Team 17: Griffin Addison, Gabriel Bennett, Jonathan Lee

Prof. Mark Yim

MEAM5100: Design of Mechatronic Systems

22 December 2022

Final Report

Functionality

Our strategy for winning the game is to prioritize the tasks in order as follows: ensuring the trophies are on our side, ensuring the fake's are on the enemies side, and then ensuring the police car is on the enemy side of the field. This will be done by, at game start, moving halfway down the field from our starting corner and then scanning the field and logging the positions of every item. After the items are logged, we check through the priority list to see if any item is not where it belongs. Then we will retrieve this item, and then move back to the middle of the field and scan again. This will continue until all items are where they belong. If they all are, then we will attempt to push the police car as deep into enemy territory as possible, before returning to center field. Once centered, our bot will scan until some item's position is disrupted. We planned to do all of this autonomously.

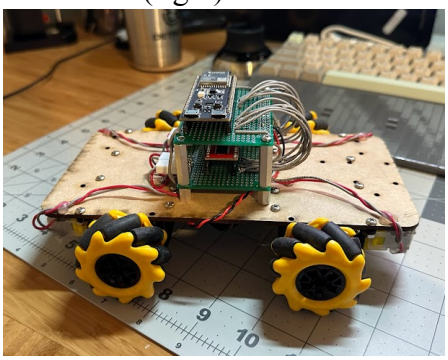
With this idea in mind, we decided to first choose wheels that had the most flexibility while not sacrificing any grip. This is why we went with holonomic wheels. Next, the motors we chose were stronger than most so that we would have more power behind pushing the police car. Then, we chose to separate out the power supplies and use two different microcontrollers for faster and more efficient resource allocation. We built a claw for the front with a strong servo motor to ensure that when we retrieved a trophy, it would not drop it, nor would it be removed without our desire. We decided to build our bot with a multilevel design to reduce our overall footprint and better organize our boards. We also decided on using vive sensors, ir sensors, and TOF sensors to more broadly sense the arena, as well as the contents within.

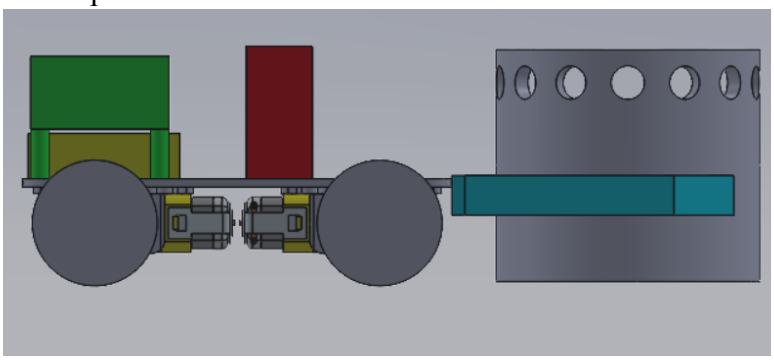
The IR circuits worked well, and so did the vive circuit. Unfortunately, Gabe detected an error with the TOF sensors that led to less than optimal performance. It was determined that the TOF operates at a frequency of 30 Hz, and the 23 Hz trophy, being somewhat close in frequency, was effectively blinding the TOF and causing the bot to malfunction. This was corrected by reorienting one of the TOF sensors.

Mechanical Design

To create a mechanical system we first identified what behaviors require mechanical interaction, then decided what sorts of mechanical solutions would be used to handle each behavior, and finally designed each of the mechanical subsystems and integrated them. First, the robot behaviors we identified that would require mechanical interaction were moving across the playfield between tasks, being able to move the police car 1 foot, being able to rotate IR and ToF

sensors for beacon sensing and wall following, and grasping beacons such that they can be moved. The first three behaviors, mobility, pushing the police car, and rotating sensors, we decided could be handled all with a highly mobile platform. We toyed with the idea of mounting the IR and ToF sensors on a servo, but decided that the added mechatronic and time complexity of such a solution would not be outweighed by the relatively unimportant sensor precision benefits, at least within the context of performance expected in this competition. As such, we decided that these three behaviors would be handled by reusing our holonomic mecanum drive from lab 4, which allowed precise-enough translation in X and Y, rotation, and any linear combination of the 3. The design of this base was functionally identical to that of lab 4: four mecanum wheels situated roughly at 4 corners of a square, each driven independently by a TT motor. The only difference about the final base versus the lab 4 base was the reorientation of the motors so that the motors protruded inwards towards the center of the base as opposed to outwards forwards and backwards of the wheels. This was necessary in order to fit the relatively large claw design in front of the mobile base while staying within the 12” cube bounding box restriction. The pictures below show a photo of lab 4 mobile base (left) compared to final lab full block CAD (right) and how the motors are pointed inwards in the latter.

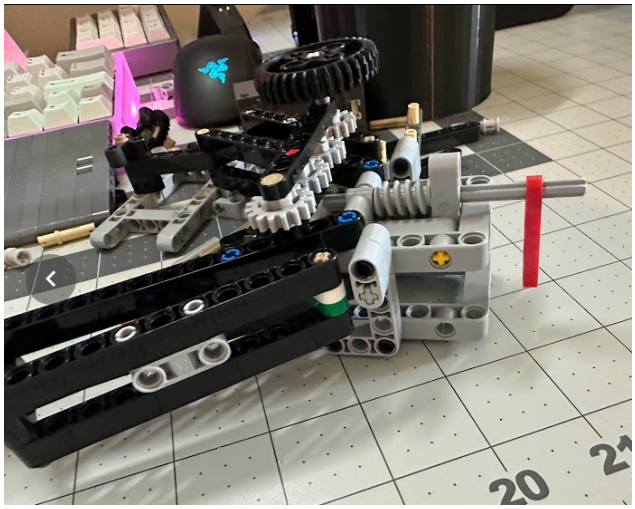




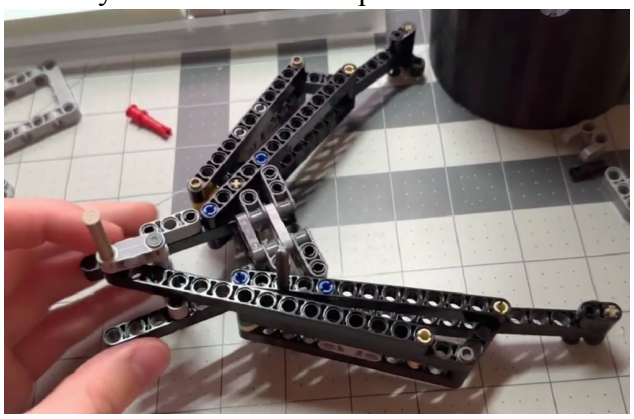
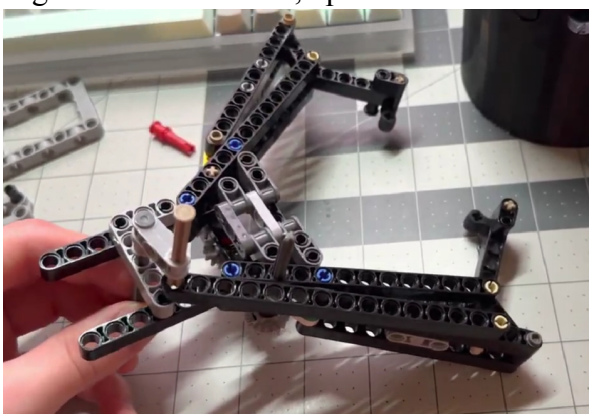
This mobile base architecture fulfilled 3 of its 4 goals with almost no problem: moving across the playfield, moving beacons, and rotating for IR and ToF sensors thanks to the independent motor control and the superior degrees of freedom of a holonomic drive. The robot had a little bit of trouble pushing the police car, as the police car was somewhat heavy and there was quite a bit of friction with the field floor. The base definitely had enough power to push the base, with 4 well-powered TT motors all working together. The issue was more a combination of mecanum wheels having slightly inferior forward grip to traditional rubber/grippy plastic tires, as well as not having enough weight to give the robot enough grip. This was minor, however, and the robot still had enough grip to push the police car, albeit slowly.

The other primary aspect of mechanical design was the claw design. We wanted to create an active claw design that actively gripped the beacons to give us full control of the beacons once gripped. In terms of literal mechanical design, this gave us two options: one, to design something out of 3D printed and laser cut parts, or use a hybrid of legos and 3D printed and laser cut parts. Given that the design freedom allowed by legos had enough resolution to produce a claw with sufficient performance, and with less tedious work needed to correctly size holes for pressfits and for freely rotating fits with the inconsistent kerf of RPL’s laser cutters and tolerancing of the 3D printers, using legos for some of the dynamical parts allowed a more complicated design to be realized. The claw went through a few iterations, one that used a worm gear and DC motor to provide slow, torquey, non-backdrivable claw clamping, and one that used a strong non-Lego servo with a custom 3D-printed servo-horn to clamp. The earlier design even converted the DC

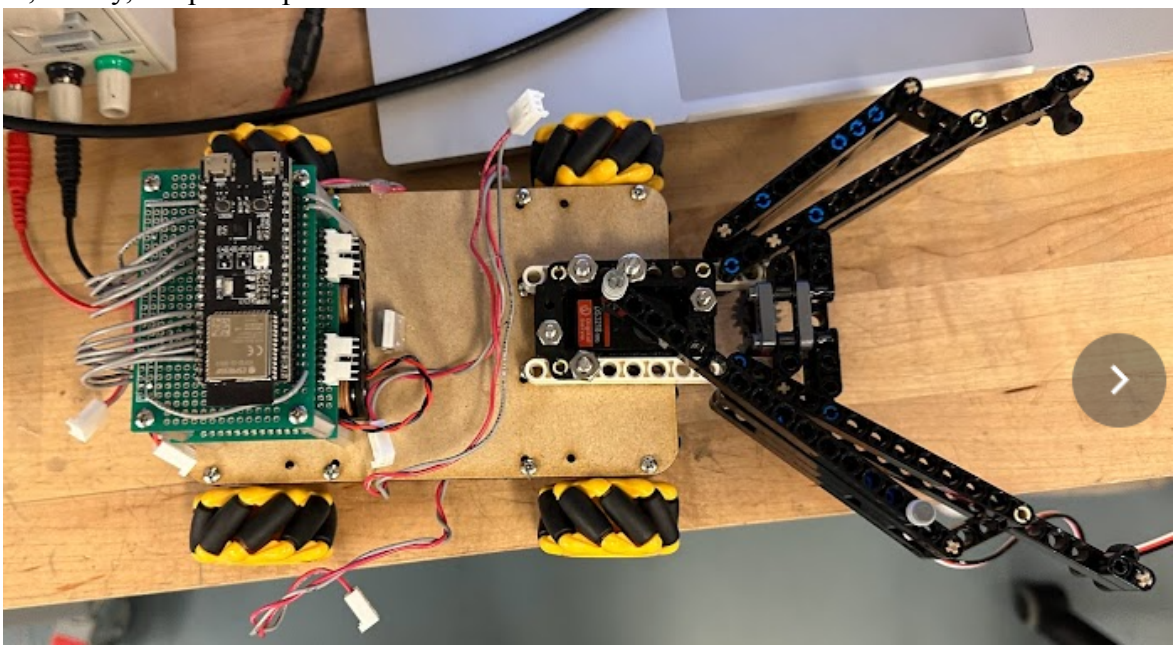
motor output simultaneously into claw clamping via worm gear and into subsequent lifting of the clamped claw, in order to lift the beacon above the ground and prevent beacon friction with the ground from interfering with our holonomic locomotion (video here).




Ultimately, however, the simple clamping via servo design was chosen for its simplicity and reliability. Shown below are pictures of the final design without the servo, open and closed.

And, finally, a top view photo of the claw with servo mounted on an earlier version of the robot.

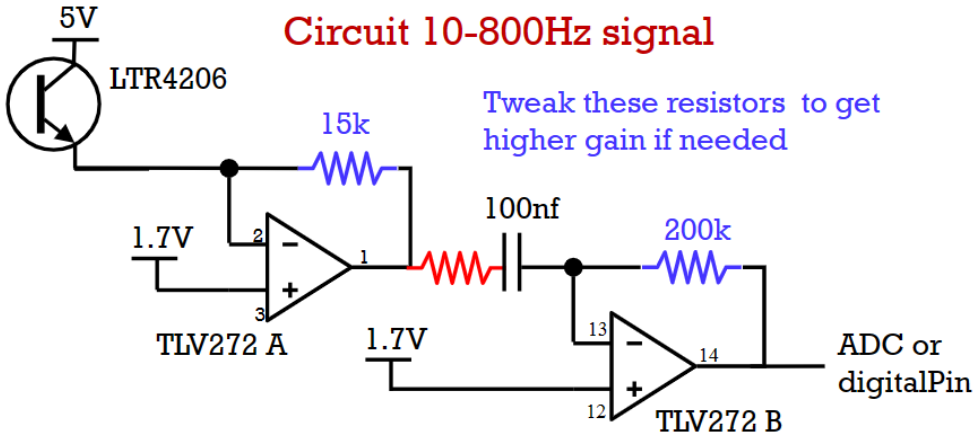


Finally, we will discuss the resultant performance of the mechanical components. The mobile base worked quite well, was highly articulate, and precise enough for our IR sensors to be used for beacon heading determination. It was also grippy and heavy enough to push the police car, though could have done so with less wheel spin and, thus, quicker had we put more weight on the robot. The claw worked basically flawlessly. The geometry of the claw was perfect for providing a tight grip on the beacons, and easily had enough grip to lift the beacons off the ground if we lifted the robot while clamping. It was not overly compliant thanks to the design with stiffness in mind, and the servo was given as much mechanical advantage over the closing of the claws as possible without losing too much motion ratio linearity. Additionally, the closing parts of the claws were intentionally made as short as possible to require less torque from the motor, thus less force through the linkages, and thus less compliance and likelihood of braking. Additionally, the oversizing of the servo torque certainly helped with gripping strength and overall reliability. If we were able to make changes to the mechanical design, they would be 1. Either adding weight to the robot for better grip, or choosing gripper mechanum wheels, perhaps that are larger as well since we had extra torque that could be converted into better speeds. Additionally, cleaner cable management and an overall shorter robot would be nice for practicality and reliability, as well.

