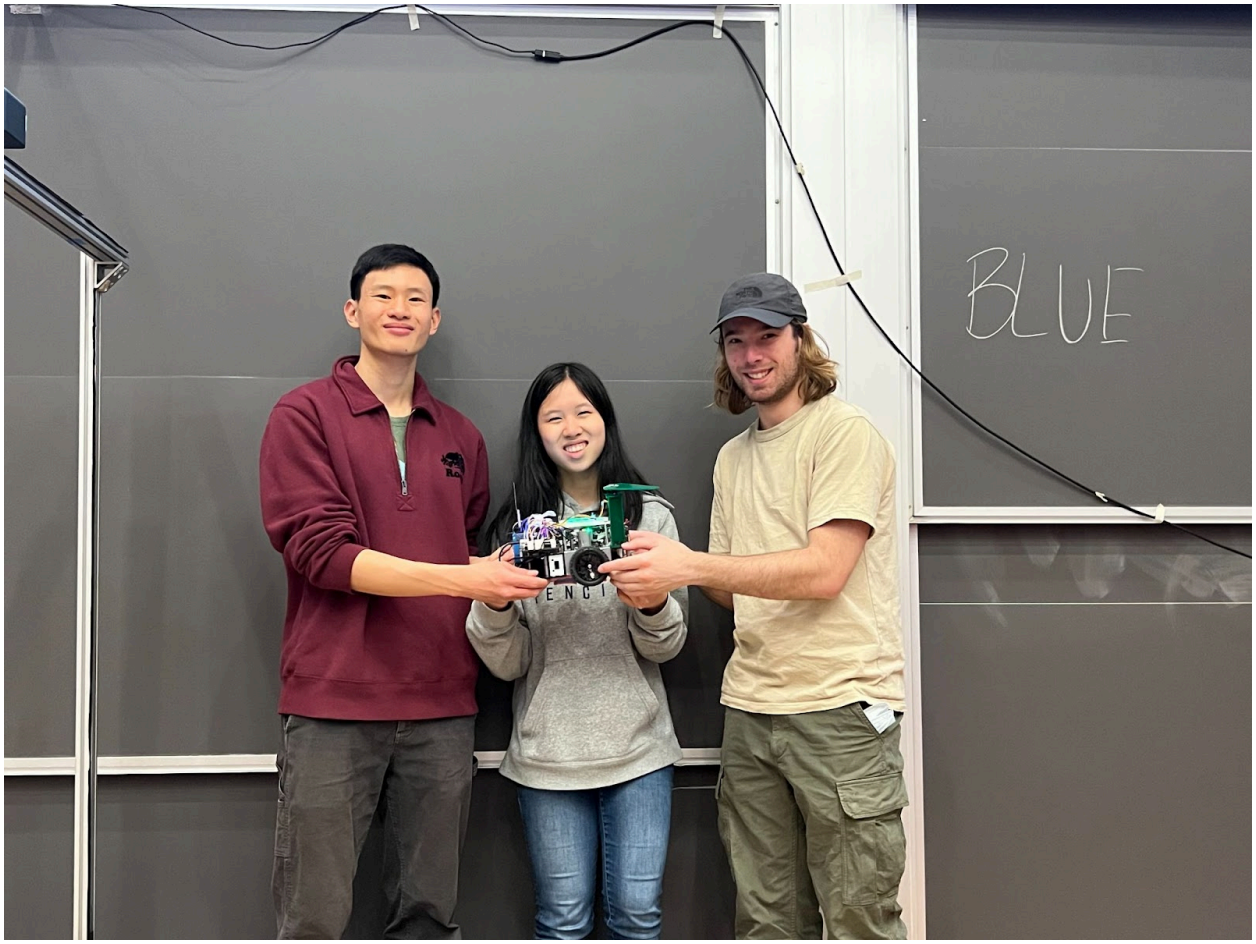


IDK: A Semi-Autonomous Robotic Car for the Robot Battle Arena Game

Tyler Gong, Ian Lamont, Justina Lam

Overall Functionality:

High Level Description of Design and Strategy

For our robot, we designed a 2-layered acrylic mobile base with 2 driven front wheels and 2 roller-ball rear caster wheels. Key capabilities included active-feedback motor control, three Time of Flight (ToF) sensors, tophat integration, as well as a servo-actuated, acrylic, rotating attack arm for striking other robots' whisker switches. Robot control and algorithms were run using an ESP32 S2 microcontroller, and key electronic components include two 12V, high-torque (6.4 kg-cm) DC motors with built-in magnetic encoders, VL53L0X ToF sensors, and a dual-output 12V - 5V battery bank.

We operated under a very simple, overarching mantra when designing our robot for the game: consistent and robust mobility over all. Given the chaos of a multi-robot battle arena, forcing navigation around stationary towers, moving Swinging Arms of Death, and other robots moving in unpredictable manners, we foresaw significant difficulties with developing sophisticated automated tasks that could adapt and avoid these dynamic obstacles. Especially given the inconsistencies in VIVE location tracking, relying on navigating around obstacles seemed unreliable at best, causing us to focus instead on pushing through other robots/obstacles as best as we could so we could reliably pursue whatever gameplay objectives are most advantageous at the time. This resulted in a rather heavy robot (our battery alone was 1.3 lbs), using higher-torque but lower-speed motors with built in encoders, and a low center of mass for maximum stability. This also resulted in us taking on more of an enforcer/support role in the game—instead of quickly flying around the field trying to capture buttons, we focused on defending our nexus and already captured buttons, by pushing other robots off of buttons and hitting their whisker switch with our servo arm.

Approach for Functionality:

For general mobility, our robot employed a differential drive system, driving wheels at the same speed but opposite directions in order to turn, and at the same speed in the same direction in order to drive straight. To ensure consistent motor speeds/straight and predictable mobility, we integrated PI control into our motor control, directly counting encoder counts from the built-in motor encoders and optimizing our control algorithm for the desired number of counts per 50 ms PI control period. This would also allow us to ram forwards into buttons, scoring points.

For wall-following, we used three ToF sensors, one on the front of the robot in the center and two on either side of the wheel on the left side of the robot. By polling the readings from these sensors over I2C communication, we were able to determine both distance from a wall in the front (informing when we should turn), as well as our angle relative to a wall to our left (as determined by the difference in readings between the two side ToF sensors. Therefore, our wall-following took the following general approach: poll side ToFs to determine angle against

the wall and distance from the wall, drive the motors to turn the necessary amount to even out against the wall and/or correct the distance from the wall, and then once even, drive as far as allowed before approaching a wall or having to even out again. Once a wall is detected within a certain range from the front ToF, the robot backs up, turns right 90 degrees, then goes forward a set amount before continuing the wall following algorithm. This algorithm is described in more detail in the software Processor/Code Architecture section.

For our other autonomous tasks (ie. autonomous button pressing), we navigated using a simplified dead-reckoning system using pre-specified rotation rates and durations to run the motors in order to reach specified buttons. We decided not to include VIVE partially due to time constraints, but also due to general concern over the reliability of VIVE measurements; we felt more comfortable spending extra time making sure that our dead reckoning and motor control were very consistent, rather than relying on VIVE outputs that could be shaky. This method rested on the accuracy of our encoders and PI control, as well as the heavy weight of our robot preventing any wheel slipping. While definitely not the most sophisticated, we were able to very simply create additional autonomous tasks, as no additional sensor calibration or testing was required. In the end, we only had time to create and test two routines: one for hitting the far nexus, and one for hitting the central tower and then backing up into the far nexus.

Performance:

Our robot performed very well on both demo day and competition day, scoring 55/50 points needed in the demo and participating in the second-place alliance at the final competition. In large part, this was due to success in our strategy—our robot was heavy, powerful, and had consistent and reliable movement, enabling successful button defence against other players. When we lost in the finals, it was largely due to our lack of speed, which we knew was a downside of our robot. The fishtank robot was also heavy and high-torque, but had a much faster driving speed due to them finding very powerful and fast motors, enabling improved maneuverability allowing them to pin us against the wall and buttons. While disappointing, this result was always seen as a possibility and a risk of our design choices.

Additionally, our autonomous tasks did perform as expected, although much less consistently than we would have liked. On demo day, our robot successfully navigated to and pushed the button of both the far nexus and the central tower, but failed to do both in sequence. This can be explained by a few reasons. Firstly, our robot did end up curving very slightly left when trying to drive straight, which was very unexpected. This actually only started occurring right before demo day so we are still unsure what the reason is, but our best guess is that an internal issues with a motor encoder led to count inconsistencies, since this error started occurring without any software changes. This forced us to add some slight orientation correction maneuvers into our automated tasks, providing additional opportunities for errors to occur in our navigation.

