# EAP Robot

# (MCEN 4228 - 5228 Spring 2024)

**Chammy The Robot Chameleon**

**Team Members: Allister Sequeira - Ely Toledo - Preston Brumley - Whit Whittall - Nick McConnell**

# I. Introduction

The EAP (excellent autonomous pet) involved a seamless integration between lidar, vision, and movement. The objective was to utilize ROS (robotic operating system) to create a functional robot that was able to navigate and pick up objects autonomously. The lidar provided sensing for the SLAM application to generate a live map of the area, while the vision system would identify and guide the robot to the predetermined object.

The robot outlined within this design report is designed to interact with objects and retrieve them while being able to navigate any area autonomously. This EAP robot performs the required utilization of vision and lidar integrated in ROS to move about and function properly.

# II. Technical Approach

The technical approach encompasses the design requirements outlined by the provided documentation. These requirements are derived from what we as a class envision a functioning autonomous pet will be able to perform. The intent is to create a unique challenge for the mechatronic system to compete in. All design requirements are outlined in the competition documents [1].

## A. Design Requirements:

The EAP should be autonomous and mobile. The speed and how it utilizes the locomotion is up to the teams decision, but it must navigate a small open area. It is not allowed to be tethered to a stationary object however it can utilize a “leash” as part of the user interface.

For this project, the budget for the EAP must be under $250. This budget is for all aspects of the project including: drive system, chassis material, additional motors, interfaces, and other accessories. Components resourced for free are considered free for budget purposes and lidar, depth camera, and lipos provided by the class do not affect the overall budget.

The design and functionality is left open depending on what each team wishes to pursue for their EAP. They can focus on the user interface and personal interaction, mobility and manipulation systems, or try to achieve a good balance. Possible functions include quadrupedal locomotion, responding to verbal commands or hand signals, reacting to the facial expression of its guardian, following its guardian around, playing fetch, navigating an agility course, or anything else that is important to the experience with the designed EAP.

# III. Design Approach

In order to produce a functioning EAP, several factors must be considered. Knowing it was not feasible to include all EAP functionality within a short timeline, it was essential to determine what functionality did we want our EAP to have. After brainstorming, it was decided to pursue the design and functionality of a chameleon, as seen in Figure 1**.** The main component that was identified as essential was utilizing the lidar sensor for navigation. Using the lidar would allow us to have a dedicated sensor to navigate while we could implement other smaller modules for functionality. While the depth camera provided would also allow for navigating around, the lidar provides 360 degrees of sensing which we could use to produce a map of the area.

The next major function we wanted the chameleon to possess was the ability to locate an object and pick it up with its “tongue”. This would require utilizing a camera to identify the predetermined objects and navigate the robot to them. Then with the tongue we would be able to pick it up and bring it back to the user. The tongue feature would be a claw that extends out to grasp hold of the object.

Additionally, with chameleons having the unique ability to change color depending on their surface, we designed the chameleon to do the same. Attaching color sensors under the chassis of the robot, it could identify the color of the surface and relay that information to led strips on the robot. The shell of the robot would be made of opaque material that would help diffuse the color evenly while adding the chameleon aesthetic.

Finally, the tail of the chameleon would be constructed of linkages so we could use a servo and a tensioner to contract and retract the tail. This would add a more animal characteristic to the chameleon, making it feel more realistic.

