EECS 192 Progress Report

Ian Krase, Daniel Lee, Nicholas Gan

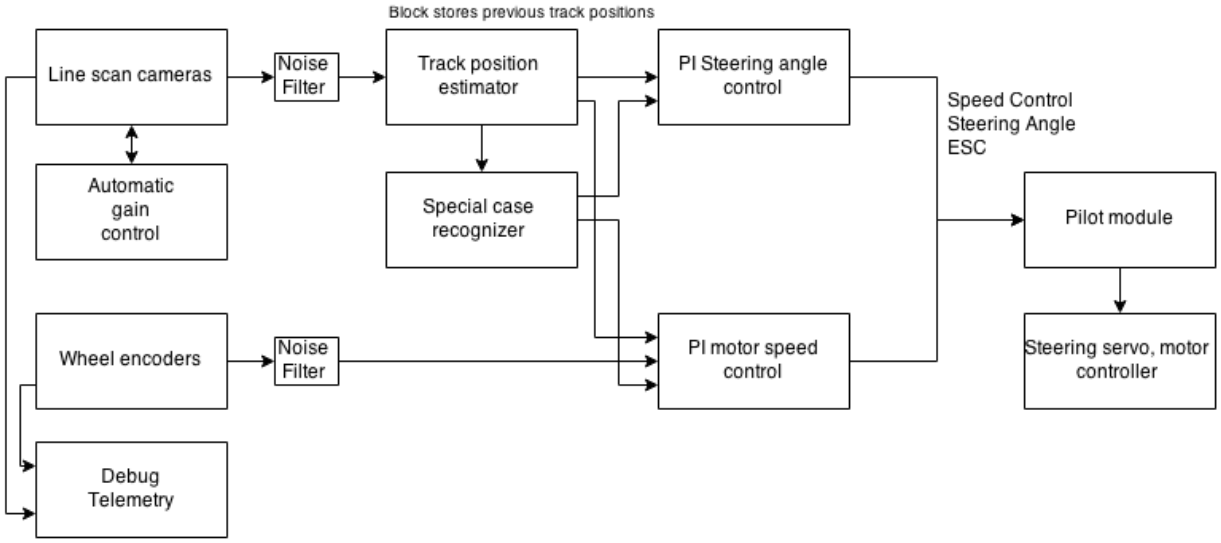
1. **Current State of Project**

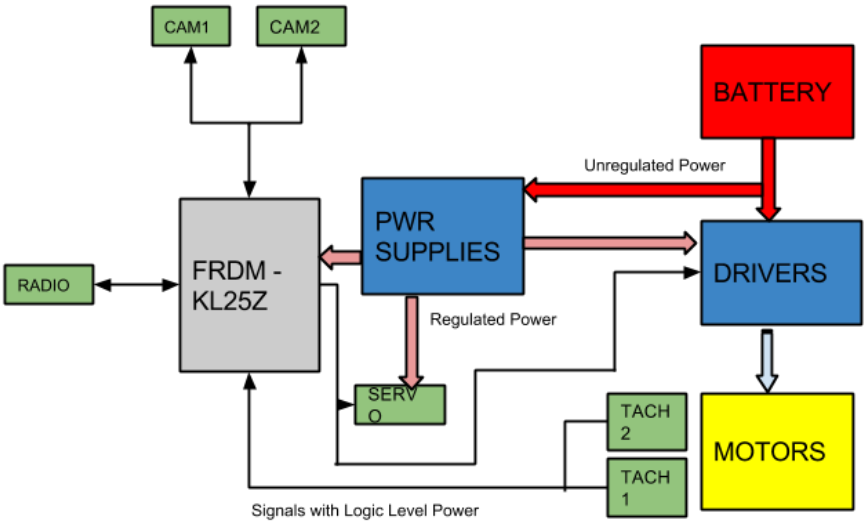
Currently, our car is using largely the same hardware as we originally constructed, with one change: we are adding a second camera (with a further scan distance) to the top of our tower. Ways of bracing the tower to reduce mechanical vibrations are under investigation, but it appears that the flexibility is actually mostly within the car chassis. Active lighting is under consideration.