Secondly, our robot also had some difficulties moving a predictable amount over short drive times, like when making slight turns. This could have been avoided by employing

dead-reckoning based on actual encoder counts rather than time, which was something we did try to do—we were forced to abandon it due to time constraints, however. Finally, our robot also would frequently slip off to the side of the button when pressing it, since we were pressing round buttons with a flat acrylic plate. This led to inconsistent orientations once finished pressing the button, causing our task linking multiple button presses together to be especially inconsistent. However, we did manage to run several successful attempts of each routine displaying all intended functionality (see Appendix), including successfully autonomously navigating to the opposing nexus during the actual competition.

Mechanical Design

Overall Description:

Base and Driving

Our robot mechanical design centered around a 2 laser-cut, ⅛ inch acrylic base layers held an inch and a half apart using four sets of two stacked ¾ inch 4-40 standoffs, approximately in each corner of the design. Approximately a third of the way from the front of the base, each layer featured a cutout composed of two stacked rectangles to hold the motors and wheel, such that the wheel only protruded about half an inch from the edge of the base. Motors with our driven wheels attached were inserted into these cutouts by first screwing them onto a motor mount plate, and then press fitting that plate into the base layers as well as struts press fit between the base layers. This fit was secured with duct tape to ensure a robust mounting while still allowing for easy disassembly (we did not attach the mounting plate to the plate using screws, since the screw sockets would have interfered with the turning of the wheels as we did not have very much clearance). Wheels were attached to the motor shaft by screwing directly into the threaded motor shaft, and held away from the motor body using a nut, washer, and 3mm standoff.

Component Mounting

Between our base layers, press-fit ⅛ inch acrylic mounting plates were inserted, used to mount various different electrical components. These included a mount for a horizontally-oriented ToF in the front center of the base, two mounts for vertically-oriented ToFs on either side of the motor on the left side, and plate towards the back of the base for mounting the whisker switch. This plate also contained holes for accessing the charging/discharging ports of the battery, which was kept between the two base layers. Finally, there were additional spaces for ToF mounts on the right side of the motor, though they were never implemented as we did not implement sensing from the right side into our navigation.

Circuit Mounting

All circuitry was kept on top of the top base layer, and arranged so that they remained easily accessible and modular. As discussed in our Electrical Design section, we soldered all circuitry for maximum modularity—these resulted in several, function-specific perf boards (ie.

one for all I2C connections, one for interfacing with the motor driver, and one which had the ESP, in addition to our pre-fabricated motor driver board and top hat circuit board). Especially since all boards were connected using twisted wire molex cabling, circuit organization presented a difficult mechanical design challenge. We settled on the layout portrayed in Figure 2 in the Appendix (see note in the caption), placing the motor driver board and circuit close to the motors and allowing all circuits to remain easily accessible for quick changing of molex connections. These boards were mounted offset from the base using 12 mm M2 standoffs (except for the motor driver interfacing perfboard, which had screw holes that were too small and therefore required using 2-56 screws/nuts to mount), with the tophat board requiring two stacked 12mm standoffs to be higher than other electrical components and not interfere. Molex wire routing was accomplished via a small hole in the top base layer at the front of the base, as well as winding excess wire around standoffs/under perf boards.

Attack Method

Finally, we implemented a servo-actuated attack arm on the front right of the base. A rectangular arm stand was assembled using press-fit ⅛ inch acrylic, and screwed into mounting holes in the base. Critical press fits were reinforced with hot glue and duct tape to ensure a secure connection. The servo was screwed into this stand, and the servo horn was hot glued into a blade-shaped attack arm approximately 7.25 inches long. For attacking buttons, we did not include any additional design features, instead allowing the robot to ram forward into buttons using the front of the base in order to push them.

Performance Analysis:

For the most part, our mechanical design worked extremely well for our demonstration and competition: our battery and sensors were protected, our circuits remained solidly attached to our base, and our robot in general was very robust to bumps and jostling during normal operation. The only points of mechanical concern came from the wheels and the servo arm: firstly, while the wheels were securely attached for all of demo day and most of the matches, they did loosen over the course of all of the seeding rounds, causing one to come loose during our second to last match. This was quickly remedied before the next match by just re-tightening the wheel on the axle, but a potential way to prevent this coming loose would be through using loctite or a laser-cut interface for the motor-axle d-shaft glued to the wheel. Secondly, our servo attack arm did get stuck on another robot and come loose from our robot during our final matchup, despite being reinforced with hot glue and duct tape. Screws would have offered a more secure solution, though our first attempt proved difficult due to the limited space left available on the robot for this attack mechanism.

Iteration Process:

Mechanical design for our robot was an intensely iterative process, facilitated by rapid-prototyping practices. Firstly, several iterations were created in CAD before even beginning manufacturing, as components were repeatedly workshopped to ensure components

would all fit while still allowing the robot to meet the necessary physical requirements/dimensions. This was done through creating rough CAD models of all store-bought components, with dimensions taken from a combination of physical measurements and product documentation.

Next, we moved onto physical prototypes, which evolved as we both tested further and added additional functionality. For example, our first two attempts at building our basic robot for the lab 4.2 mobility test had misaligned press fits and poor tolerances, requiring additional redesigns. Redesigns were also necessary upon adding the ToFs, tophat, and servo actuator arm, which was relatively easy to do though still inconvenient. Thinking through these additional functionalities early would have allowed us to include them in our early prototypes and prevent further redesigns.

Electrical Design

Design Philosophy:

In order to achieve our goal of consistent operation, we decided to build using modular design. This allowed us to test individual components away from known working components so we could easily find lower level bugs before integration. This modular design allowed us to slowly build up features, ensuring by the time it came to integrate them any errors would be from the newly added component. We used this philosophy to build both our physical wiring network and our digital software.

Electrical Design:

In alignment with our modular philosophy, we designed circuitry on multiple different perf-boards based on functionality, allowing for component-specific testing and verification. This manifested in three main circuit boards (one for all I2C connections, one for interfacing with our motor driver, and one for managing all connections to the ESP), connected to each other by several different twisted-wire molex connectors. Each of these subsystems/subboards are highlighted below.

I2C Board: Sensing and Tophat

We controlled I2C communication to three VL53LOX ToF sensors, as well to the tophat, through one clock and data line. This was accomplished through use of the x-shut pins on the ToF sensors, which allow for each ToF to be initialized in order and assigned a custom address for I2C multiplexing. Therefore, our I2C board circuitry featured 13 sets of molex headers, arranged as shown and described in Figure 8. Through this board, ToF and tophat scl/sda lines are connected together and linked to header pins that can easily be connected to the appropriate MCU GPIO pins on the MCU board.

Motor Board and Motor Driver

We decided to drive our motors using the L298N Dual H-Bridge motor driver. We used a line directly from the MCU using LEDC pwm to drive the signals to the driver. The driver uses 12V power to drive the motors and an internal voltage regulator for logic (ground is still connected to common ground with the MCU). The outputs of the driver are sent to the motor board where they are routed to the JST adaptors directly connected to the motors—this enables the motors to actually be driven by the output of the motor driver. Also connected to this motor board are logic and ground voltages from the MCU (fed to the motor encoder leads), as well as routing to two GPIO pins on the MCU. This allows us to extract the encoder A signals from the motor encoder and digitally read them through the MCU, preventing use of quadrature but allowing for easy encoder reading.

MCU

The MicroController Unit we went with was the ESP32-S2 due to its high number of GPIO pins. We were running into limitations with the ports on the ESP32-C3 M5 early on and believed that as we scaled we would need to expand. This turned out to be a good call as the final number of GPIO pins we used was 15, more than the 13 available on the ESP32-C3 M5. The MCU board contains the molex connectors to the other boards. To the I2C board there are power, xshut, and I2C bus connectors. To the motor board and motor driver there are motor driver controller and encoder input/power output connectors. Connected directly to the board is a 5V output for the tophat, a common ground, and pins for the servo. We additionally had a connection for limit switches that was unused in the end.

Power routing

We purchased a power bank that had both 5V and 12V outputs. We directly wired the 5V output to the ESP32-S2 which has an internal regulator to bring it down to 3.3V logic for the rest of the logic system. For the 12V power system we spliced the output barrel jack using 18 AWG wire for direct 12V connections to the tophat and motor driver. We additionally split out an additional ground connection to connect to the ESP32-S2 to ensure a common ground.

Sidebar: Current Draw Estimates:

With motors drawing (at stall) 1.3A each, tophat and main board MCUs drawing around 20 mA each, servo motor drawing (at stall) 650 mA, we estimate an overall current draw of around 3.27 A at absolute maximum. This is slightly above the rated 3A current output from the battery, but since this current draw is split up across the 12V and 5V output jacks (and it's unlikely we'll hit stall simultaneously on all components anyways), we concluded that these were safe operating conditions for our robot. A current limiter would be useful for future iterations.