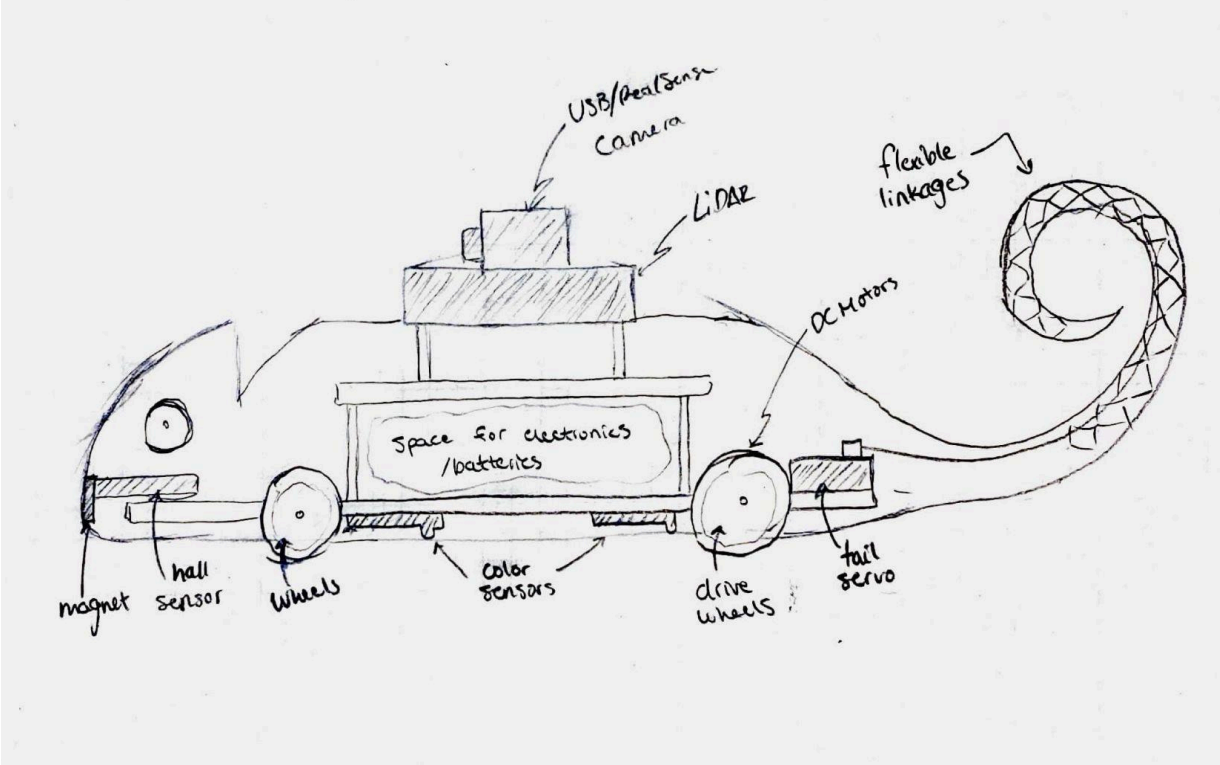


**Figure 1 –** Initial Concept Sketch

# V. Design

## A. Electronics

The electronics used in this design consist of: RaspberryPi 4, Arduino Nano, lidar sensor, USB camera, brushless DC motors with encoders, a motor driver, and a voltage regulator. The RaspberryPi was imaged with Ubuntu code and was written in conjunction with ROS application requirements. The Arduinos were utilized by writing code in the Arduino IDE and were used to drive the motors which utilized the h-bridges for direction control. The system was powered with a lipo battery utilizing a voltage regulator to ensure proper operating voltage of the arduino and raspberry pi.