The electronics are largely as originally designed, and quite simple, with no sensors besides the two encoders and two cameras. The boost converter/microcontroller connection board has been equipped with a dip switch to allow setting of configuration options, and the motor driver has been re-laid-out to use surface mount FETs and match the dimensions of the motor and gearbox frame.

Our car runs five major modules that handles the following tasks: read and process wheel encoder inputs, read and process linescan camera inputs, steering control, motor speed control, and user interface. Each module is assigned it's own thread to allow independent operation. Modules that are not functioning at full capacity are the user interface, which is able to communicate via serial communications over bluetooth but unable to transmit telemetry data, and the motor control, which currently does not vary the velocity according to the track data.

The following subsystems require some debugging: the automatic gain control in the linescan camera module which behaves oddly in bright light, steering algorithm to handle bad data (e.g. noise or being blinded), and the telemetry output which is in development.





2. Hardware Documentation

- **Hardware Overview**

In order to keep clear of mechanical attachments, and to make up for the limitations of available CAD software, the electronics have been divided into two boards: a motor driver board which is attached directly over the motor and gearbox frame of the car, and a boost converter and control board which attaches to the top of the FRDM-KL25Z microcontroller module. This unit is then attached over the steering servo, making the cable to the sensors and servos short. To provide 5v power for logic and for the servo, a 5V linear regulator is used. In addition to carrying the configuration switches, linear regulator and boost converter, this board includes signal and distribution to all part of the car, including 5V logic power and 7.2V boosted power for the motor driver. A cable is used to attach a Sparkfun BlueSMIRF radio modem for remote debugging.

To provide power, the Natcar and TFC legal 7.2V battery is directly connected to the input for the motor controller. From the motor controller board is output moderate-current battery voltage to the boost converter, which boosts the battery voltage to 7.2V if it falls below this level due to a dying battery or internal resistance. This 7.2V moderate-current boosted supply is then fed to the motor driver board (where it powers the gate predriver) as well as to the FRDM-KL25Z module and a moderate-current 5V linear regulator. The resulting 5V is used to power the servo and sent to the motor driver board to provide logic level supply (after appropriate filtering.)

The motor drivers consist of a pair of simple half-bridges (providing only forward rotation and braking), with the high-side N-channel FETs driven by a MC33883 gate predriver. A toggle switch with detachable bamboo wand allows the motors to be enabled and disabled without affecting any other part of the car. A set of 7400-series logic gates provides both level-shifting (the MC33883 is not natively

capable of being controlled by 3.3V CMOS signals) and enforces shoot-through protection.

To measure the speed of the wheels, a pair of the ESG-designed reflective sensors are mounted between the wheels and the frame, in position to read the reflective disks on the backside of the wheels with two marks per revolution. Since the car only travels forward, quadrature encoders are judged unnecessary.

A carbon and glass fiber tripod tower supports two mounts for TFC linescan cameras at adjustable angle. Currently only the lower (scanning closer to the car) mount is populated. It is planned that the upper mount will soon be added and used to distinguish approaching curves from tracking error.