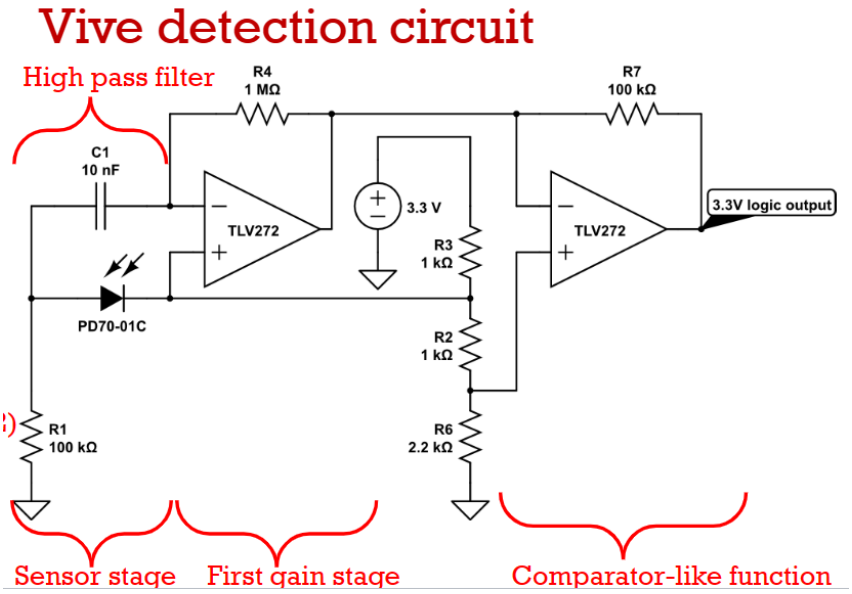
Electrical Design

The ESP32-S2 was used as the main microcontroller for our system. This was connected to four motor drivers which in turn connected into our four main motors for the holonomic wheels. The S2 was also attached to an ESP32-C3 Stamp. The Stamp was used to process the information from the TOF sensors without hindering the processing time of the S2. Two TOF sensors were connected to the Stamp, and these fed data that were used for both wall following as well as claw functionality.

Our other sensors included two Vive Circuits as well as two IR detection circuits. These circuits are illustrated below:



"IR Circuit"



"Vive Circuit"

These were the circuits provided to us in lecture. Nothing was changed with the vive circuit, and two of them were built to manage both direction and orientation. These circuits were placed on top for clear view of the lighthouse, and they were connected into the S2. The IR circuit 200K resistor was changed through testing, and we eventually settled on its replacement having the value of 3.4 MOhms. These two circuits were mounted in front with an opaque blocker separating their fields of view. These two circuits were linked back into the S2. In order to achieve the 1.7 V, a simple voltage divider circuit was implemented with the same resistor value to cut the 3.3 in half.

In order to handle power, both the C3 Stamp and the S2 were powered separately. The C3 Stamp, and by extension the TOF sensors, were powered by a lithium ion battery. This battery was routed through a voltage regulator circuit which in turn stepped down the voltage to 5V without

overheating. The motors, and the S2, were powered by a battery bank of four AA batteries. These batteries fed into the motor drivers as well as the S2.

In order to view the overall circuit diagram, please refer to the circuit schematic in the appendix.

Processor Architecture and Code Architecture

High Level Design Approach

We chose to take a software-first approach for the robot. We sought to minimize the hardware requirements and effort and resolve any hardware performance issues purely with software. In this way, software design and limitations dictated the bare minimum hardware requirements that were ultimately implemented on the robot. To enable this approach, software was continuously designed and written before and in parallel with hardware manufacturing. Furthermore, due to the vision restrictions and the emphasis on performing tasks autonomously, we also decided to approach software design with the intent that the robot would be fully autonomous. We reasoned that any decision we made using data received from the robot could be encoded in the software.

The first step for software development was to determine the absolutely necessary information required to perform each task, which would be used to inform hardware development and requirements. For location transmission, it was necessary to know the robot's position, and any coordinate-based movement system would need to know the robot's orientation. For hardware, this meant that the robot required two vive circuits. For wall-following, the robot needed to be able to measure its distance from obstacles in front of it and to its side. This required at least one angled time-of-flight sensor. For trophy navigation, the robot had to be able to detect and calculate the relative direction of IR signals as well as determine when a trophy was in grabbing range. This meant the robot needed two IR detection circuits and one front-facing time-of-flight sensor. For police car pushing, coordinate-based movement could be used, so it did not require any additional hardware. Thus, the robot required two vive circuits, two IR detection circuits, and two time-of-flight sensors.

Once hardware requirements had been set, the next step for software development was to create the algorithms to perform each task. Many weeks were spent on algorithm development as different scenarios were devised and considered and processes were abstracted and refined. Eventually, we determined that trophy navigation and coordinate navigation could be achieved using a single movement pattern. Both required the robot to calculate and achieve a desired heading and maintain that heading while moving towards the goal. Thus, we settled on the following movement algorithm (`move_to_template` in `robot_movement.cpp`):

```
while (goal is not reached and robot is not stuck):
    while (heading is not reached and robot is not stuck):
        adjust heading
    while (heading is reached and goal is not reached and robot is not stuck):
        move forwards
stop
return whether or not the robot is stuck
```

This algorithm continuously achieves the desired heading and moves along the desired heading until the robot reaches its goal or determines it is stuck. It returns whether or not the robot is stuck so that next steps can be determined. The reason why the heading must be continuously checked is because the robot is not guaranteed to move in a straight line, so it could veer from the desired heading while moving forwards. Abstracting the movement algorithms meant that we only needed to make a single functioning algorithm to perform two tasks. For each movement type, the goal, heading, and heading adjustment could be specified separately. For wall-following, two separate algorithms were devised. One relied on coordinate-based movement, while the other involved maintaining a specific distance from the wall using time-of-flight sensor data. Ultimately, the latter algorithm was used as it was more robust and made less assumptions about the environment. The wall-following algorithm was as follows (`wall_follow` in `Final_Project_master_team17.ino`):

```
while (wall following):
    if (not too far and not too close):
        keep going straight
    if (too far):
        turn in / right
    if (too close):
        turn out / left
```

Once the algorithms were created, the next step for software development was implementation.

Low-Level Design

For software implementation, there were three important issues to consider: runtime, data accuracy and precision, and adjustability. All three heavily dictated software design.