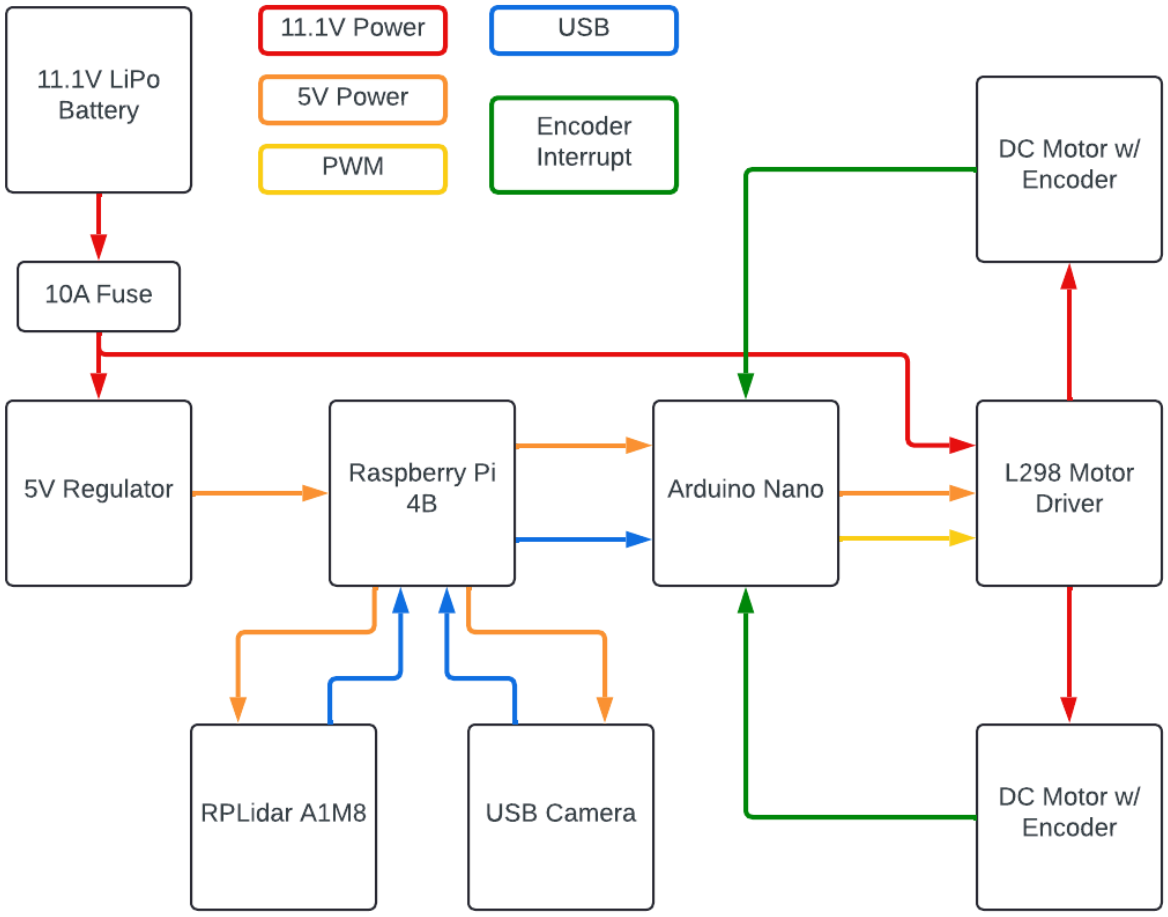


**Figure 2 –** Simplified Electrical System Schematic

### 1) Lidar:

In order to facilitate gridSLAM and pathfinding algorithms so the robot could avoid obstacles and track its position accurately, we included an RPLidar A1M8 2D lidar sensor onboard our robot. This lidar communicates and receives power over a USB serial interface and can be driven with an open-source ROS driver node provided by Slamtec. These traits simplified integration of the lidar and allowed our team to focus our efforts on implementing SLAM and other more challenging systems on the bot.

### 2) Microcontrollers:

The raspberry pi was used for its ability to image and run the ubuntu software. This was the primary means of operating the ROS program and nodes that integrated the subsystems. The raspberry pi was powered using the 11.1V lipo battery that ran through a 5V voltage regulator.

The arduino, lidar, and camera were plugged into the pi using the USB ports on the side of the device. This allows the devices to power themselves and communicate with the raspberry pi simultaneously.

The Arduino was used by the ROS program to control the drive system of the robot. It took communications through the raspberry pi and used arduino IDE code and the L298N motor controller to properly generate the robots motion. Encoder motors were used to theoretically PID control the robots range of motion. This PID control was under a closed loop control where ROS2 control would read and write data to the EAP robot without any external influence. The read data was the encoder motor output and positions and the PID information was updated based on the write data ROS2 control was sending to the arduino

Additionally, the arduino had a set of header files, driver files. Each header file associated with one node ros was being referenced in the XACRO files extended by URDF files.