- **Schematics and Board Layouts**

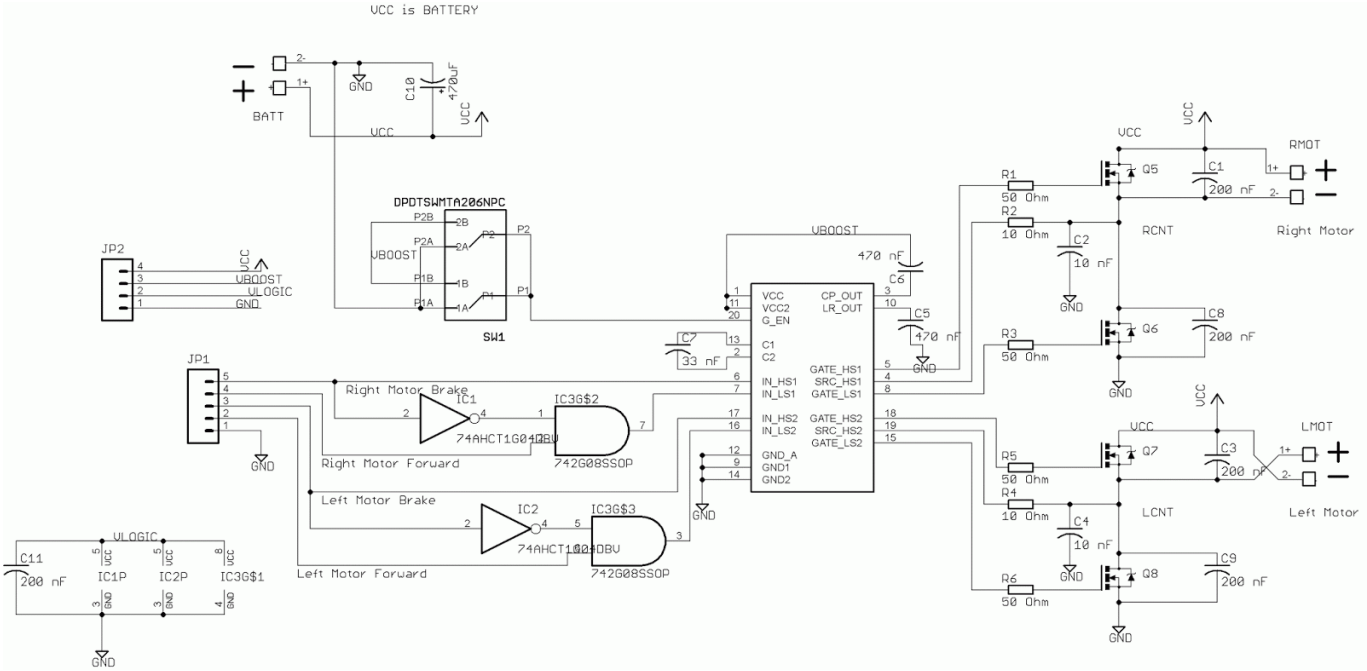


Figure 1: Motor Driver

Figure 2: Power connections

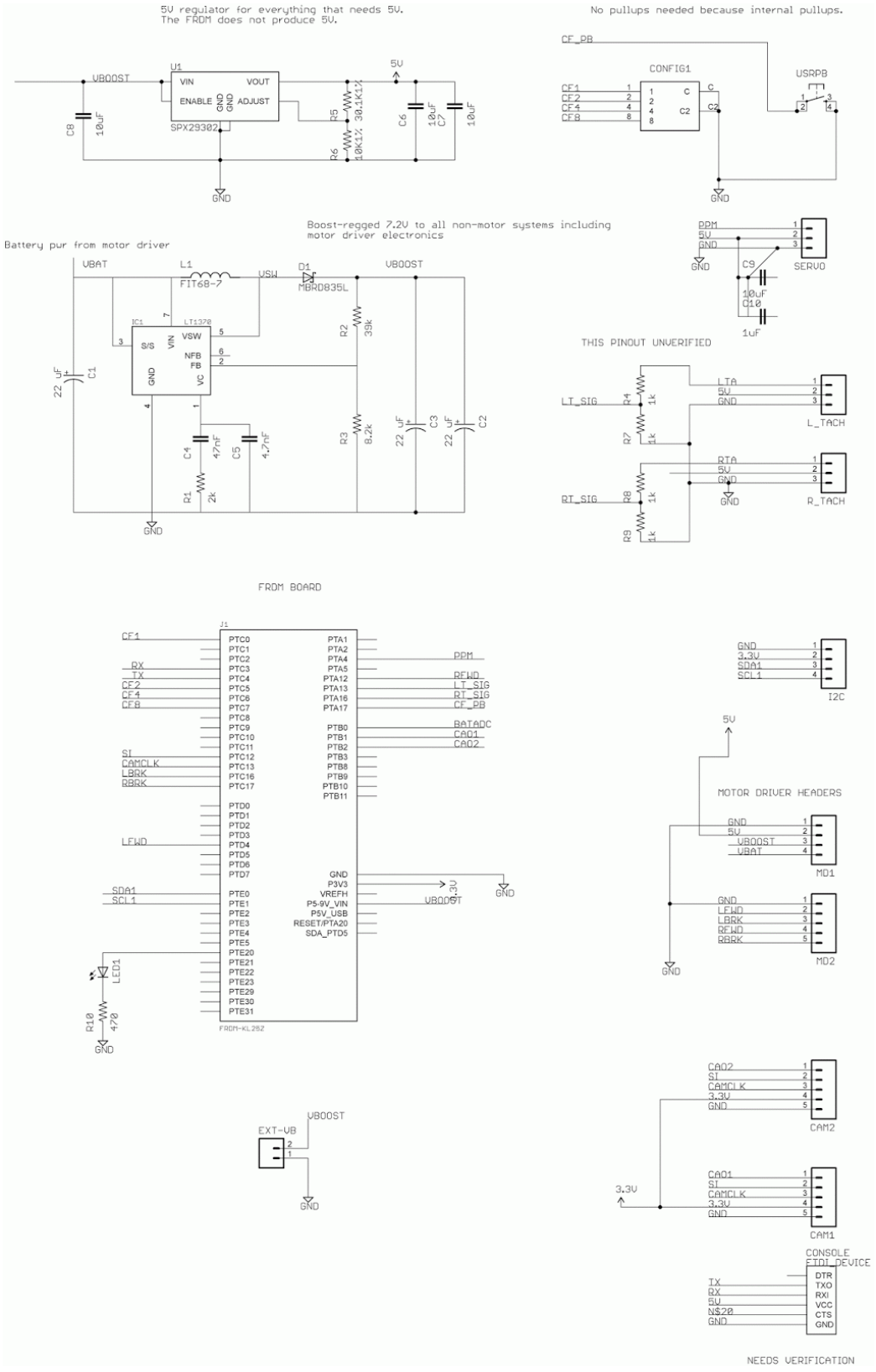


Figure 3: Boost Converter / Control Board Schematic

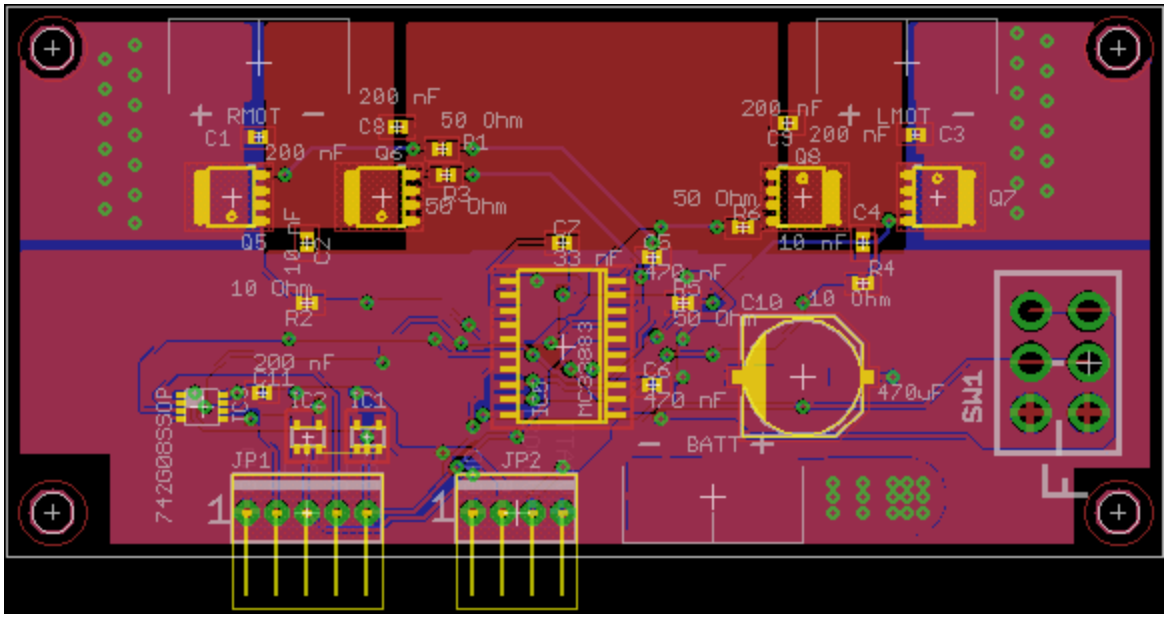


Figure 4: Motor Driver Board Layout

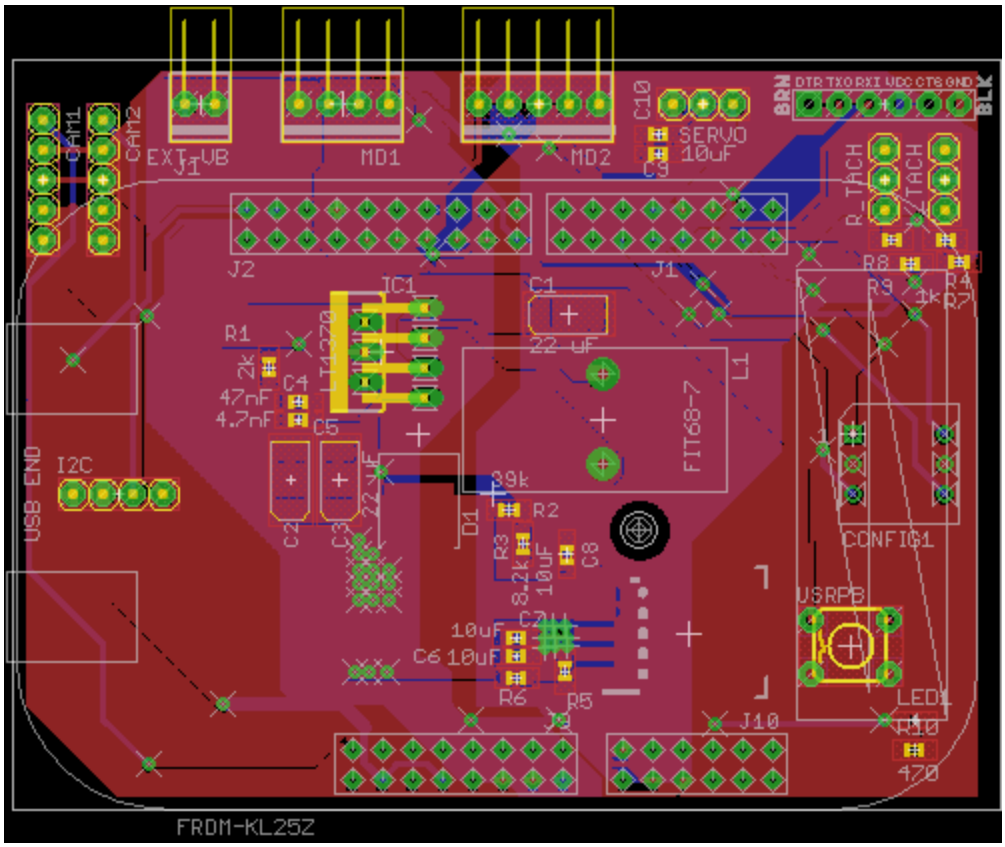


Figure 5: Boost Converter Board Layout

- **Pin Assignments Table**

	PTA1	**PTC7**	DIPSW8		**PTE30**	**(RTC)**	
	PTA2	**PTC0**	DIPSW1		**PTE29**	**PTC2**	
	PTD4	**PTC3**	RF_RX		**PTE23**	**PTB3**	
RMOTOR_FWD	**PTA12**	**PTC4**	RF_TX	LMOTOR_FWD	**PTE22**	**PTB2**	CAM2_IN
SERVO	**PTA4**	**PTC5**	DIPSW2		**PTE21**	**PTB1**	CAM1_IN
	PTA5	**PTC6**	DIPSW4		**PTE20**	**PTB0**	
	PTC8	**PTC10**					
	PTC9	**PTC11**			**PTE5**	**(VIN)**	TO BOOST
					PTE4	**(GND)**	
L_TACH	**PTA13**	**PTC12**	CAM_SI		**PTE3**	**(GND)**	
	PTD5	**PTC13**	CAM_CLK		**PTE2**	**(5V)**	
	PTD0	**PTC16**	LMOTOR_BRK		**PTB11**	**(3.3V)**	OUT
	PTD2	**PTC17**	RMOTOR_BRK		**PTB10**	**(RST)**	
	PTD3	**PTA16**	R_TACH		**PTB9**	**(3.3V)**	
	PTD1	**PTA17**	PBTN		**PTB8**	**(SDA)**	
	(GND)	**PTE31**					
	(VREF)	**(NC)**					
	PTE0	**PTD6**					
	PTE1	**PTD7**					
	User USB				**Debug USB**		