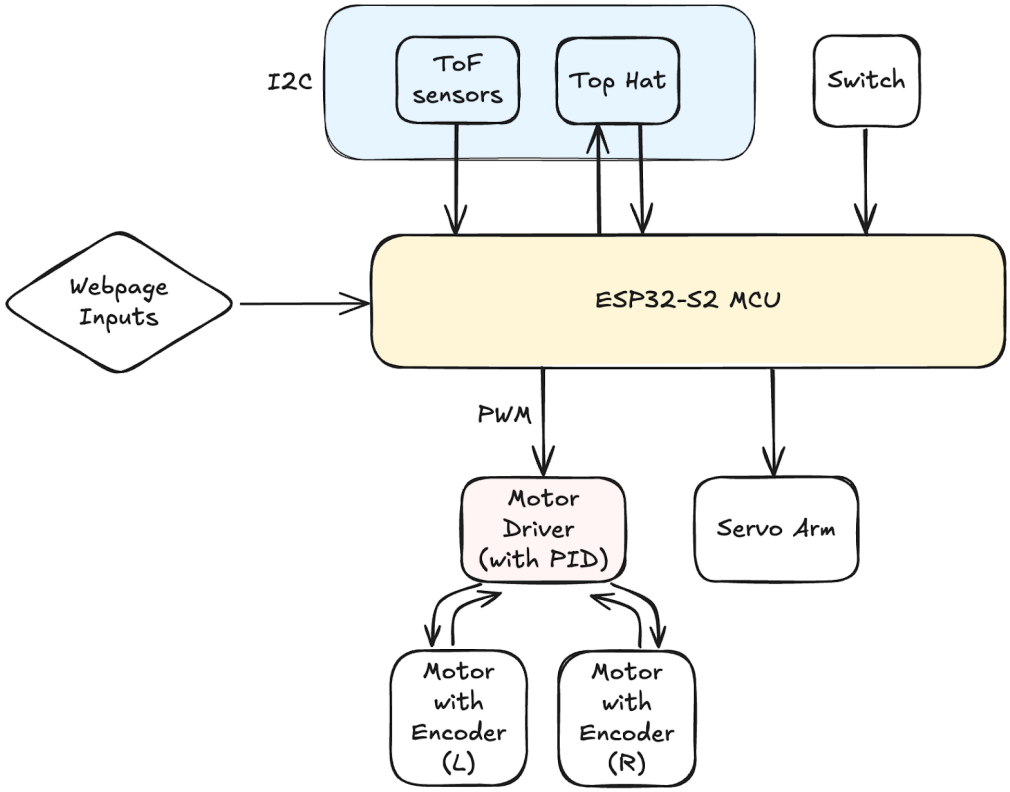
Design Iteration:

The above subsystems took some time to mature. The iteration was mostly non-linear, with a majority of development happening early in the process. As noted before, we had planned to use an ESP32-C3 M5 for our main processor. After testing with our TOF sensors, we realized that we would have barely any extra GPIO left. We decided that we would likely need more than a couple of extra GPIO so we upgraded to the ESP32-S2. Additionally we knew early on that we would likely need to upgrade our motor drives from the ones in the GM lab like the SN754410NE. We had heard that this was a limited driver that had given teams trouble in the past, and we had experimentally validated a significant voltage drop across the H-bridge with a multimeter. Additionally, we saw that there was a current limit of around 1A for the SN754410NE, which could theoretically be passed by our motors which have a stall current of 1.3 A. We chose to upgrade to the L298N breakout which is a hobby staple and known for its reliability and higher voltage capabilities. After these two upgrades, we kept all of our main

components the same. All other improvements we made were in our handling through software and not upgrading hardware.

Processor architecture and code architecture

Block Diagram of Logical Connections:



Processor Architecture:

In order to simplify communication and code complexity we decided to go with a one processor architecture. We had been warned that we already had many I2C devices on our bus and adding another processor that would likely be communicating frequently would put strain on the bus. Additionally we never had any issues with computation that another processor would benefit from so we decided on reducing the complexity. Our processor constantly ran through the main loop in under a millisecond and we artificially inserted delay to get it to 20ms.

Code Architecture:

At the code level, the software was compartmentalized into separate files for each task. The main file, *webpage-tof.ino*, included the setup and loop functions. Setup includes initialization processes for the ToFs, motors, switch, and servo, as well as the webpage, which contains buttons to start specified actions via WiFi commands. Keyboard inputs were also added to the webpage, allowing the user to use the WASD or arrow keys to send signals for forward and backward movement, as well as turning and stopping; Q, E, and R were mapped to routines for attack arm servo rotation and autonomous drive tasks.

The servo, connected to the attack arm, is controlled by the servo module, initially in the stopped state. Pressing a button on the webpage triggers an event handler that starts the back and

forth rotation of the servo through a 180º range of motion, and another button stops the servo at its current position.

For tasks involving manual control of the robot, code for simple drive instructions is kept in a globally accessible module, for setting the direction of rotation of each wheel. These were used in the main drive module, which included functionalities of simple manual driving as well as movement with respect to the wall. At any given time, the robot would be in one of eight different states: moving forward, turning, evening out to become parallel with the wall, moving away from the wall, pulling towards the wall, manual drive, switch reading, or following a preset routine. The current state of the robot was maintained as a global variable and checked in the main loop on every iteration.

Many of the drive states were designed around the wall following task, to achieve specific types of movement relative to the wall. During this task, the robot rotates through the various states, repeatedly sensing its position relative to the side and front wall and driving accordingly. At a high level, the robot regularly checks its angle with respect to the wall as well as its distance from its front and left sides to the wall, and according to the result of these ToF readings, chooses an action: moving forward, rotating to become parallel with the wall, or turning right at a corner.

Wall Following Procedure:

Specifically, during wall following, we start in the GOFORWARD state, which reads from the front ToF sensor and moves the robot forward, either until it reaches the detected wall or for a specified maximum number of milliseconds. At the end of this phase, if it has reached a front wall, the robot will TURNRIGHT, turning 90º at the detected corner. In either case, we then transition to the EVENOUT state, in which it queries the distance from the wall to each of the two ToF sensors on the left side of the robot. First, the difference between these distances allows us to calculate the angle between the wall and the current direction of the robot; for example, if the difference between these readings is equal to the distance between the sensors, then the robot must be at a 45º angle towards the wall, and should turn right to even out. We then calculate the amount of rotation needed in the opposite direction until the robot becomes parallel with the wall, and perform this movement. Finally, the EVENOUT state begins once again, and if the robot is confirmed to be even with the wall, we transition back to the GOFORWARD state.

However, while this ensures that the robot remains parallel to the left wall, during this process the robot will occasionally drift too far from the wall, or too close to it and get stuck. Therefore, if during the EVENOUT state, both ToF sensors on the left side show readings over a specified threshold, we enter a HUGWALL state, in which the robot moves to the left for a specified duration, pulling it closer to the wall. On the other hand, if during the EVENOUT state, both ToF sensors read under a specified threshold, we enter the FLEEWALL state, moving to the right for a specified duration. The combination of these ensures that the robot is always within a given margin of distance from the left wall, and both states complete by transitioning to the EVENOUT state.

Autonomous Button-Finding

Due to the lack of a working VIVE, our group based all autonomous navigation to buttons on dead-reckoning, driving the motors at a specified PWM for a specified duration of time to consistently get to predictable locations. To actually code these autonomous tasks, we implemented live-troubleshooting feedback to our website, outputting both the specific command as well as the duration it was given for to the user while in manual mode. This allowed us to manually drive the robot along the desired path, marking down the commands and time durations, which we then hard-coded into specific subroutines to retrace these paths. Through this method, we created two subroutines: one for navigating to and hitting the far nexus (Video 1), and one for navigating and attacking the central tower before backing up and hitting the far nexus (Video 2).

Iteration Process:

The software iteration process followed the incremental addition of tasks; it began as a single consolidated file and, after realizing that this would soon become messy and difficult to structure, it was modularized early on to separate functionalities into different files. Header files were added to define pins, global variables, and pre-set parameters such as those determined by trial and error, so that these could be quickly modified after observation of performance. As mentioned earlier, much of the drive functionality was built around the wall following task, and later adapted for manual drive and autonomous routines.

For autonomous driving tasks, we initially tried to read from ToF sensors on loop to continuously inform movements, but after discovering the significant latency added by repeated ToF reads, we adjusted our design to minimize this delay by alternating between motor movement and periodic sensor reading, moving as far as safely possible on the information gained from each measurement. Having initially planned to use 4 or more ToF sensors around the perimeter of the robot for the game and wall following task, we reduced it to the minimum-necessary pattern of 3 sensors along the left and front sides, for easier integration. To speed up development and testing, functions were effectively parameterized by global constants for optimal thresholds, durations, and distances to move under each state and condition, determined by a pattern of trial and error reminiscent of binary search. Autonomous routines to attack a tower or nexus were developed in a similar way, by using dead reckoning and manually defined timing parameters to achieve a sequence of movements to reach each button. We did try to redefine this system of dead-reckoning to base movement off of the physical encoder counts instead of for a set period of time, but we quickly ran into bugs and out of time, forcing us to shelve the idea.