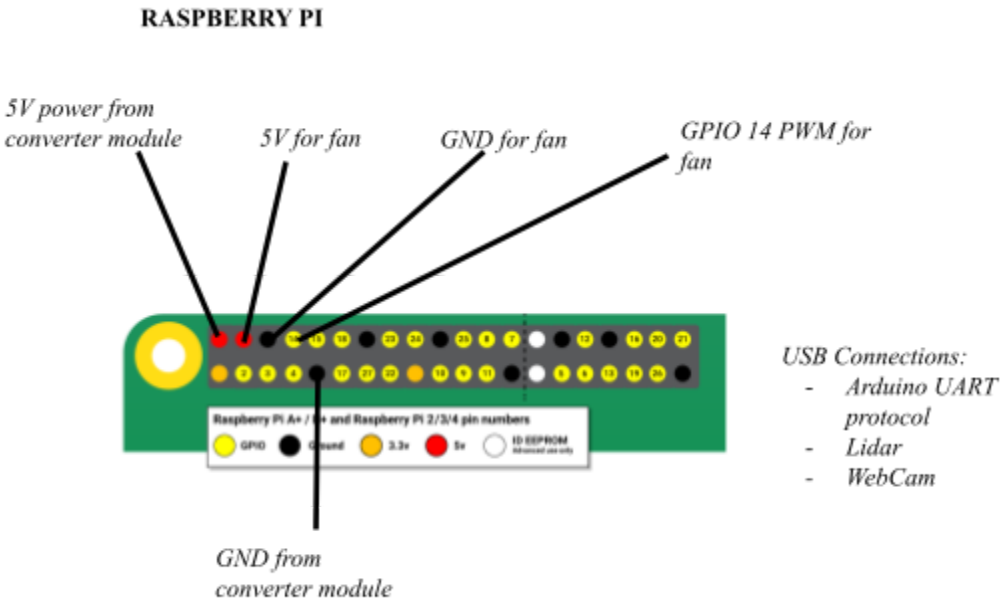


**Figure 3 –** Raspberry Pi 4 Pin Out

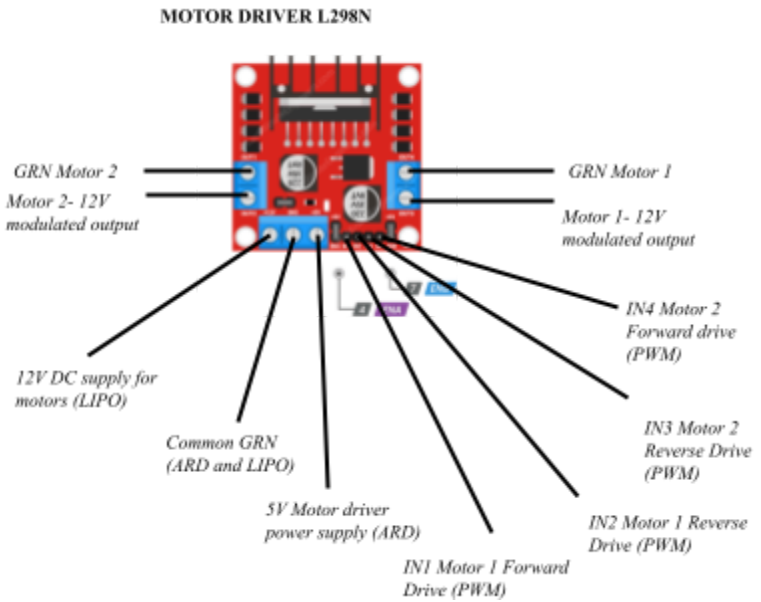


**Figure 4 –** Motor Driver L298N Pinout

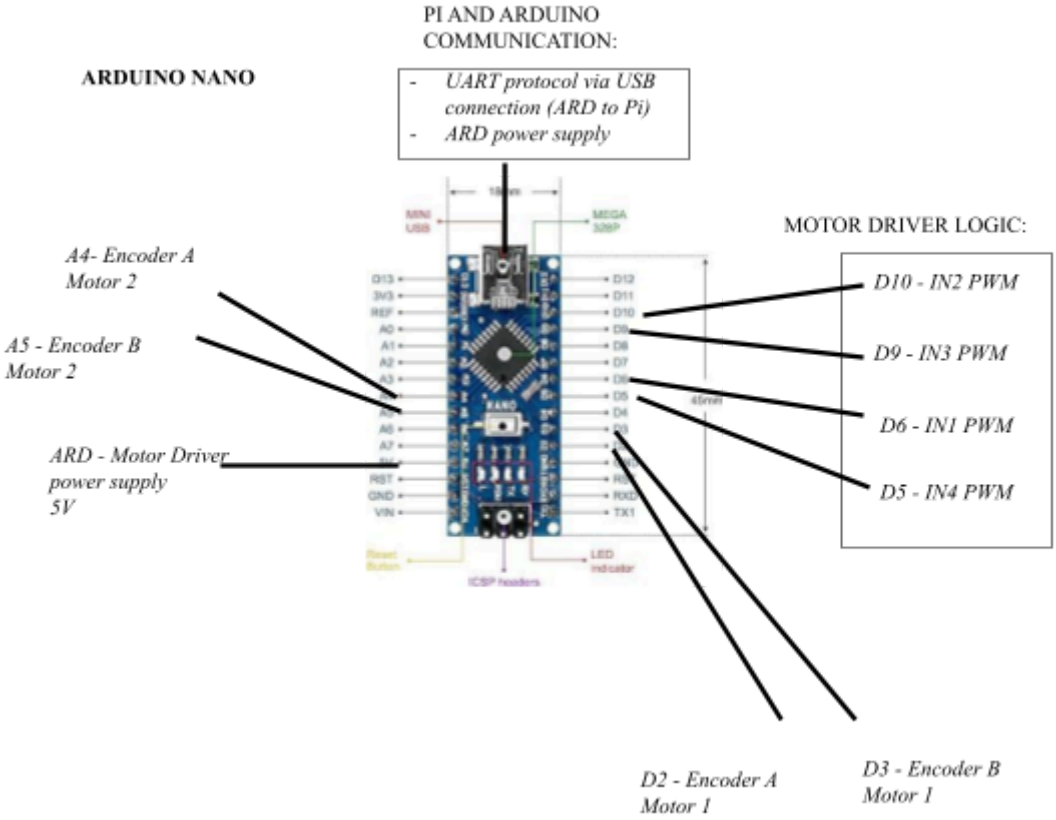


**Figure 5 –** Arduino Nano PinOut

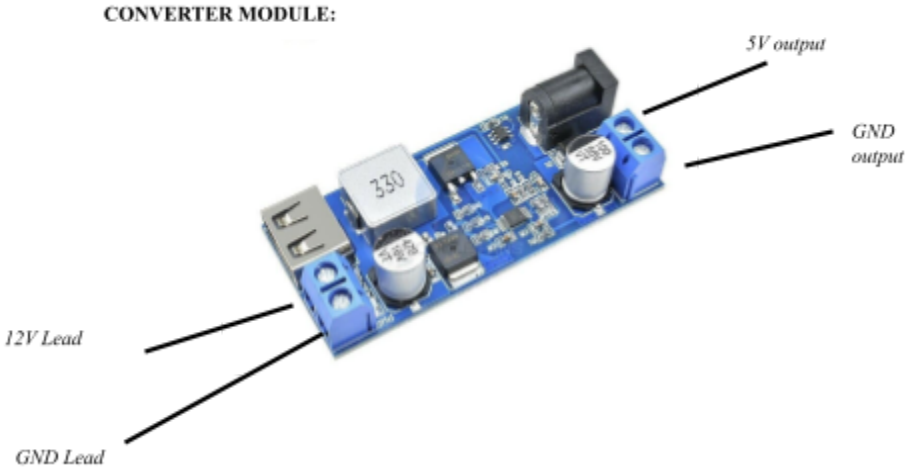


**Figure 6 –** 5V Voltage Regulator PinOut

**3) Camera**:

In order to implement a fetch function, it is necessary to add a camera module that helps identify the object. A basic USB camera was chosen as it was able to plug directly into the raspberry pi allowing for easy interfacing. The camera comes with a mounting hole making it ideal and easy to mount to the front of the robot. It was designed to be low on