Table 1: Pin Assignments

DIPSW1:8	Inputs for 1's,2's,4's, and 8's of configuration dipswitch
PBTN	Input (with interrupt) for user pushbutton
R_TACH, L_TACH	Right and left encoder inputs (interrupts)
CAM_SI	Shared Camera Start Capture output
CAM_CLK	Shared Camera Pixel Clock output
CAM1_IN, CAM2_IN	Camera 1 and 2 analog inputs
SEVO	Steering servo PPM output
RMOTOR_FWD, LMOTOR_FWD	PWM outputs for left and right motor-forward pins on half-bridge
RF_RX, RF_TX	Serial UART IO for Bluetooth radio modem
RMOTOR_BRK, LMOTOR_BRK	Outputs to braking pins of half-bridge

Table 2: Pin Definitions

3. Proposed Control Methods

○ Steering Control

The latest code implements a proportional-integral controller. First, we mapped our possible range of steering angles to floating points between 0 and 1 where 0 corresponds to left lock and 1 corresponds to right lock. Second, we designed the line scan cameras to output a floating point value of 0 to correspond to the leftmost point in its field of view and 1 to the rightmost point.

The control system takes in the sensed line position from the linescan camera and subtracts the reference value corresponding to the center of the camera's field of view (we plan to make this reference an input value later). We accumulate this calculated error for the integral part of the controller. We then calculate the steering angle output as:

