;
852
853     }
854   }
855 } // BIG while loop
856
857 }//main
```

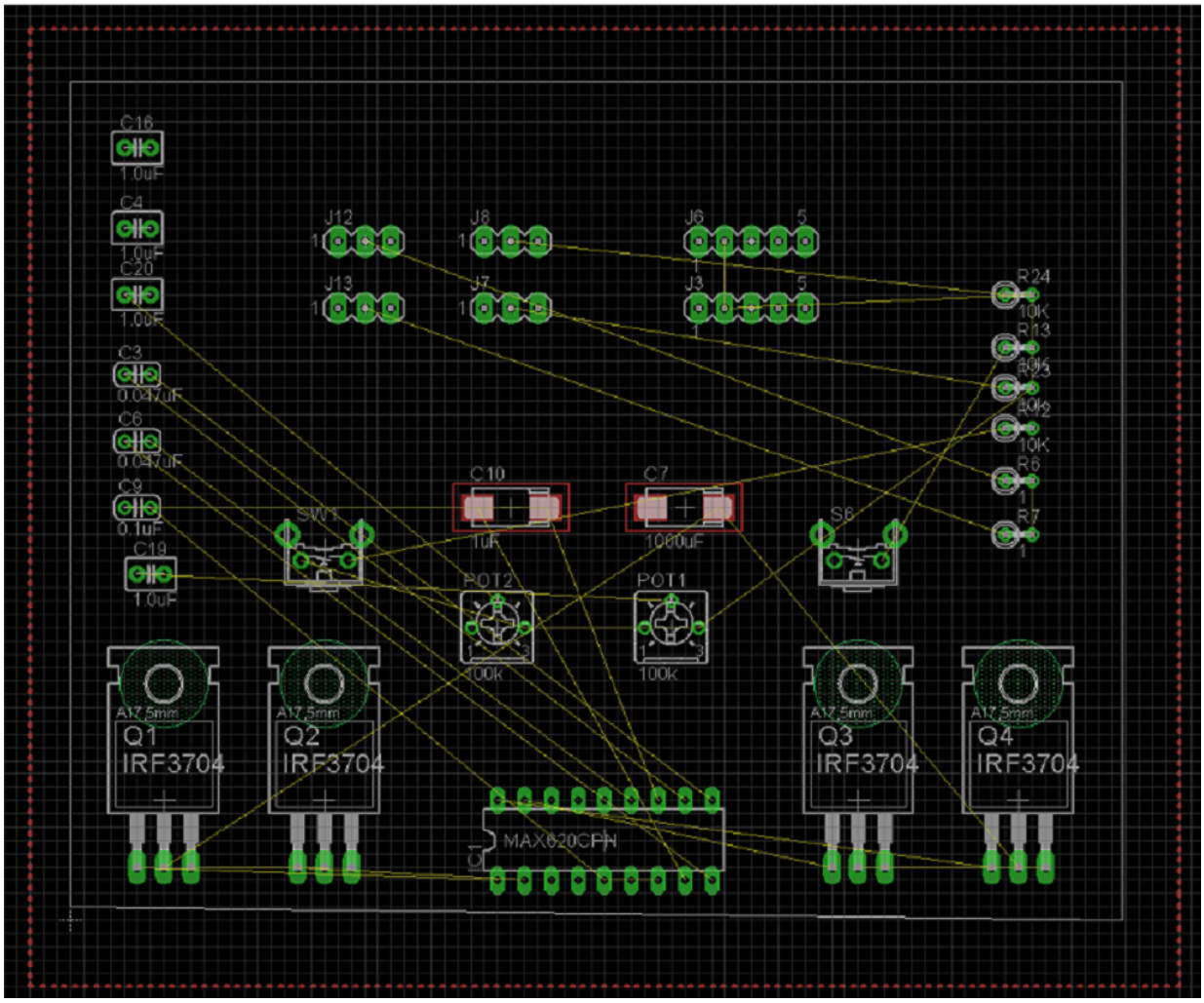
Pictures of schematic and layout included for both competition 2 and PCB:

PCB schematic:




LineScan Camera In
J6
+3V3
GND
J3
+3V3
GND
Speed Sensor Inputs
+3V3
R23
10k
J7
+3V3
GND
+3V3
R24
10k
J8
+3V3
GND
SW1
R12
10K
+3V3
GND
SWITCH
S6
R13
10K
+3V3
GND
Servo
1
VBATTERY
R6
GND
J12
1
VBATTERY
R7
GND
J13
POT
+3V3
POT1
100k
C19
1.0uF
GND
+3V3
POT2
100k
C20
1.0uF
GND

PCB layout:


C16
1.0uF
C4
1.0uF
C20
1.0uF
C3
0.047uF
C6
0.047uF
C9
0.1uF
C19
1.0uF
J12
J13
J8
J7
J6
J3
5
R24
10K
R13
10K
R6
R7
C10
1uF
C7
1000uF
SW1
S6
POT2
100k
POT1
100k
A17.5mm
Q1
IRF3704
Q2
IRF3704
Q3
IRF3704
Q4
IRF3704
MAX620CPN

Pictures of various camera designs: