EECS 192 MECHATRONIC DESIGN LABORATORY PROJECT PROGRESS REPORT

Xingchun Wang, Wencong Zhang, Yingzhe Fu

APRIL 6, 2015

1. Current State of Project

In general, we have been able to follow the checkpoints despite all kinds of accidents and intractable hardware and software bugs. As for hardware, we were able to assemble the second PCB last week, and it seems that all hardware is currently working as expected. Our CPU and LED headlights run from a 5.3v voltage supply from LDO which is connected to the boost converter. Our servo is run from another 6v LDO connected to the boost converter to ensure its proper working. All sensors were connected to a 3.3v LDO which is directly connected to the battery. Motor is directly connected to the battery. All modules on the motor driver circuit including logic gates, gate driver and power MOSFET have been extensively debugged. Please see section 2 hardware documentation for more details.

In terms of sensors, we have relied on one encoder and one camera for checkpoints up to now. We are planning to incorporate another camera this week. Figure 1a shows the overall block diagrams for the homework and simulation systems and the issues that still exist in our current state of the project.

Software and software-to-hardware-integration has been the bottleneck for our project. Given that none of the teammates have taken CS162 Operating System, we were audaciously ignorant to assume we could implement 5+ threads on this processor and the end result was that we were wasting a lot of debugging things we did not know. The current working version still relies on barely functioning multi-threading process which we intend to replace with simple single control loop. We have 3 line detection algorithms to handle nearly all cases of line input. Our current speed detection is based on sampling in a fixed duration (10ms). This works but it stands in the way of automatic gain control so we are planning to replace it with interrupt-based speed reading. Motor and servo PID control have been implemented but PID constants have not been optimized in the real system. What requires our immediate attention is the interrupt-based speed reading module which is essential to the robustness and controllability of the system but we have not been able to get it to work yet. Finally, the integration of different parts seems daunting but given the deterministic single loop control, we feel confident in debugging it if problems arise.

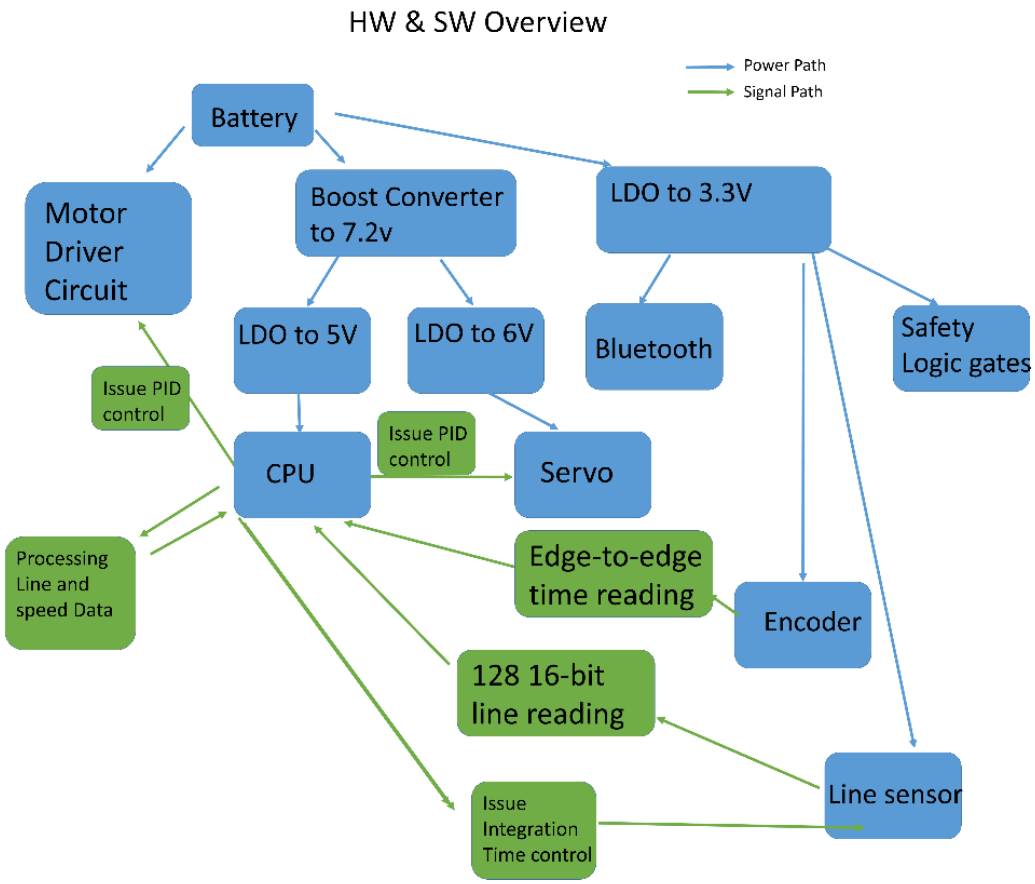

Figure 1a: overall block diagram for the HW and SW systems

2. Hardware Documentation

a) Figure 1 shows our motor drive circuitry schematic

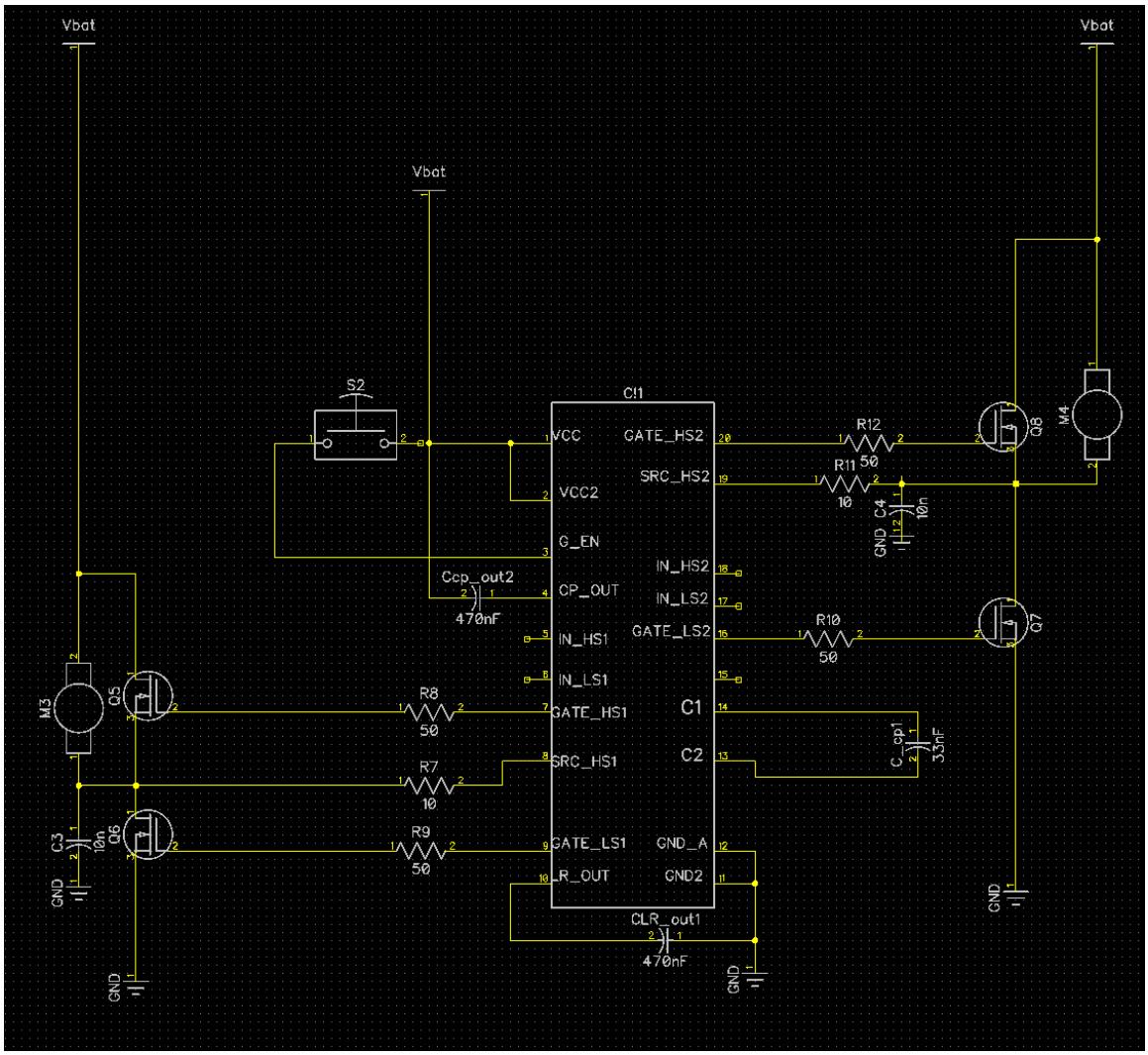

Figure 1: motor drive circuitry schematic

To prevent shot-through in the high and low MOSFET, we implemented logic gates protection. For the low side gate, the logic we implemented is

LS = LS & (~HS);

For the high side gate, we implemented:

HS = HS & (~LS);

b) We used two line sensors, two encoders, and one Bluetooth attached on the vehicle. Figure 2 shows the schematic of the connection of sensor electronics.

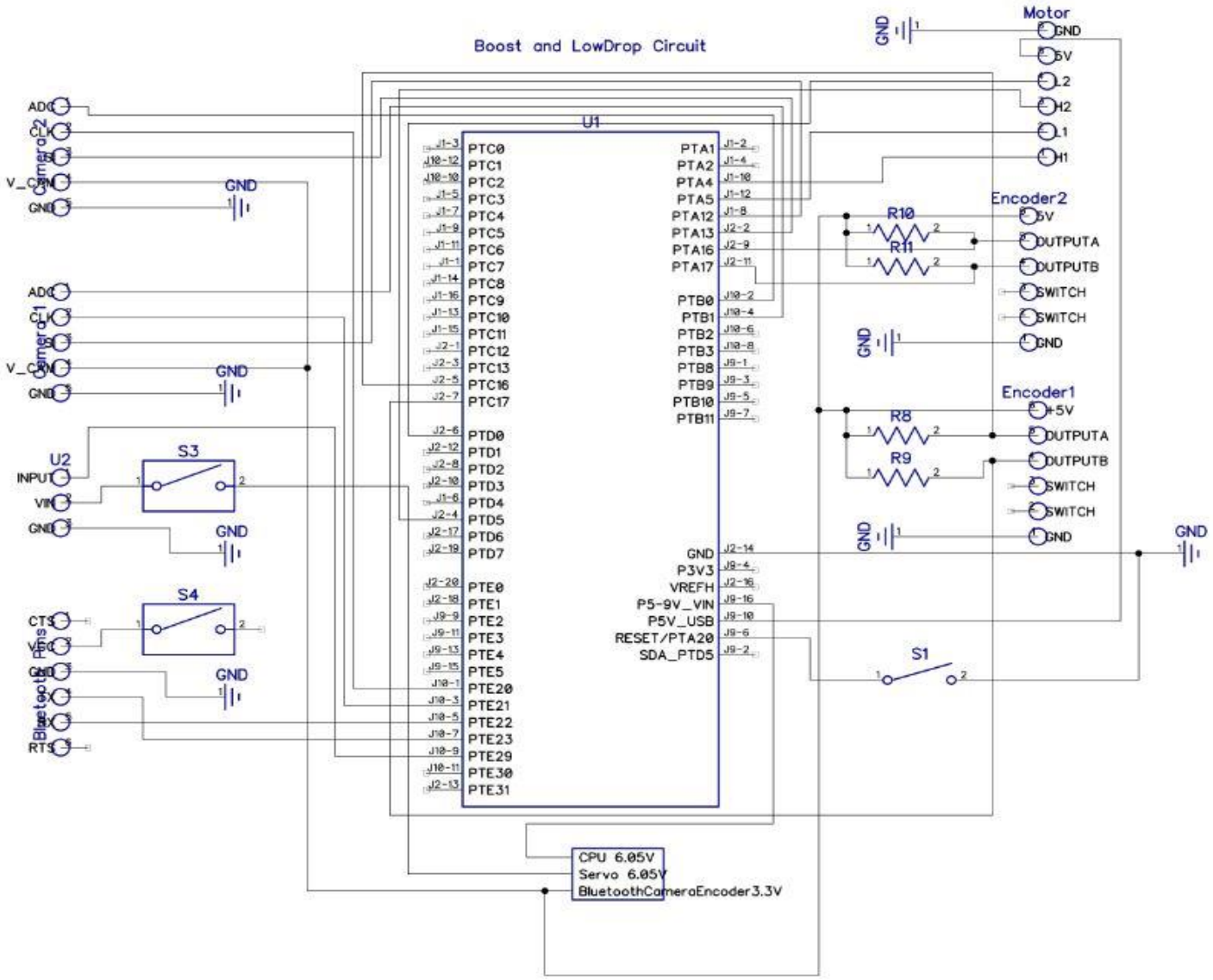

Figure 2: the schematic of the connection of sensor electronics

c) Figure 3 shows the schematics of power supplies.

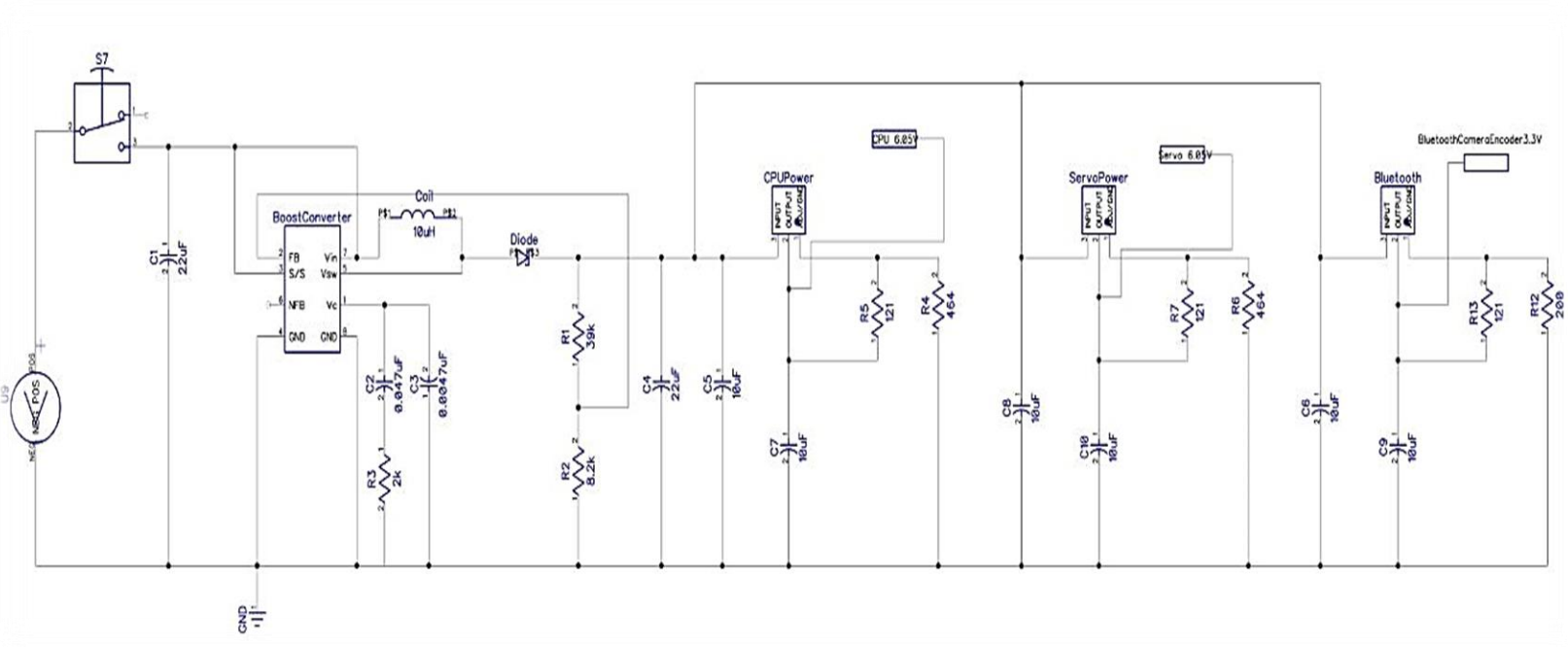

Figure 3: the schematics of power supplies

d) Figure 4 shows the top layout of the print circuit board (PCB). Figure 5 shows the bottom layout of the PCB

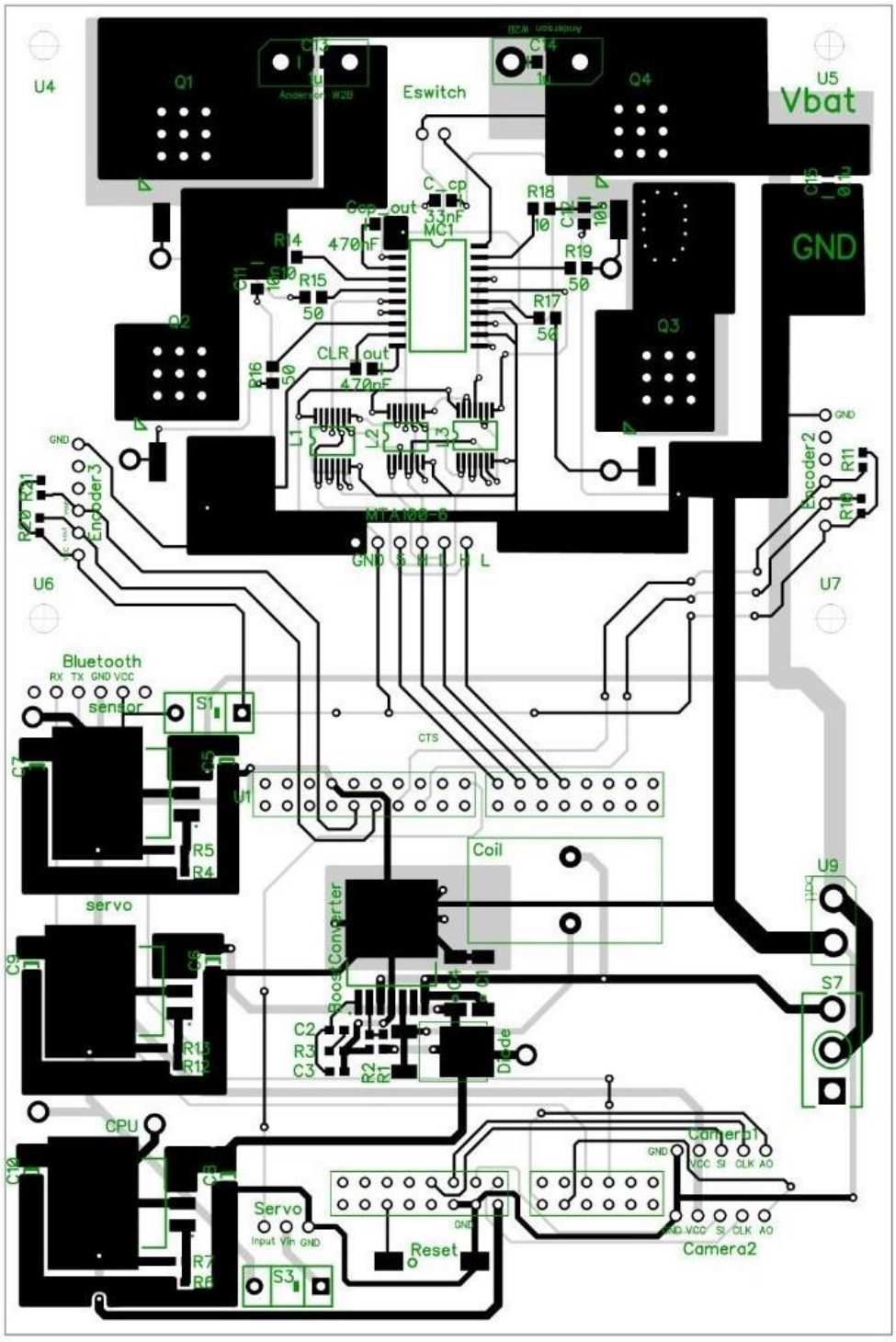

Figure 4: the top layout of the print circuit board

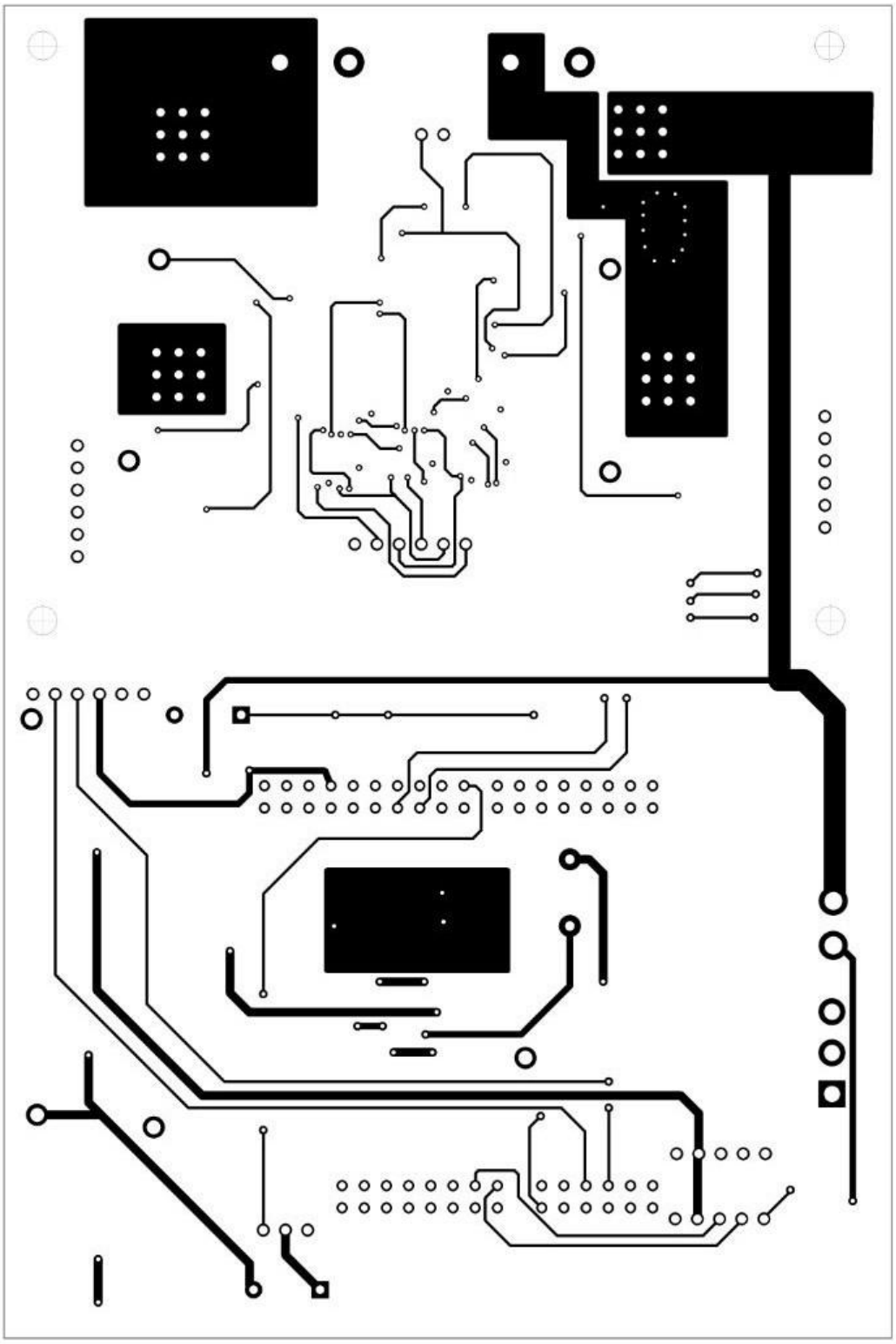

Figure 5: the bottom layout of the PCB

3. Proposed Control Methods

1) Velocity Control

We face two problems to solve when considering about velocity control: the first one is what the most appropriate speed for a given input state is; the second is how to make the motor approach the appropriate speed. Both parts are necessary and we want to address both in detail.

a) Target speed as a function of time:

The reason for target speed as a function of time is that we the car can address different track pattern at different speed. For example, at S turn, if the speed is too high, the servo controller has not enough time to response and the tracking will fail. In contrast, in the straight-line situation, the speed limit is very high. Think of the track as a set of components each of different speed upper bounds. Then the maximum overall speed is achieved when we hit each upper bounds of speed for each component. So our goal is to find by experiment what the maximum speed we can go at each different pattern and try to come up with control algorithm to achieve optimum.

Our current software structure sets the target speed as a continuous function of time. The speed is inversely proportional to the center tracking error E_{cam} by the following equation (P control):

$$V_{target} = Speed_{normal} - Kp_{tgSpeed} * \left(abs(E_{cam}) - M_{pixelLength}\right) \ldots\ldots \quad (1)$$

When tracked center is within M pixels (e.g. M=10), our speed is greater than $Speed_{normal}$, otherwise smaller than $Speed_{normal}$. This distinguishes the straight line and S-turn by curvature.

In two-camera system, use the error of far camera to control speed to make sure we brake before S turn occurs.

b) PID controller for drive motor

Now, we are given target speed. Also, we assume we are given continuous and reliable speed measurement from encoder. The goal is clearly minimize the error between target speed and speed measurement. The acceleration should be as fast as

possible and the overshoot should be within reasonable range. We propose to use discrete time incremental PID controller for this particular problem:

$$u_k = u_{k-1} + Kp(e_k - e_{k-1}) + Kie_k + Kd(e_k - 2e_{k-1} + e_{k-2}) \dots\dots (2)$$

Where e_k represents the speed error (= current speed – target speed) of the kth loop iteration. The Ki term tries to bring the steady state error e_k to zero but causes delay to fast change of target set point, while the Kd term reduces response time of the change in error, which gives us faster acceleration and deceleration. The exact determination of the PID parameters is done by experimenting on real track and looking at the real speed data using telemetry. Output is given a protection:

$$u_k = \min(u_{MAX}, \max(u_k, 0)) \,; MoterPWM = u_k$$

2) Steering Control

In steering control, we use the same form of incremental PID as drive motor. The equation is exactly the same:

$$u_k = u_{k-1} + Kp(e_k - e_{k-1}) + Kie_k + Kd(e_k - 2e_{k-1} + e_{k-2}) \dots\dots (3)$$

Where the e_k term is the difference between measured center and reference center. The Ki term try to minimize the error term e_k and the Kd term try to deal with the sharp turns.

Integration protection should be implemented and the input should be smooth to avoid rapid shaking.

The "No input" is one edge case where no camera data is found. In practice, it will be interpreted as that a sharp turning is not performed on time, so it will be soon leave the whole track (30cm overshoot maximum). One way to remedy is that when no center is found, the car use the last few inputs to determine which way the track is lost, and apply full steering to that direction.

Also, very importantly, a good control strategy should allow flexible Kp, Ki, Kd at different input pattern. These can be tuned using real experiments. (Usually, PD control is enough). The K terms should also be related to current speed. At different speed, the same steering angle means different thing actually.

A rigid set point is not always optimal. The reference center, which is usually 63, which is the center index of the line pixel from 0 to 127, can be changed to improve efficiency. When in circular

arc, the steady state error should be constantly nonzero. Our Ki term takes care of this as it usually does. But we want it to move in the inner region of the circle so that it does not get off track but complete the circle more efficiently and remove the overshoot back and forth. We can temperately set the reference center to the direction of the arc a little bit to give better tracking

The integrated control method is illustrated as following:

$[Kp, Ki, Kd] = get\_PID(calcTerrianType(\ \ ), V_{current})$;

$center_{ref} = getREF(calcTerrianType(\ \ ), V_{current})$;

$center = getCamCenter(\ \ )$;

$if\ center = NULL$:

    $steer = sign(error_{last}) * U_{max}$;

    $OutputSteer(\ \ )$;

$error_{prev} = error_{last}$;

$error_{last} = error_{now}$;

$error_{now} = center - center_{ref}$;

$Apply\ equation\ (3)$;

$OutputSteer(\ \ )$;

4. Interim Budget

In general, we have been a high-cost team, not only in terms of the components that we needed replacement, but also mainly in terms of time commitment to almost every subsystem along the way.

Mechanical construction: Reg is totally in charge of all aspects of mechanical design. Total time spent is roughly 20 – 25 person-hours.

Motor drive system: a simple system that gives us all kinds of headaches. Topology seems fine yet debugging has been a nightmare. We hold the highest record for gate driver chip kills. Total time spent is 50 – 75 person-hours.

DC – DC converter: after the initial time investment in understanding how it works and a one-time explosion of capacitor soldered in the wrong direction, DC-DC converter has been a relatively low-cost area of our design. Total time spent is 25-30 person-hours.

CPU board understanding: the CPU has not been kind to us at all. Not having worked with embedded system before, the behavior of it seems very strange to us. We strongly suggest that either the Professor or the TA should caution students next year against all pitfalls of using the board. Debugging the system such as looking for independent pins has been very time-consuming and the time cost is still adding up. Total time spent is > 100 person - hours.

Optical Sensor and encoders: we were able to get them to work relatively fast but the integration of different parts was very time-consuming. Total time spent on individual sensor: 25 person-hours.

Software: Again, the integration of parts is the most difficult and time-consuming step, especially when threads start to fight one another and we had no clue why everything just fell apart. Total time spent on all software and total integration > 75 person – hours.

Out-of-pocket monetary cost: ~ 20-30 dollars per person.

5. Refined Proposal for Software Architecture

After experimenting with multi-threads without much success, we have decided to forgo the multi-thread approach as we realize that we don't have the necessary knowledge or make it work as we would wish. Therefore, to construct a more robust and easy to debug system, we are opting for the single deterministic loop approach to minimize problems brought by asynchronous threads. Figure 5a shows our software architecture of vehicle control loop. After the initialization of all variables as the system first starts, we enter our control loop. Inside the control loop, we will trigger two cameras and obtain their line readings. These readings are buffered since servo and motor decisions are not as fast as line readings. Then we pass the line data to our line detection algorithms to find the center of the track and to spot patterns that we have pre-programmed for. PID control then kicks in to adjust servo and motor PWM. Threshold-based automatic gain control (control exposure time such that the white line/black line have certain values) then runs to determine how long we need to wait to start the next control cycle. IO is running during the wait time to print useful data for debugging and tuning. Other than the trigger camera input stage where we read from the line sensor, speed is measured by an interrupt based module where we measure the time elapsed from edge to edge in the encoder output and calculate the corresponding speed.

For the preliminary round, we need to have all modules listed below integrated and functioning properly. Other than the integration of the ISR into the main control loop, other pieces of the software should not have timing conflicts or result in difficult bugs. Given that most functions have been implemented, all that is left is testing and proper integration of different pieces. The most challenging task right now is to implement a robust interrupt-based speed reading module that functions properly when integrated into the whole control loop.

To switch from a multi-thread approach to a single control loop approach, we think roughly 5- 10 hours are required for the proper working of the system. On top of that, 5 – 10 hours are required for the development of ISR based speed reading and the integration of which to the main control loop. After that, we expect another 10-20 hours of work in testing and tuning PID for optimal performance. Therefore, we expect at least 40 hours of work in software before the preliminary contest round. After that, we can experiment with fancier algorithms such as track memorization or better track recognition so that we can deal with some cases where our

control is not a one-size-fits-all solution. The time on such fancier algorithms is estimated to be pretty large and we would like to limit ourselves from exploring too many options since the more variables we have, the harder we can comprehend and improve the car behavior.

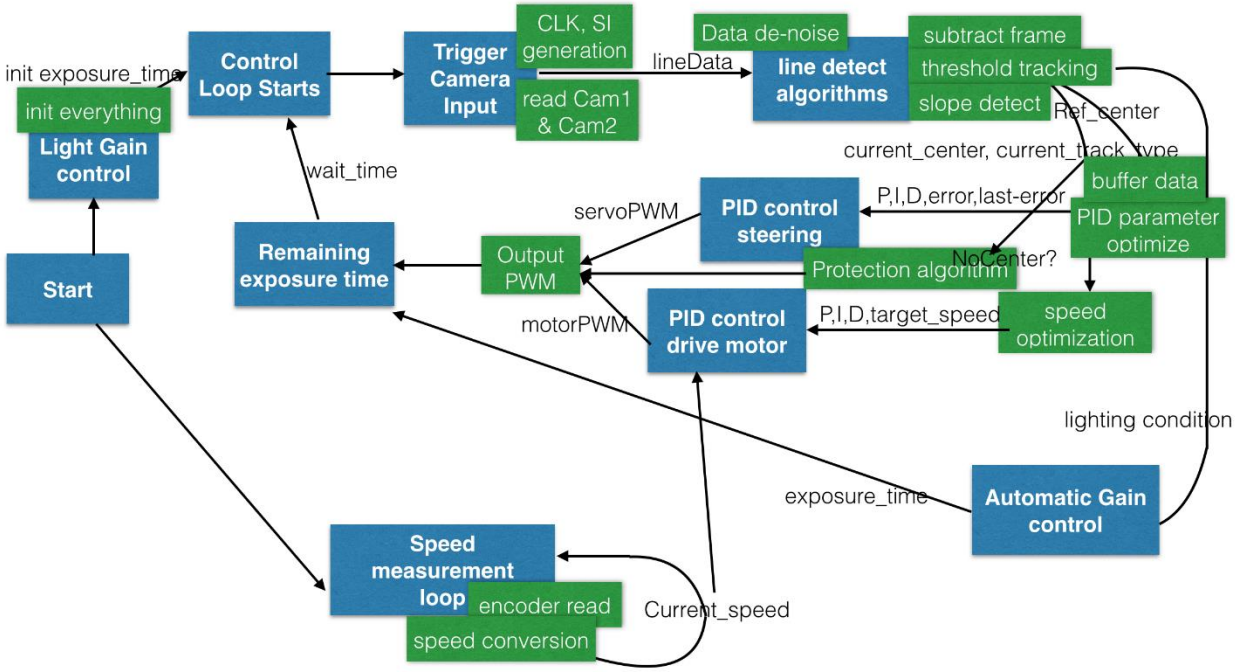

Figure 5a: software architecture of control loop

6. Additional Resources required

As we are almost done with all mechanical parts and electronic hardware, we are going full-speed with our software development and integration of different parts. One additional hardware we require is the roller for the encoder. We need an ideal material that has sufficient resistance to reliably roll the shaft of the encoder as our wheel rolls. Otherwise, we need more help understanding how software such as interrupt works on the mbed and debugging some peculiar behaviors. Ideally, we would like to have more resources to help us with the software

development since the resources and documentation on the ARM mbed website are not as sufficient as we would like them to be.