$$output = error \times Kp + accumulatedError \times Ki$$

A couple of improvements we plan on adding are: a differential part to the controller for faster rise time, a memoization buffer to smooth out the data output from linescan cameras so we can reduce jitter, additional tuning to the automatic gain control so it will be less erratic in bright light, support for a reference input, and a handler for situations when the linescan camera is blinded (memoization may make this unnecessary). We also plan on merging the linescan and steering control threads to reduce concurrency issues that seem to happen in low exposure times.

- **Velocity Control**

  Currently, we have a proportional-integral controller similar to the one used with the steering control. It takes in a target velocity, a float specifying speed in m/s, and outputs a percentage increase or decrease from the current PWM duty cycle to the motors in an attempt to make the car match that speed. THe encoder captures the actual speed and feeds this information back to the controller.

  We have yet to implement a dynamic speed controller that responds to detected line position. However, we've found through simulations that a speed controller that varies speed about proportional to the rate of change of the error from the linescan camera best. To achieve this, we will have a second linescan camera look ahead of the camera used in steering controls. With the two points of the line we can estimate the gradient of the track and decide on whether to accelerate or brake.

  Our dynamic speed controller will output the target speed for the motors to aim for. It will take in a reference of $ref = 1 - (lookahead - steering)$ where lookahead and steering are the steering angles calculated from the lookahead and steering cameras. Then it will apply a PID controller to that reference. Given the time, we will add special cases to handle late hard braking and full throttle acceleration.

- **How do you propose to stabilize the system?**

  The biggest threat to stability we face was identified to delays in the system. Even tiny time delays can move a system into instability. We combated this issue by angling our cameras far forward such that we cancel out the delay and possibly "look to the future", as such improving stability.

4. **<u>Interim Budget</u>**
   - **Time Budget**

The most time-consuming parts of the project thus far have been coding and software debugging. We have spent comparatively little time on hardware, especially recently, as we have not needed to make many modifications to our hardware past the original build.

Car disassembly/reassembly:	3 h
Tower Build	3 h
Tower Design	1 h
Electronics Design	12 h
Electronics Construction	6 h
Misc HW time costs	4 h
Framework programming	12 h
Device Driver Programming	5 h

- **Monetary Budget**

  No money has yet been spent out-of-pocket. A variety of items have been purchased from class budget. A length of fiberglass rod was supplied from a group member's personal scrap bin.

5. Refined Proposal for Software Architecture

Currently, all planned software modules have been implemented in one form or another. The major components are the line position estimator, automatic gain control, variable speed control, steering control, and debug telemetry.

- The line position estimator is not yet robust enough at crossings, memoization of previous track position and track signal width detection are required.
- AGC performance is subpar in high light conditions. Testing is required.
- The speed control currently only attempts to maintain a constant speed. Instead it should take in track data and speed up in straights, slow down in turns, etc. Control is currently only PI.
- steering control is currently only PI. Further testing is required to see if the D component is necessary. Further tuning of constants to follow the improvement of the AGC and line position estimator.
- Having telemetry on currently prevents the use of serial commands through bluetooth. Working to have the car accept commands through USB serial while sending telemetry over bluetooth.

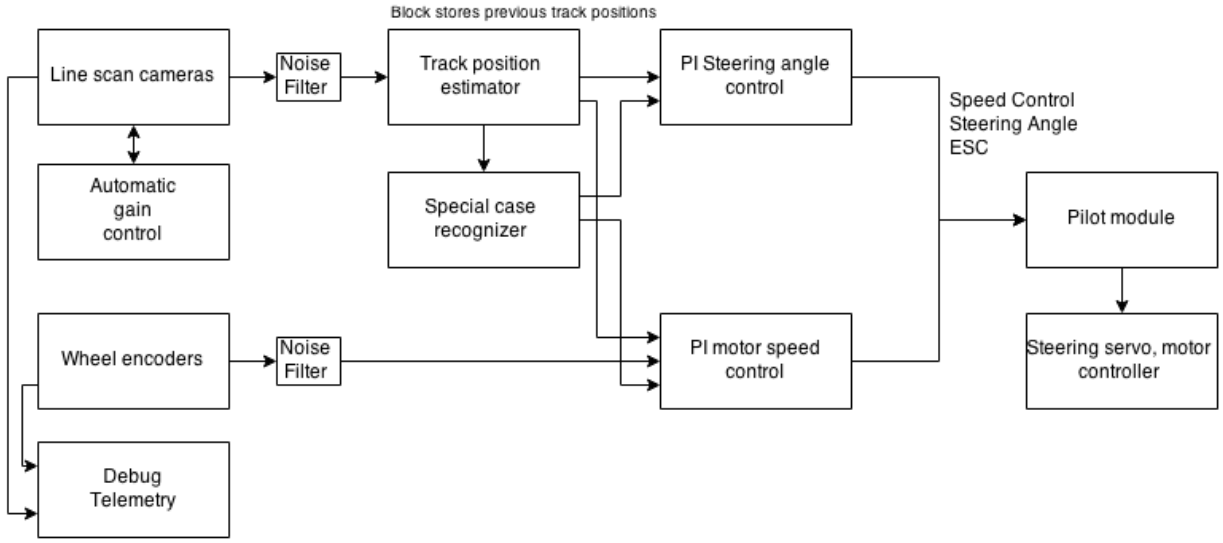


Figure 6: Software block diagram

6. Additional Resources required

- Small battery to power active lighting
- LEDs for active lighting (scavenged some from a burnt LED array)
- Spare PCBs
- Extra components, specifically transistors and motor driver boards in case of emergency.