The first tasks to be implemented were sending and receiving communications using ESP-NOW and UDP, respectively. Sending robot position data was straightforward and involved setting up ESP-NOW and attaching a send function (`ping_location`) to a 1 second timer interrupt. The modulo operator was used to ensure that position data remained under 4 digits so as to avoid stack smashing issues. Receiving police car information was also straightforward and involved setting up UDP and filtering UDP data for the police car's location (`update_police_car`). Through testing, it was determined that processing UDP data took too long to be implemented in a timer interrupt, so the processing was only conducted when the robot was tasked with pushing the police car.

The next tasks to be implemented required the time-of-flight sensors. The time-of-flight sensors have a standard frequency of 30Hz and require at least a few milliseconds to generate a new distance reading. Thus, we became concerned that constantly updating distance readings would consume too much time and would lead to delays in the robot's response time. For this reason, the decision was made to process time-of-flight sensor data on a separate MCU. Thus, we used an ESP32 Stamp-C3 as a minion whose sole purpose was to constantly process readings from the time-of-flight sensors and relay the information to the master ESP32-S2. We determined that it was unnecessary for the master to know the exact distance measurements and that it only needed to know if an obstacle was too far or too close or a trophy was within grabbing range. Therefore,

to improve the speed of communication between the master and minion, 4 GPIO pins were used to communicate different states. The original implementation had 4 states: if the beacon was within grabbing range, if a front obstacle / wall was too close, if an obstacle / wall was too far, and if an obstacle / wall was too close. We ultimately ended up using only 3 states: if the beacon was within grabbing range, if an obstacle / wall was too far, and if an obstacle / wall was too close. The connections between the master and minion to relay these states can be seen in the diagram of the MCUs. The thresholds for grabbing range, too far, and too close were made to be easily adjustable and were fine-tuned during hardware testing.

The wall-following algorithm was straightforward to implement once the minion was set up. The duration of wall-following was set to 45 seconds. During this time, the robot was commanded to continuously move while reading the signals from the minion and adjusting its steering accordingly.

The most important and involved task to be implemented was coordinate-based movement. For this task, goal testing, heading testing, and heading adjustment had to be defined. Goal testing involved comparing the robot's position to the desired position and determining whether they were close enough. This was determined by an adjustable threshold, which was important to account for the imprecise vive readings and stop the robot from thrashing due to a low threshold. Heading testing involved determining the robot's orientation and comparing it to the orientation required to travel in a straight line to the goal. This was also determined by an adjustable threshold as the imprecise vive readings meant it was nearly impossible for the headings to match perfectly. Heading adjustment involved determining the optimal rotation direction for reaching the desired orientation. One additional requirement for the movement algorithm was a method for determining if the robot is stuck. This was implemented by determining if the robot's position had changed within a given amount of time, which was specified using an adjustable stall threshold. All of these methods were used to define a method for coordinate-based movement.

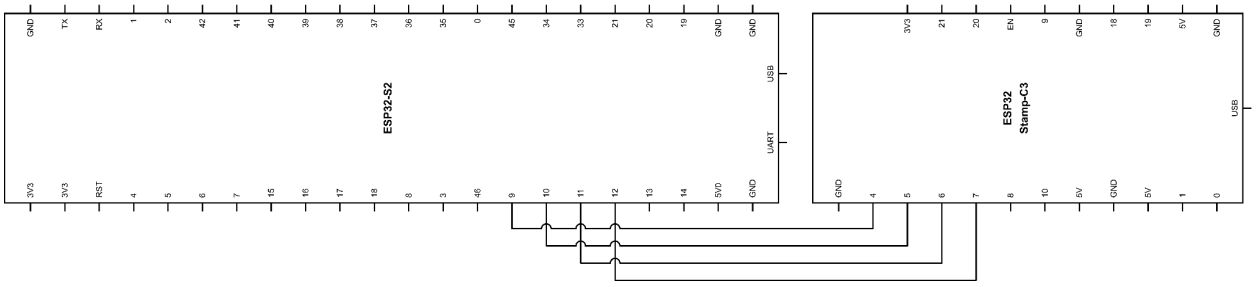
It is important to note here how robot position and orientation was determined. Checking for new vive readings was performed using a 20 millisecond timer interrupt. Unfortunately, syncing could not be performed in the interrupt as the runtime was too long. However, if the vive connection dropped at any point in time, vive readings would cease to update. This was an issue, as vive readings needed to be constantly updated to determine the robot's position and orientation. The solution to this issue was to modify the heading adjustment method. When vive readings were required, such as when adjusting heading, the heading would be adjusted a small amount, and then enough time would be given for the vive readings to update and sync if necessary. Another issue with robot position and orientation was how to calculate them using vive readings. If the vive circuits were placed haphazardly, it would be possible to calculate the location of the robot's axis of rotation and the robot's orientation by using vector projections, but this would increase runtime due to the additional computation and would require calibration. The location of the robot's axis of rotation and the robot's orientation were extremely important for coordinate-based movement, so to bypass this computation, we ensured that one vive circuit was placed at the robot's axis of rotation and the other vive circuit was placed along the longitudinal axis of the robot.

For trophy detection-based movement, goal testing, heading testing, and heading adjustment had to be defined once again. Goal testing involved checking if the trophy was directly in front of the robot and was within grabbing range. Heading testing involved determining if the trophy was directly in front of the robot by determining if both IR detectors could see the trophy. Heading adjustment involved determining which side the trophy was on and rotating towards that side. It is important to note here how the IR detection software was implemented. Pin interrupts were used to store the time each IR detector detected an IR light turning on. Multiple times were stored to reduce the effect of noise. The period of these times were then calculated to determine if the correct trophy was being detected. It was also important to check that the times were being continuously updated, as the old stored times would still contain the correct period and no new times would be added if the IR detector saw no IR light.

Once the movement algorithms were implemented, the higher-level tasks could be implemented. For trophy navigation, the robot used coordinate-based movement to navigate to an optimal position for finding trophies. It then searched for the trophy by rotating. Once it located the correct trophy, it used trophy detection-based movement to move to and grab the trophy. It then checked if it had successfully grabbed the trophy by moving backwards and determining if the trophy was still in grabbing range.

For police car pushing, the robot moved to an optimal pushing position using coordinate-based movement. The robot was then commanded to move forwards until the police car stopped moving.

Diagram of MCUs



Performance

Due to the software-first approach and multiple delays in hardware manufacturing, software development proceeded for weeks without being able to be tested and validated on actual hardware. The team member responsible for software development used this time to optimize and refine the software and algorithms and devised and accounted for many different scenarios and edge cases. The result of this effort was that, after adjusting thresholds during initial robot testing, the software was fully functional. When software was first tested on the fully assembled hardware, all issues were actually traced back to hardware failure. Therefore, once all the hardware was finally completed, only a little time was spent on tuning thresholds before the robot was able to perform all tasks. For this reason, despite there being continuous delays in hardware manufacturing, we were able to bypass software debugging and complete the robot before the deadline.

Retrospective

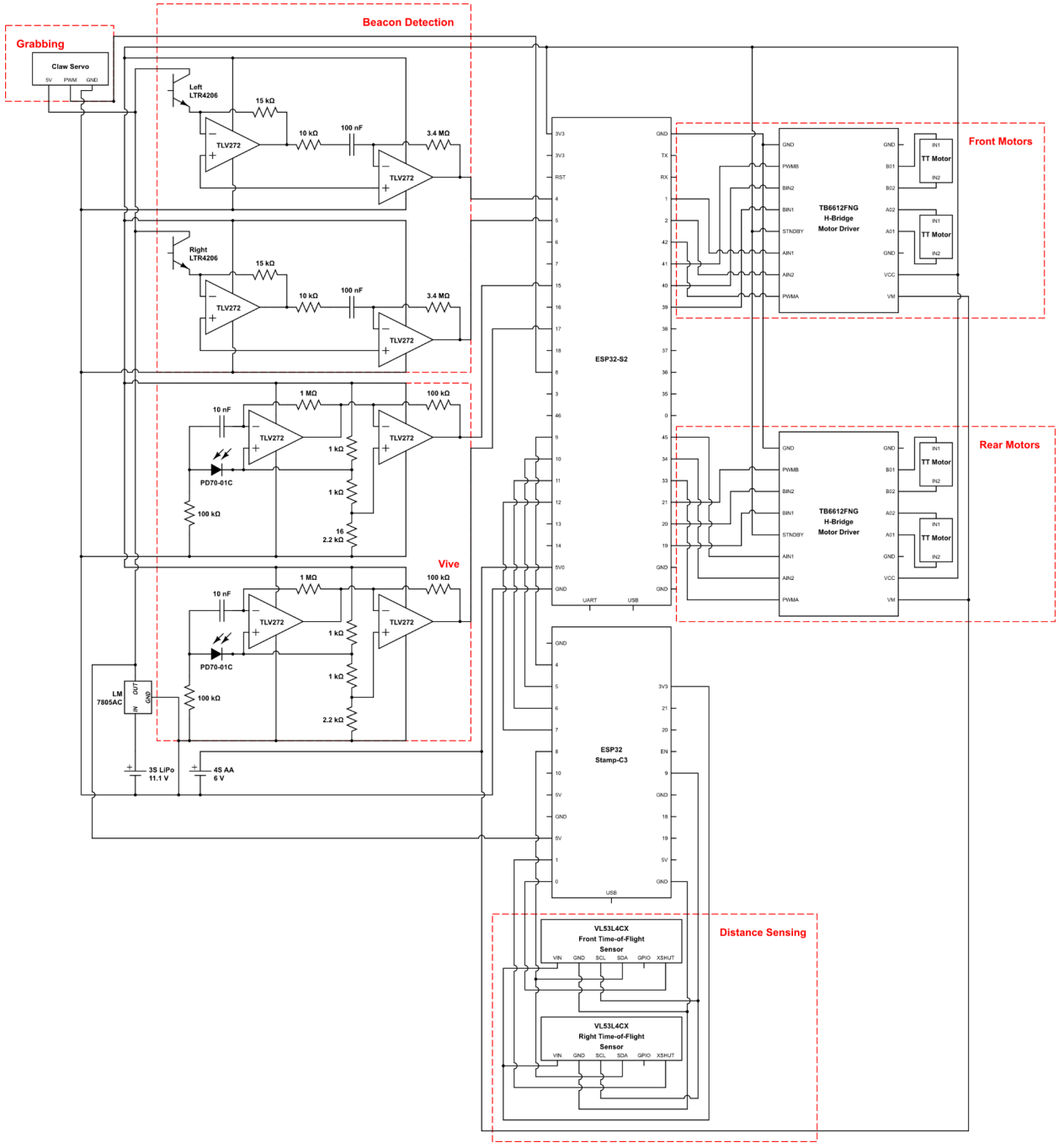
This class was a wonderful learning opportunity due to the main fact that we were able to put so much theory into literal practice. Because we had the freedom and resources of so many components as well as 24/7 open access to the lab as well as a variety of TAs, many of the normal barriers to learning were removed. This class was well put together, well organized, and the professor was an active and crucial part to students learning and understanding of the topics presented. I personally feel much more confident in reading a data sheet, tinkering with various microcontrollers, ordering parts to complete a project, and actively taking steps to make something new. I appreciate the time and energy the teaching staff has given to us, and I hope our enthusiasm was displayed in this report as well as our collective accomplishments throughout the semester.

Appendix

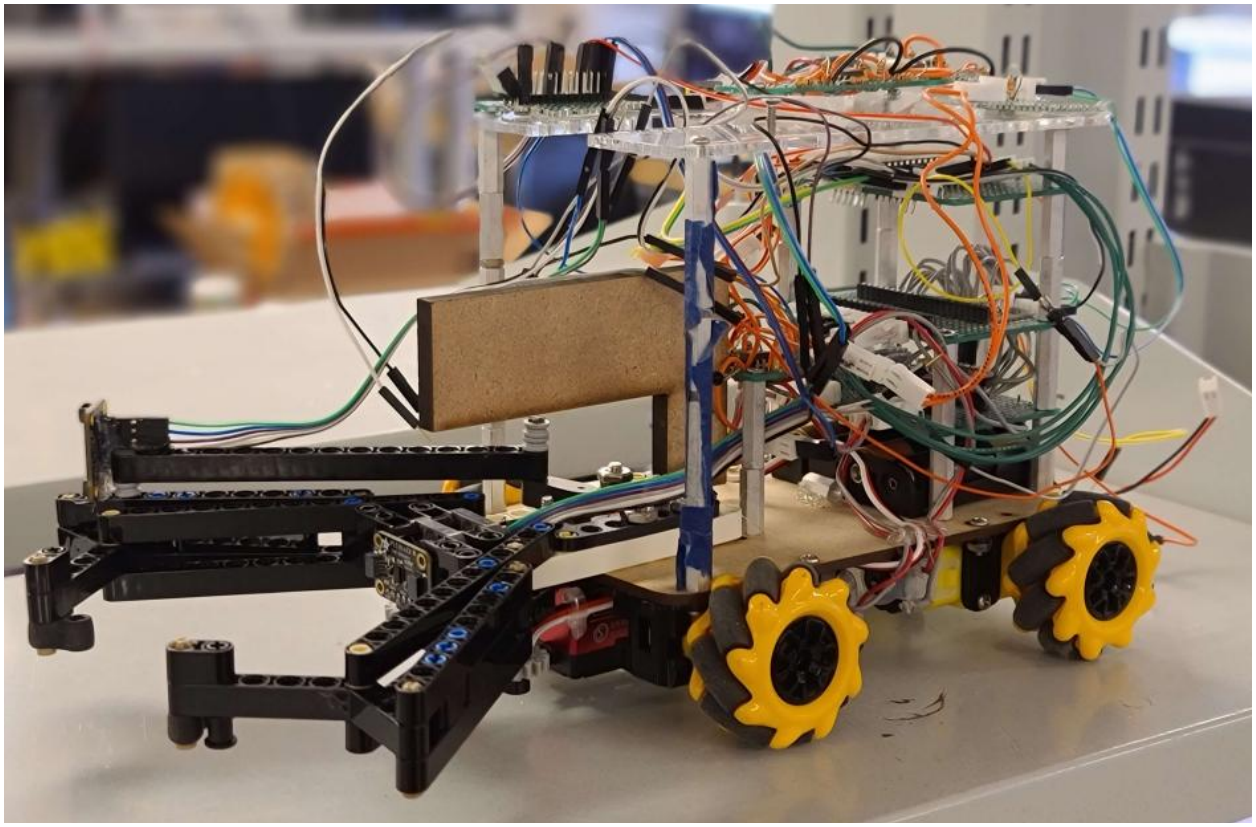
Bill of Materials

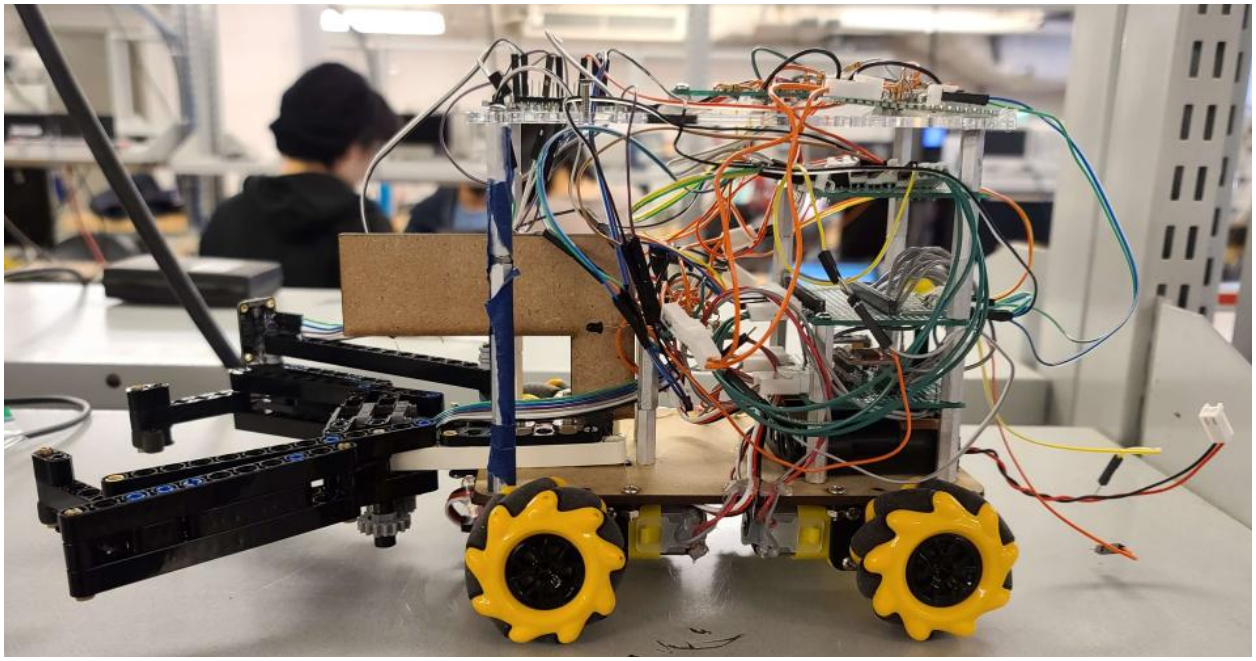
Item	Units	Price Per Unit	Cost	Actual Cost	Notes
ESP32-S2-Dev KitC-1	2	$8	$16	$8	1 from Lab 4, 1 as backup
ESP32 Stamp-C3	1	$6	$6	None	1 from Lab 4
VL53L4CX ToF Sensor	2	$15	$30	$30	Purchased from staff
TB66112FNG Motor Driver	2	$9.50	$19	None	Already owned by Griffin
20KG Servo	1	$17	$17	None	Already owned by Griffin
TT Motor	4	$2	$8	None	4 from Lab 4
Mecanum Wheel	4	$5	$20	None	4 from Lab 4
AA Battery	16	$1.50	$24	$24	Purchased from CVS
Total			**$140**	**$62**	

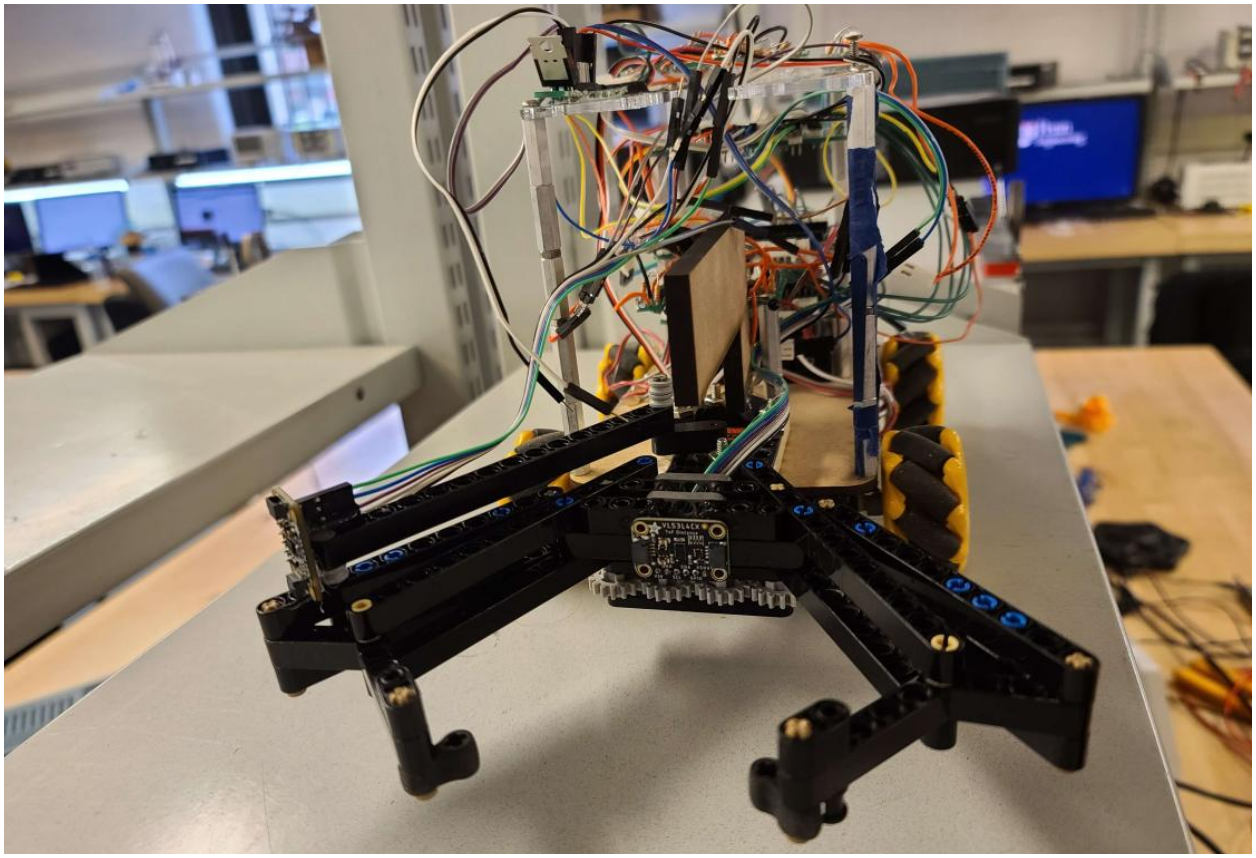
Circuit Schematics

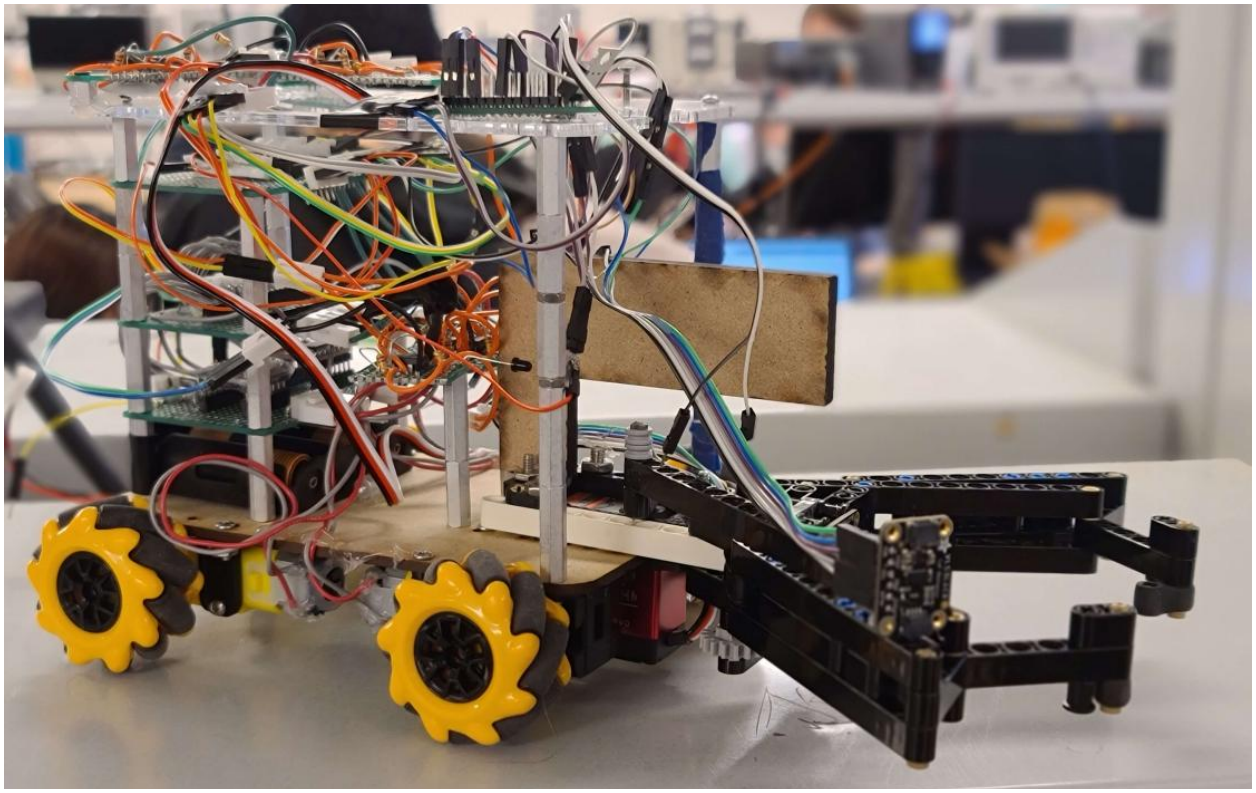


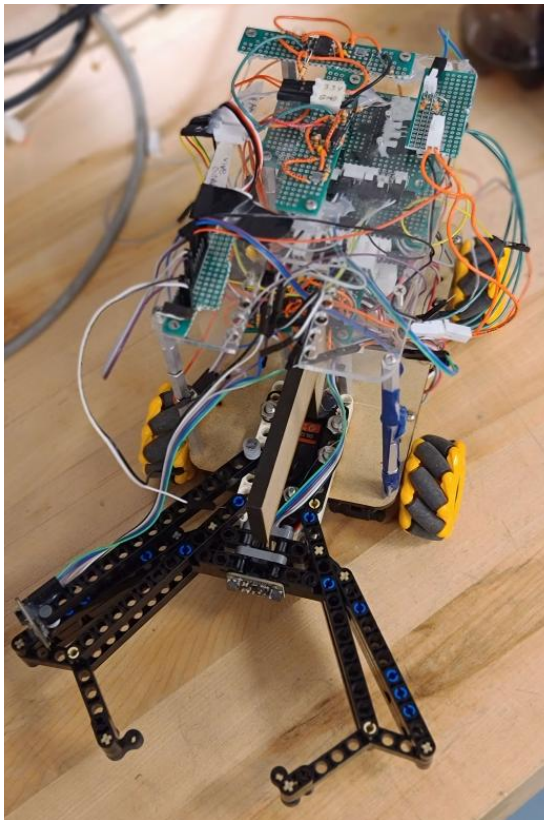
Photographs

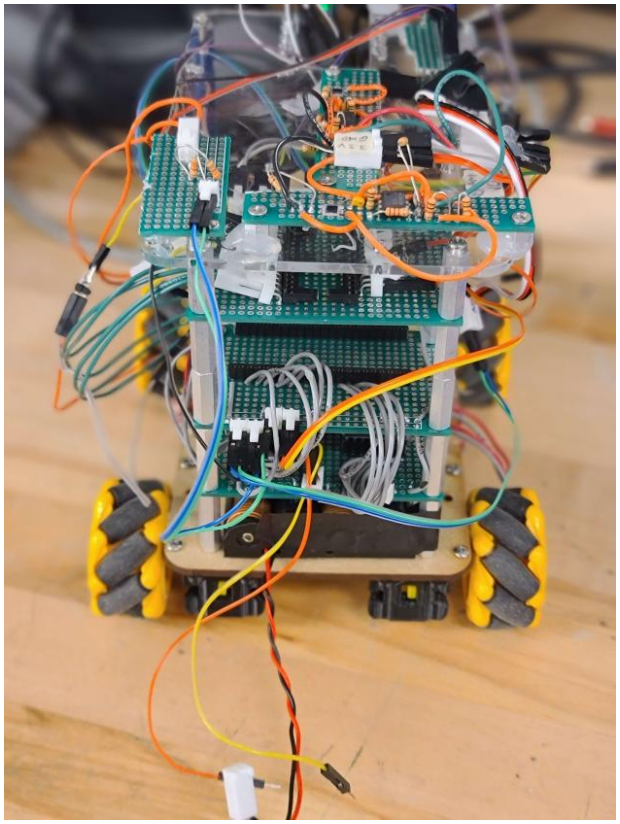


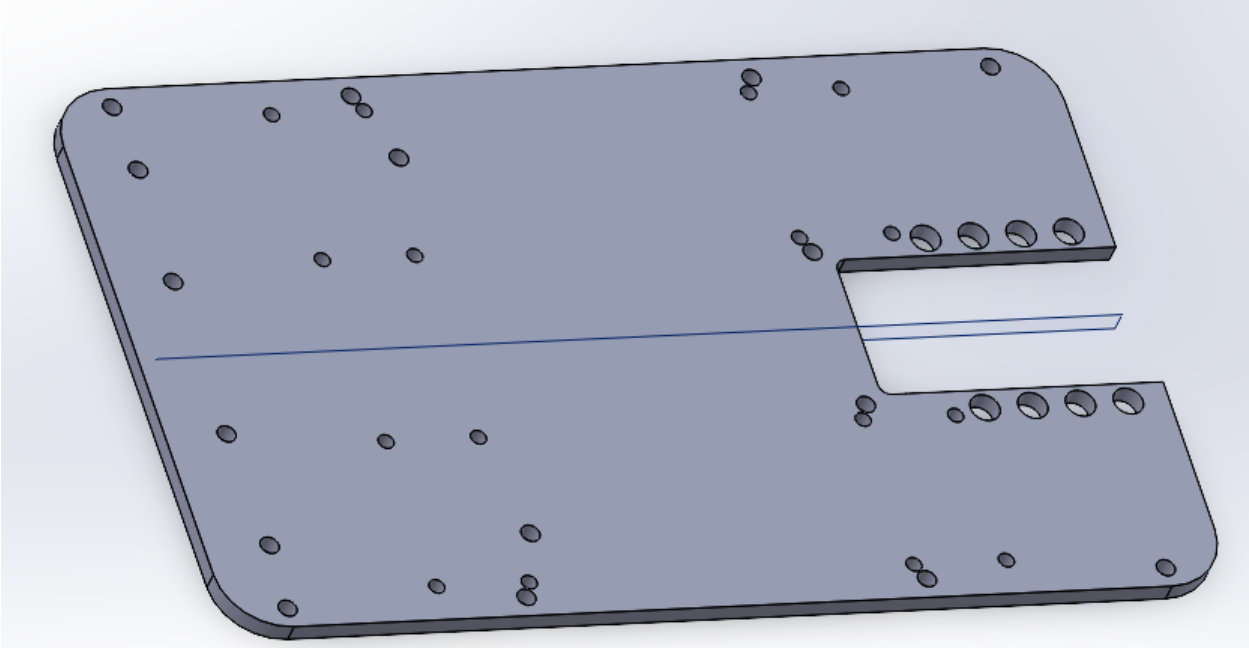




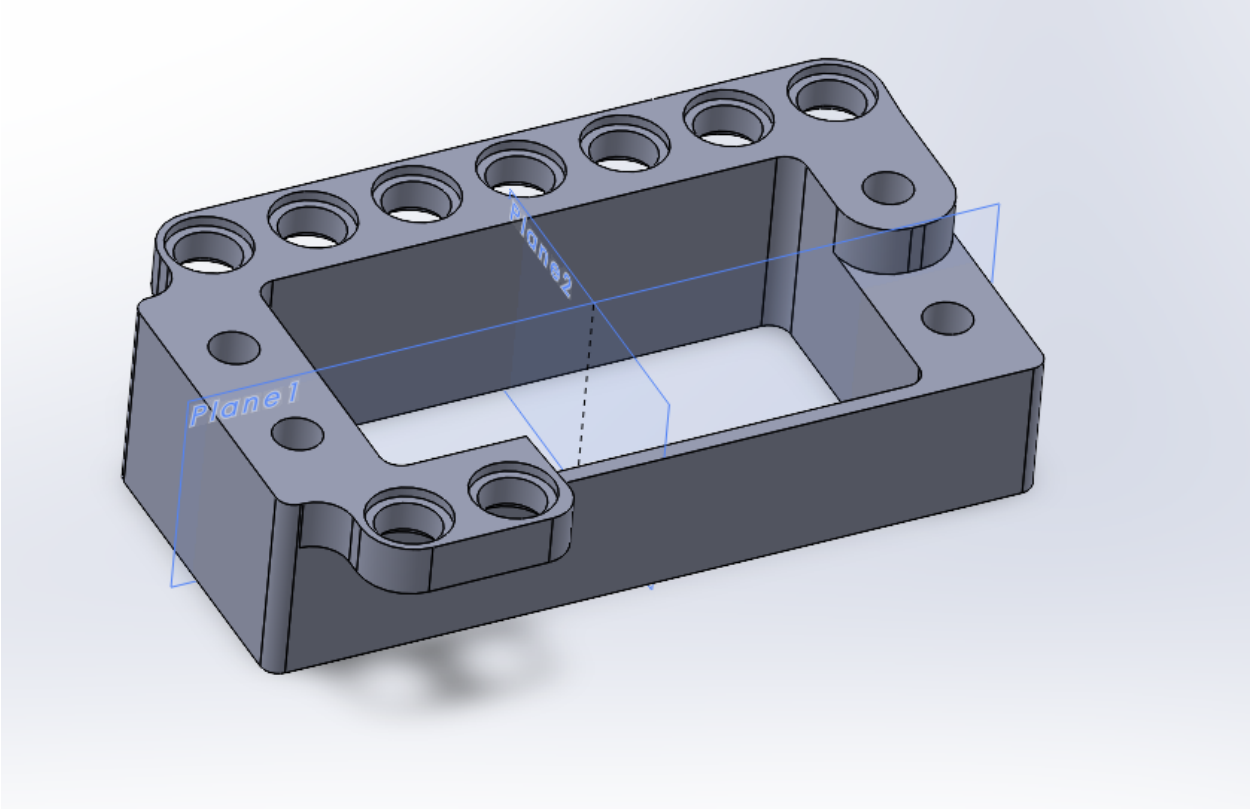








CAD of MDF baseplate (chassis)



CAD of Lego-to-servo custom 3D printed adapter

Datasheets

- VL53L4CX: https://www.adafruit.com/product/5425
- TB6612FNG: https://www.sparkfun.com/datasheets/Robotics/TB6612FNG.pdf
- 20KG Servo: https://m.media-amazon.com/images/I/81A7ByLKqDL.pdf

Functionality Video Link

https://www.youtube.com/watch?v=Z9yA_-1Pns4