Retrospective

Tyler Gong

Very sincerely, I thoroughly enjoyed this class. I had had some mechatronics experience previously, but through the lecture content and guided activities, I gained experience with so many more sophisticated topics in the field, while still keeping a grasp of their physical applications. Perhaps most important, however, was the practical debugging and project-development skills I learned. With regular, tough deadlines and relatively unguided labs, I improved significantly in my ability to drive my own learning, and put together these systems myself. These were no doubt tough, but I believe that added to the experience immensely; by the time the final project came around, I knew that things weren't going to work the first try, but also that I was fully capable of finding whatever the issue was and fixing it. This was especially true with a lot of the more complicated software tasks, as I struggled most with this area since I had the least experience there.

For future classes, I only have a few improvements I can offer. Firstly, it would be very helpful if GM lab had more general electronics and hardware in stock—specific items like 2-56 standoffs, more varieties in color for stranded vs. solid-core wire, and other connectors besides molex (ie. JST) would be very helpful and prevent time spent scrounging up materials/walking back and forth from Detkin. Additionally, better soldering irons would be very helpful, as a lot of connectivity issues we ran into were due to poor irons (we were also missing several sets of helping hands by the end of the class). In terms of gameplay, I thought it was overall really well designed—the only thing I would recommend is a steeper penalty for WiFi commands to prioritize autonomous commands more, as well as a higher damage rate from damaging a nexus directly.

Ian Lamont

While this class seemed like an extreme amount of work, especially at the end, the overall experience was extremely rewarding. Seeing different system you created combine into one product and work makes it worth the hard work. I'm extremely proud of the work our team made in executing this project and in many ways I wish there was more time to continue expanding on our robot and adding even more features. By far this project was the most enjoyable part of the class even if it was the most difficult. I think this was partially due to the team aspect and the camaraderie from working through difficulties together. I think this actually lends itself to better learning as sometimes during the other labs you felt very alone in struggling against problems. I think this was one of the worst aspects of the class was the times when you were stuck on a problem working on a lab alone because it felt like the hours you needed to put into each of the labs was much much greater than the times TAs were available. If labs were done with a partner, it would be possible to bounce ideas off each other and work through problems together. I totally understand why this isn't the case already as it does lead to significant learning, but this might help some issues with workload. Beyond this one small issue, I really enjoyed the rest of the class. All the course content was really interesting and while I knew some little things about electronics individually, I learned much more about them in depth

and how they combine to create a system. I also want to add that I really enjoyed the game. It was so much fun to compete and show off the final product with the rest of the class. Seeing what everyone had been working on come together was really cool and the competitive aspect of it was really fun too. I think one cool change would be to add a second field. I'm not quite sure what this would look like but it would make autonomous behavior more dependent on sensing and not just knowledge of the field. Maybe this would make it too difficult so extra time would be needed to test on both fields. Overall this class was extremely enjoyable, even if it required so much work. Huge thank you to the entire TA team as well as Professor Weakly for making this class what it is.

Justina Lam:

Throughout this project and class, the most valuable learning involved new experience with hardware and integration of electrical components, and learning to adapt and absorb information when surrounded by the unknown. These practical skills and tangible applications were also the best parts of the class, and the ones where I came in with no background or grasp of "common sense" regarding electrical components. This made it infinitely more exciting to progress from googling what a PWM is to writing code to control multiple motors, or from a theoretical understanding of how a given circuit works to building and experimentally adjusting it on a breadboard to achieve a desired effect; or from spending an hour figuring out how to rotate a view in Solidworks to creating an entire CAD assembly of an excavator with press-fit precision. The most difficult, unfamiliar, and time consuming tasks are now the ones I am most proud of. And while I've grown accustomed to the all-nighter experience for a variety of projects and types of material to study, the early morning hours spent on this course's assignments have been some of the most enjoyable and rewarding, even outshining some of my favorite courses in my actual major.

The final project game was very exciting and well designed as it lends itself to a wide range of creative options, but is constrained enough to have clear tasks and expected behaviors. The only suggestions might be including in check-in any expected behaviors that might affect fair gameplay, or adding limits on weight, expense, etc. so that teams can gain more advantage through design and strategy than through component ordering choices, although this is an important aspect for industry preparedness so may be intentional. As for the class itself, sparser TA office hours and some partner assignments would have been helpful; many late days were taken while waiting for the next opportunity for a TA check in, and the most discouraging part of the class was often being lost on how to approach a task and spending days experimenting, scanning documentation, and debugging alone, when working with a partner might have saved much of that time on both sides.

Some of the most important types of intuition I've gained from this class are estimating the time needed to achieve a desired mechanical or electrical result, the complexity of integrating electrical components, the likelihood that unexpected results are due to connection issues, and the realistic timeline of going from a CAD draft to a functional prototype. Some of the most important and applicable new skills include communicating with experts in an area where I have

no experience and vice versa, soldering and Molex connections, the concept of pull-up resistors and distrusting shared grounds between differing power supplies, CAD, and dealing with noise. This class has left me with several memorable takeaways. First, while hardware in practice is often much more unpredictable and unexplainable than software, it is not as foreign, mysterious, or terrifying as I had made it out to be and shares much of the same logic as coding; and second, mechatronics is a really fascinating and enjoyable field, and I have this class to thank for a new passion that will undoubtedly evolve into a very expensive and very satisfying hobby, or potentially career.

Appendix

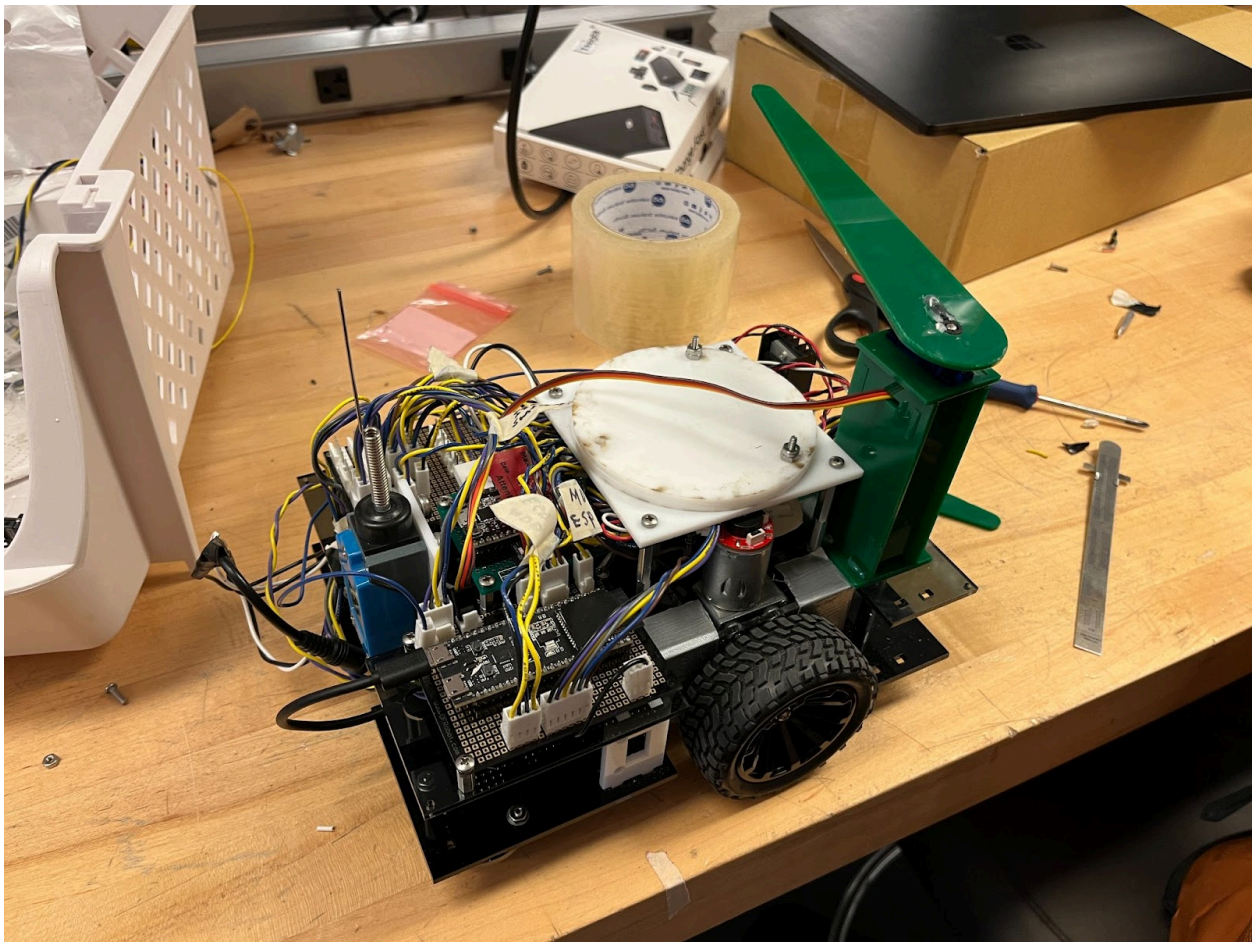


Figure 1: Nice photo of our robot.

Item	Description	Quantity	Unit Price ($)	Total Price ($)	Purchasing Link	Datasheet
Battery	20000 mAh, 12V/5V Li-Ion Power Bank	1	29.13 (on sale)	29.13	Link	Amazon description
ToFs	I2c chip, VL53LOX (4 pack)	1	12.90	12.90	Link	Breakout description on Amazon, chip datasheet

Caster Wheels	Caster Ball Wheels (4 pack)	1	8.90	8.90	Link	Technical description
Motors	12V 100 RPM Motor with Encoder	2	16.98	33.96	Link	Technical description
Driven Wheels	RC Car Wheel, 4 mm Axle Hole Diameter (4 pack)	1	14.97	14.97	Link	Technical description
Motor Driver	L298N Dual Motor Drivers (2 pack)	1	6.99	6.99	Link	L298N datasheet, equivalent breakout datasheet
Microcontroller	ESP32 S2	1	7	7	From TA staff	datasheet
Servo Motor	Tower Pro SG90 Servo	1	Free	Free	From Detkin	Equivalent datasheet

Total price: $113.85.

Table 1: Bill of Materials for our robot. Note: fasteners and acrylic not included, but acquired for free from GM lab.

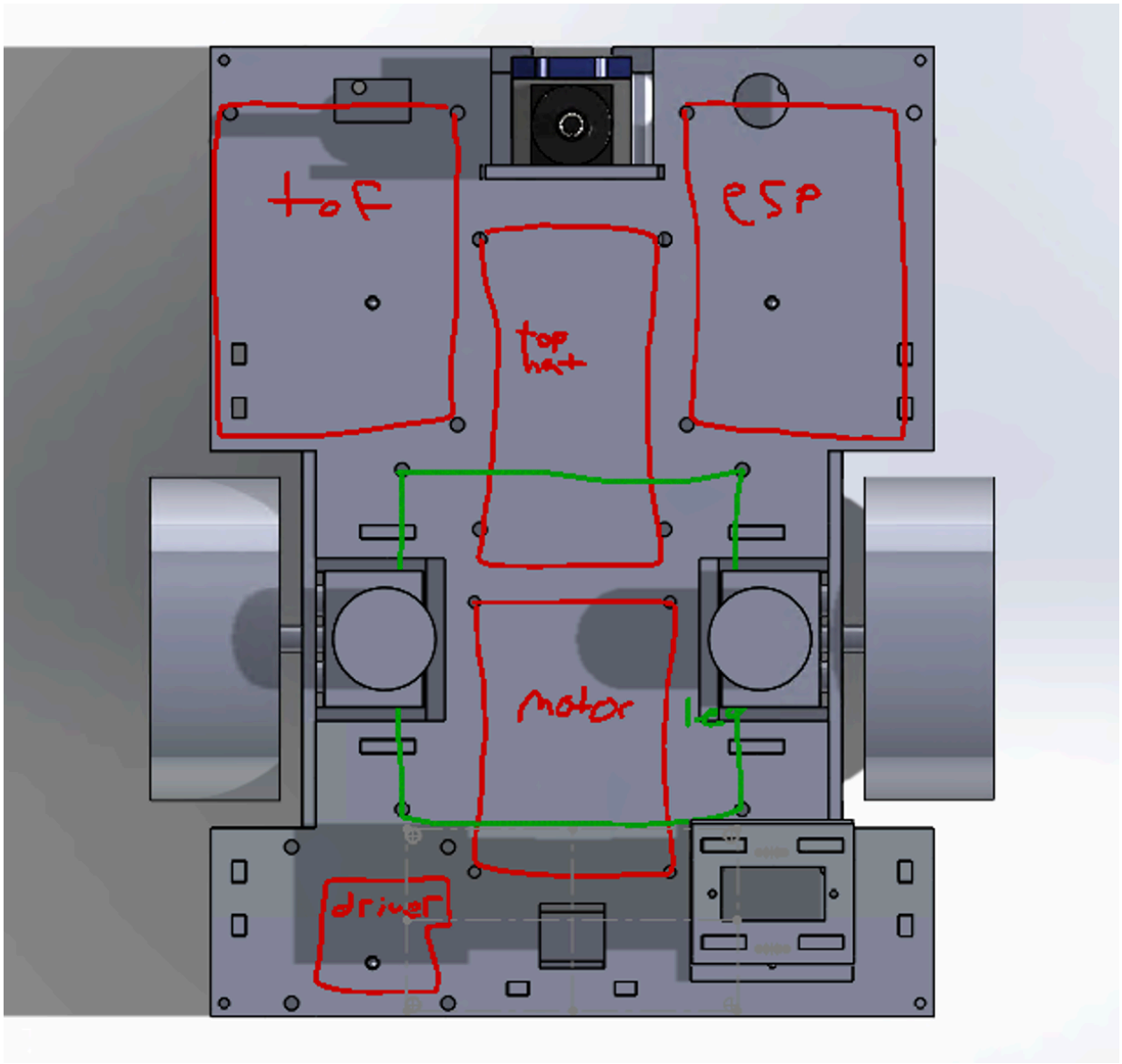


Figure 2: Electronics organization schematic, mechanical design. Note: this schematic ended up being mirrored horizontally compared to our actual design, but all organization remained functionally the same.

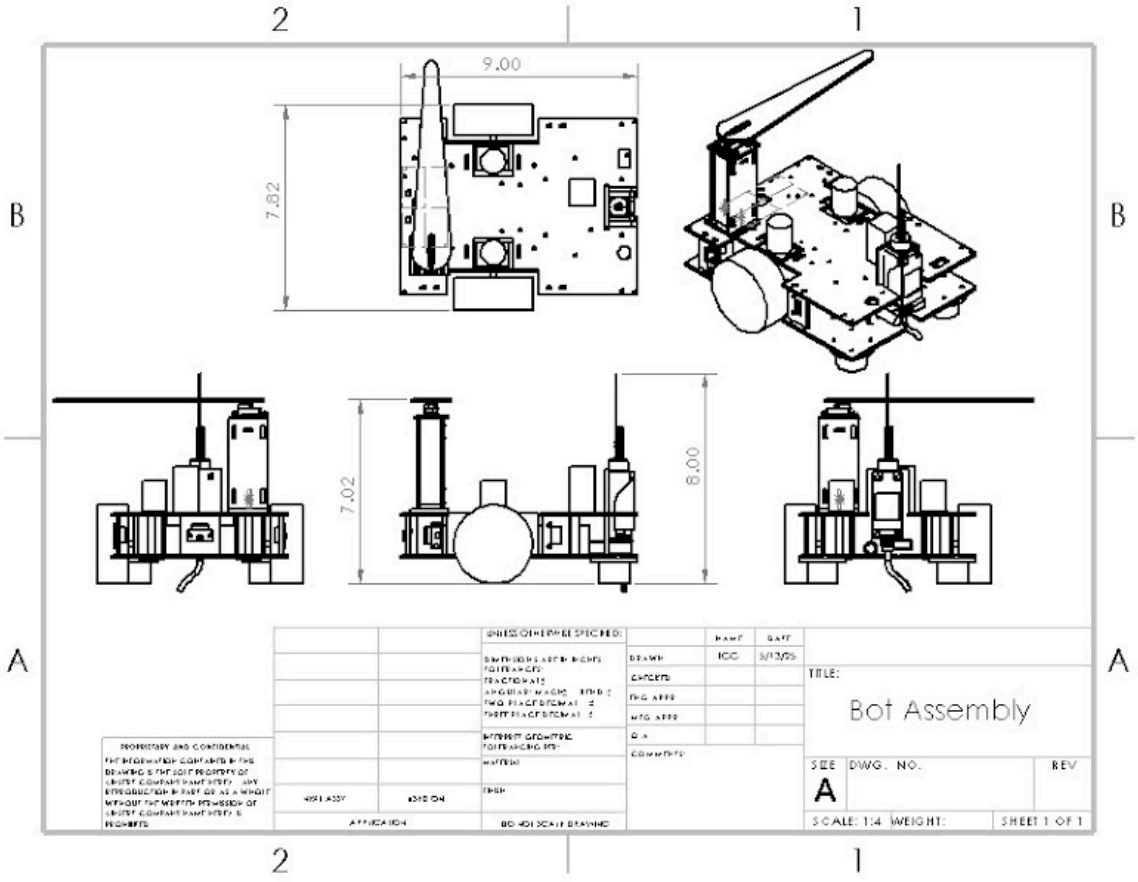


Figure 3: CAD Drawing of Bot Assembly. Drawings of selected individual components are featured below.

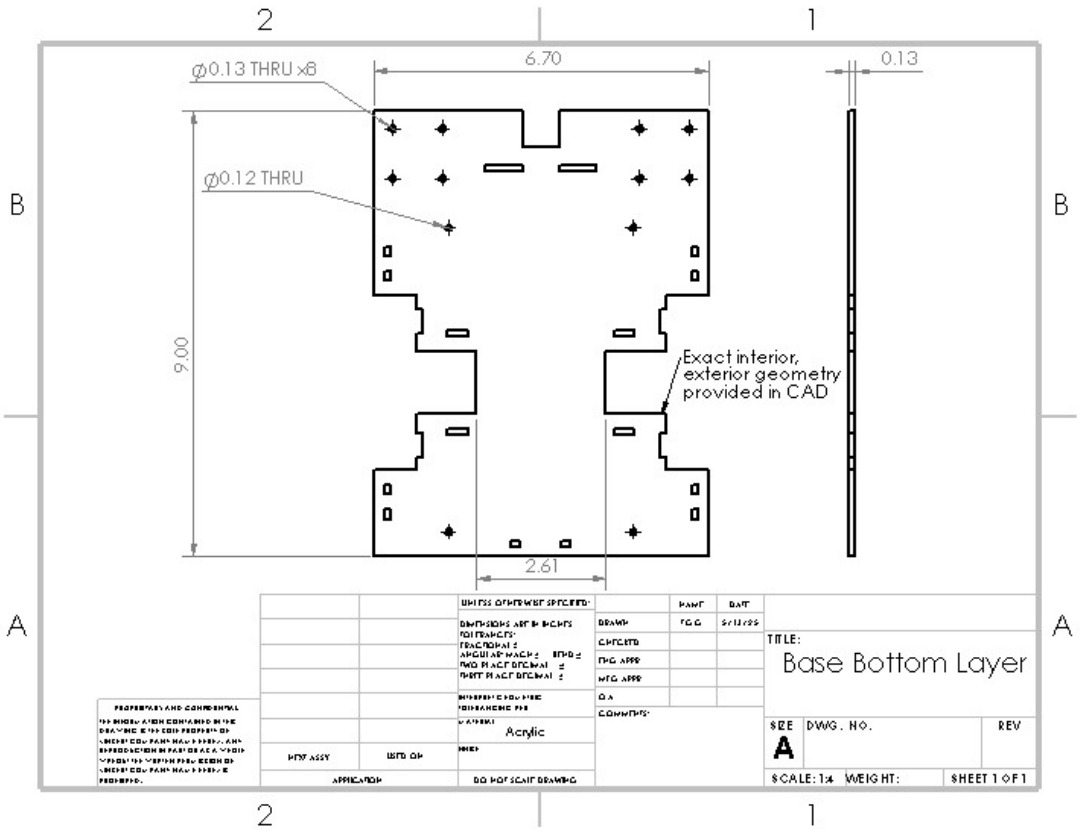


Figure 4: Base Bottom Layer drawing

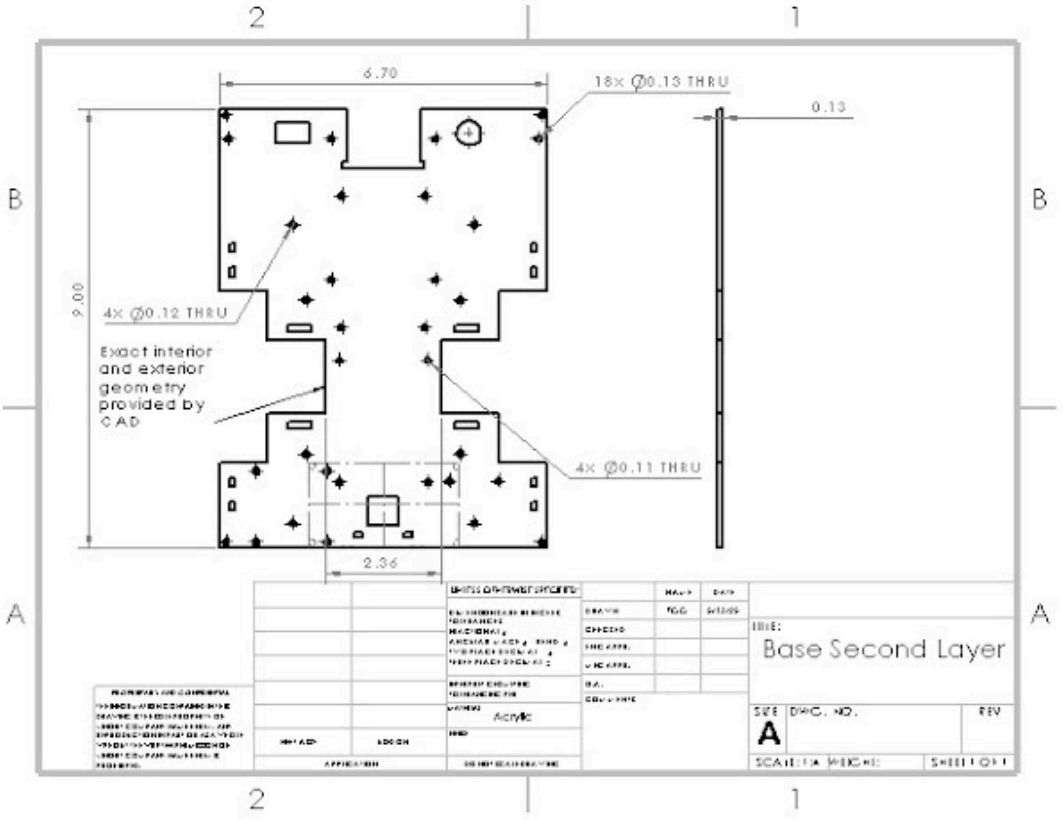


Figure 5: Base Second layer drawing

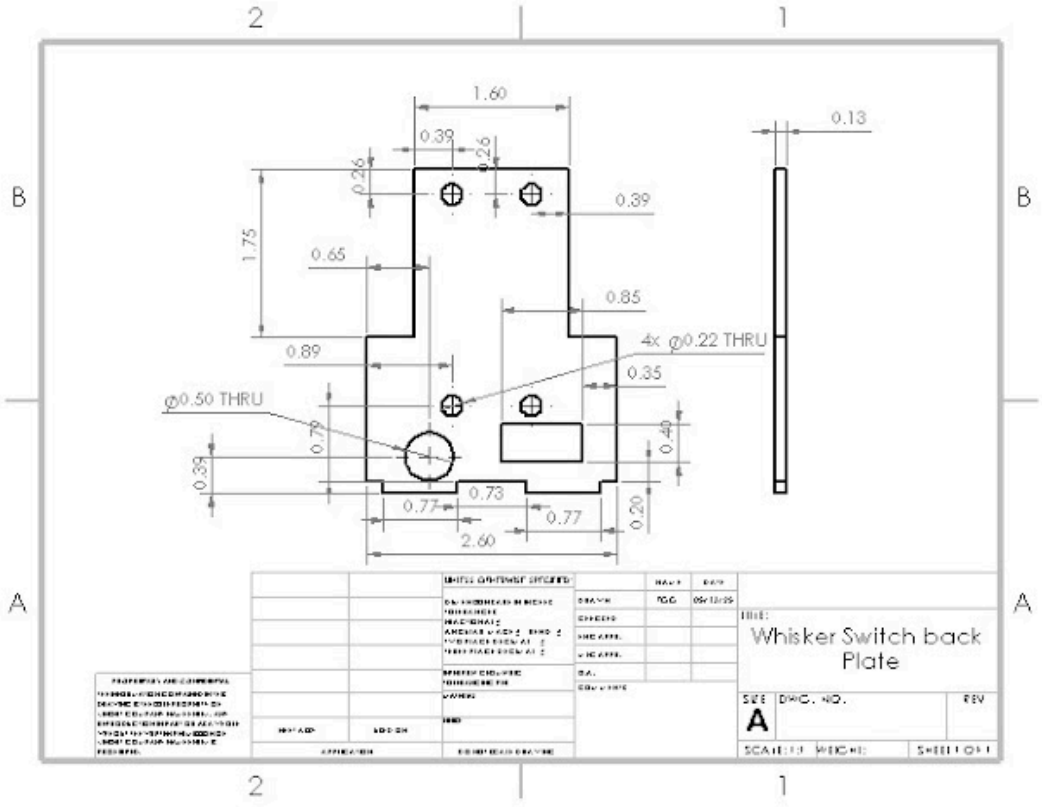


Figure 6: Whisker Switch Back Plate drawing.

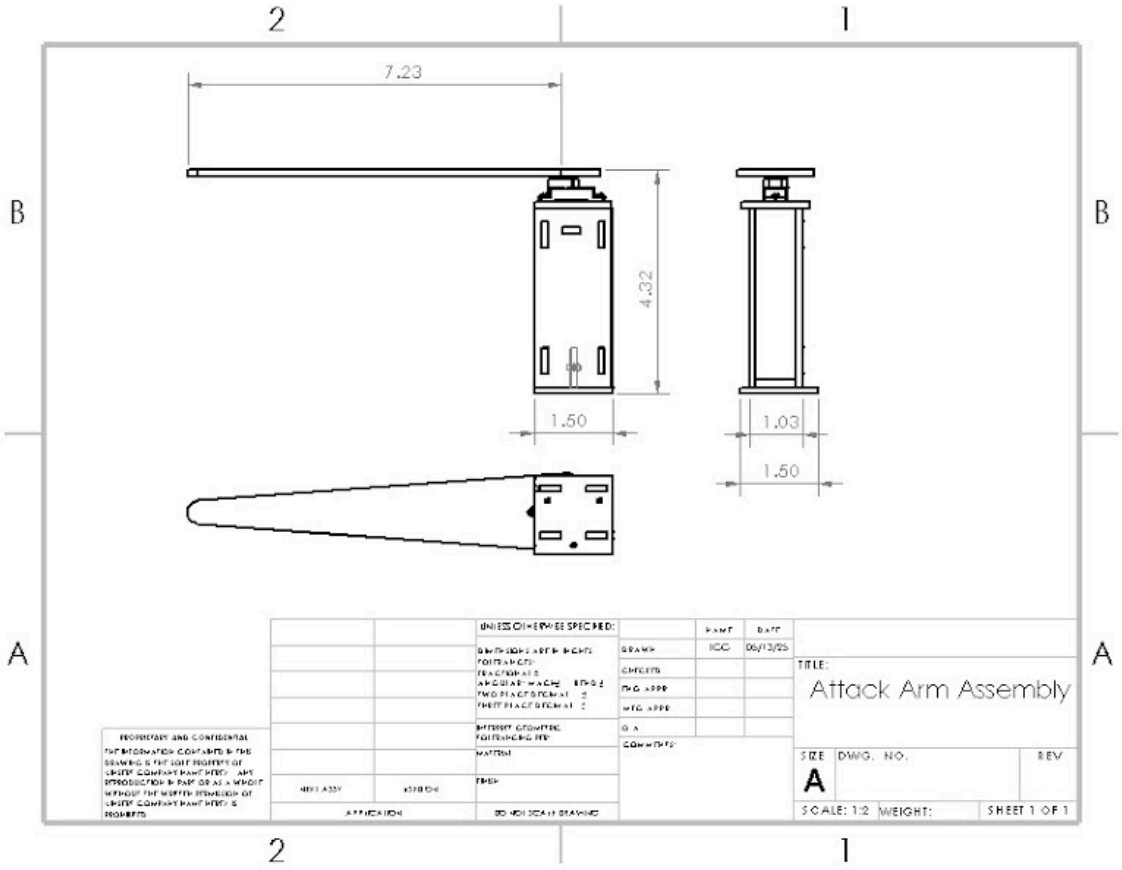


Figure 7: Attack Arm subassembly drawing.

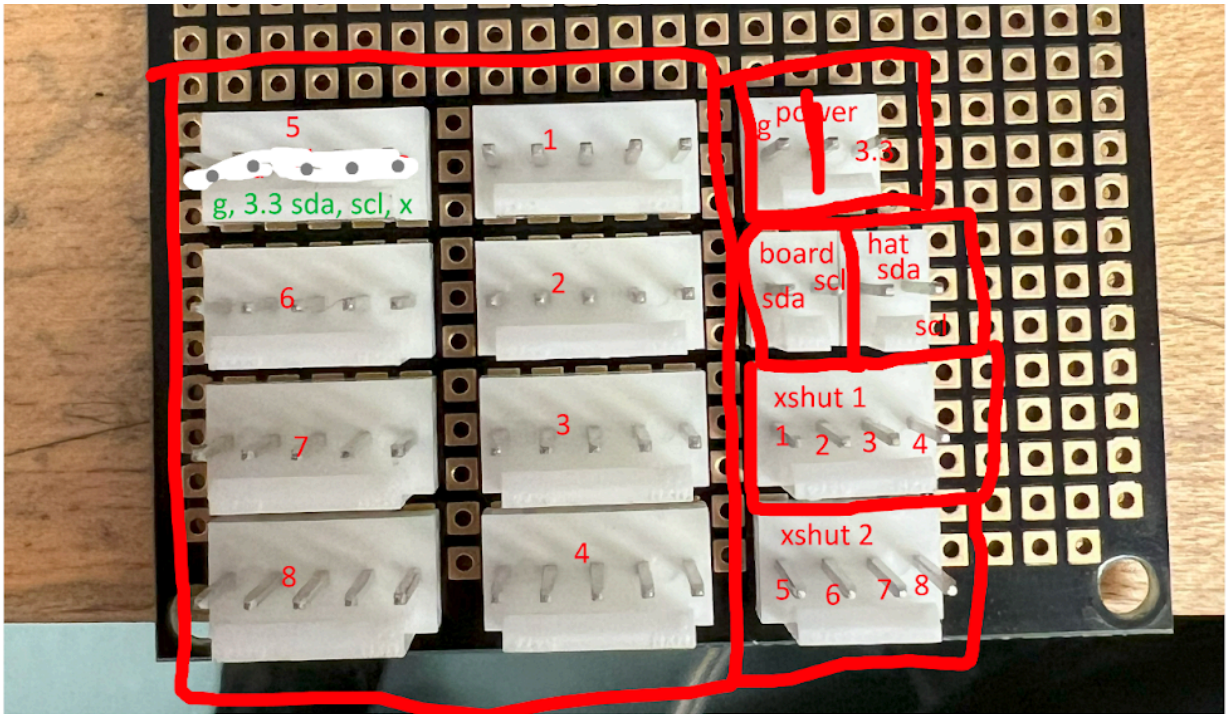


Figure 8: Connection Mapping for I2C board. Main block on the left are plugs for up to 8 ToF sensors. Xshut headers plug to GPIO pins on the MCU, along with board sda, scl pins. Hat sda, scl header pins plug to the board sda, scl pins, and serve as the connection point to the top hat circuit board. Power and ground headers plug to 3.3V, Gnd pins on the MCU.

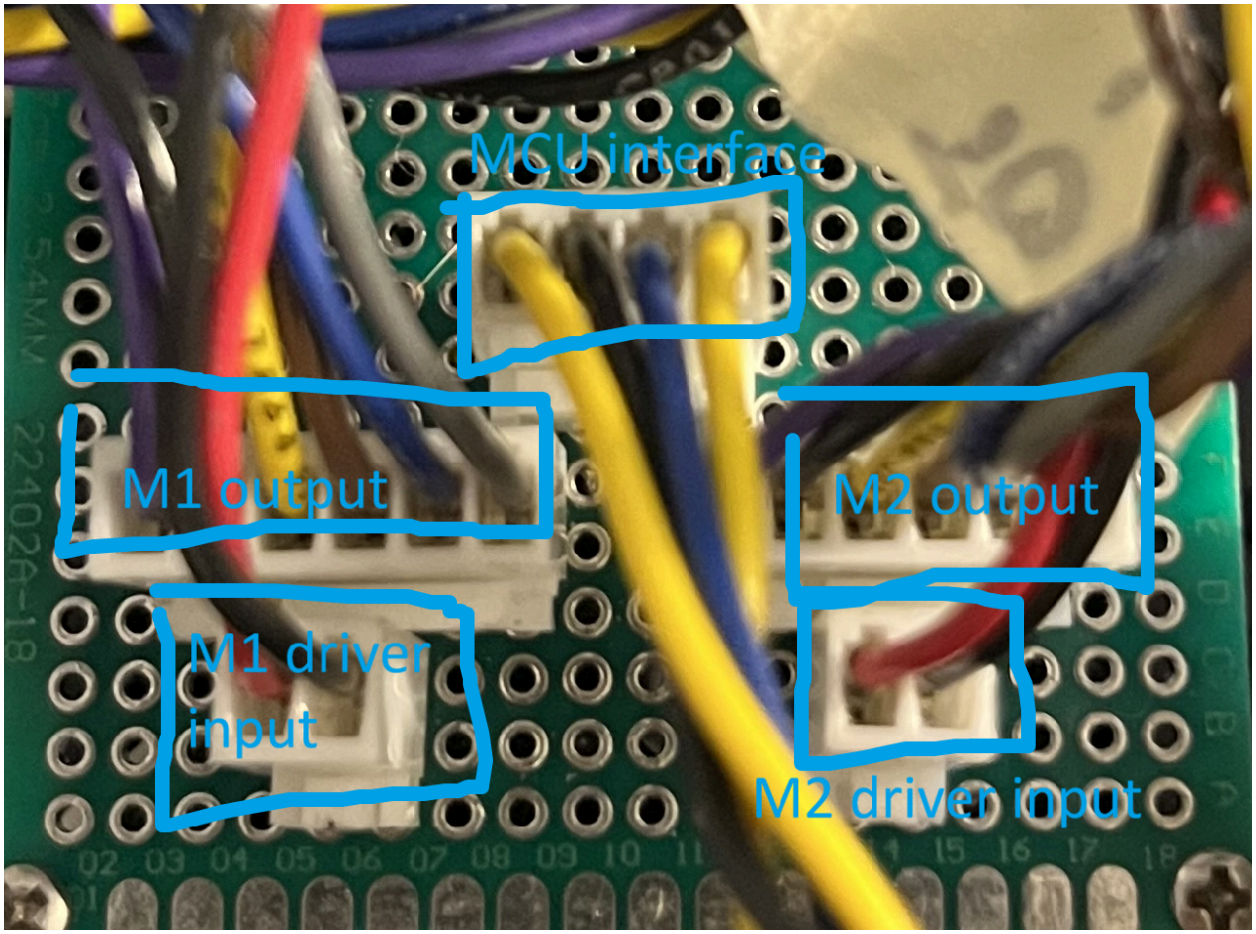


Figure 9: Connection Mapping for Motor board. M1, M2 output plug directly to the motors, interfacing with the drive circuits and encoders. MCU interface block connects to the MCU, providing 3.3V power and ground to the motor encoders and sending encoder input signals to GPIO pins on the MCU. M1 and M2 drive input blocks connect from the motor driver, and ultimately dictate the duty cycle and direction of each motor.

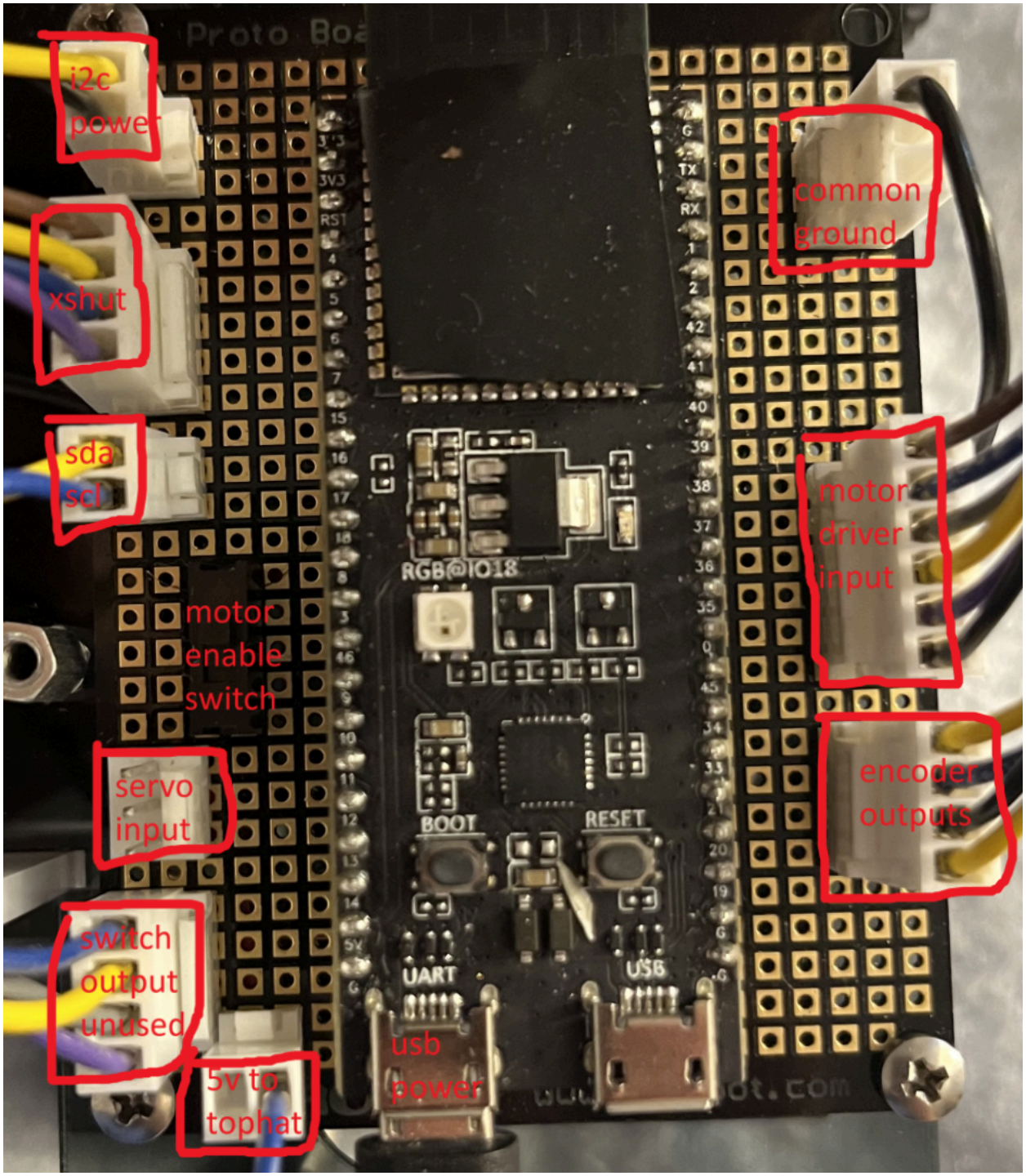


Figure 10: Connection mapping for motor board. ESP32 is held in female header pins soldered into the perfboard, allowing for easy soldering. Perfboard was soldered such that each header pin in each block connects to the appropriate GPIO or Gnd/3.3V/5V pin from the MCU. Stripped, solidcore wire was used to create a 3.3V, 5V, and Gnd rail stemming from the MCU. Note: common ground connects battery bank ground and motor driver ground to MCU ground.

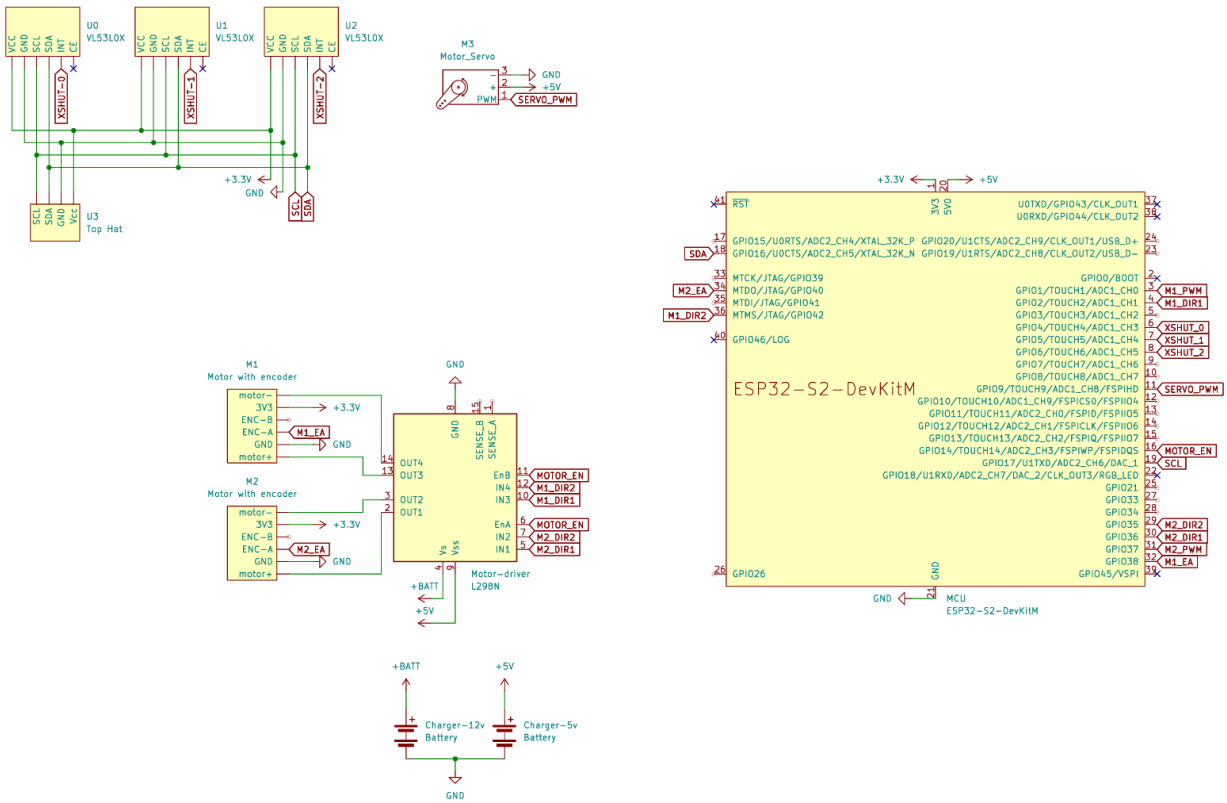


Figure 11: Final schematic for electrical design.