the chassis since it was not being used for facial recognition, and provided a clear picture of the ground and any object in the front of the robot. Implementation of the camera would allow us to see the image in the ROS program and utilizing openCV we could add filters that would allow the program to specifically trace the desired object.

## B. Chassis Design

### 1) Body:

In order to make the chassis compact, owing to the size of a chameleon, we decided to mount the components on multiple levels. This optimizes space. The lowermost level mounts the drive assembly of the robot. The drive motion is achieved with the help of two motors. The wheel motor mounts are located about a three quarters distance from the front of the chassis baseplate. The other points of contact with the ground are the two castor wheels, both mounted at the front of the baseplate. The other major components mounted on the lower level are the rack and pinion mechanism for the claw (tongue) and the tail. Since the base needed to be stiff enough we went with a 0.25 inch thick acrylic plate.

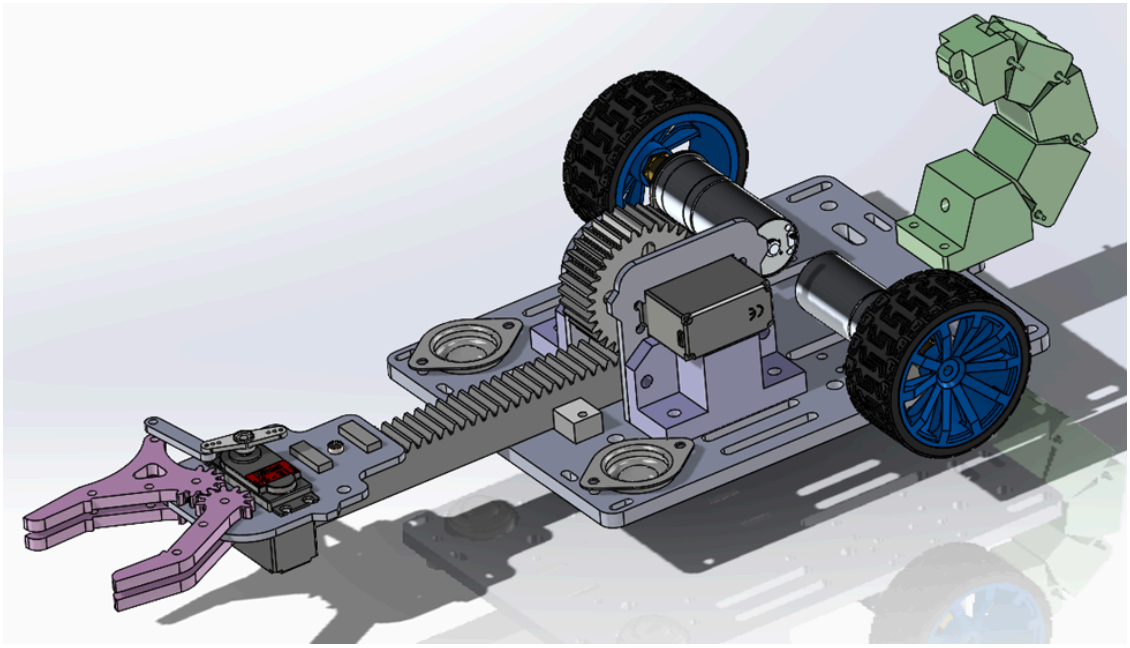

**Figure 7 –** Base level with drive train, tongue and tail mechanisms

The Arduino and motor drivers that power the wheels are mounted on an intermediate level, just above the motors. We have standoffs and bolts to connect the base plate and the Arduino mounting intermediate level.

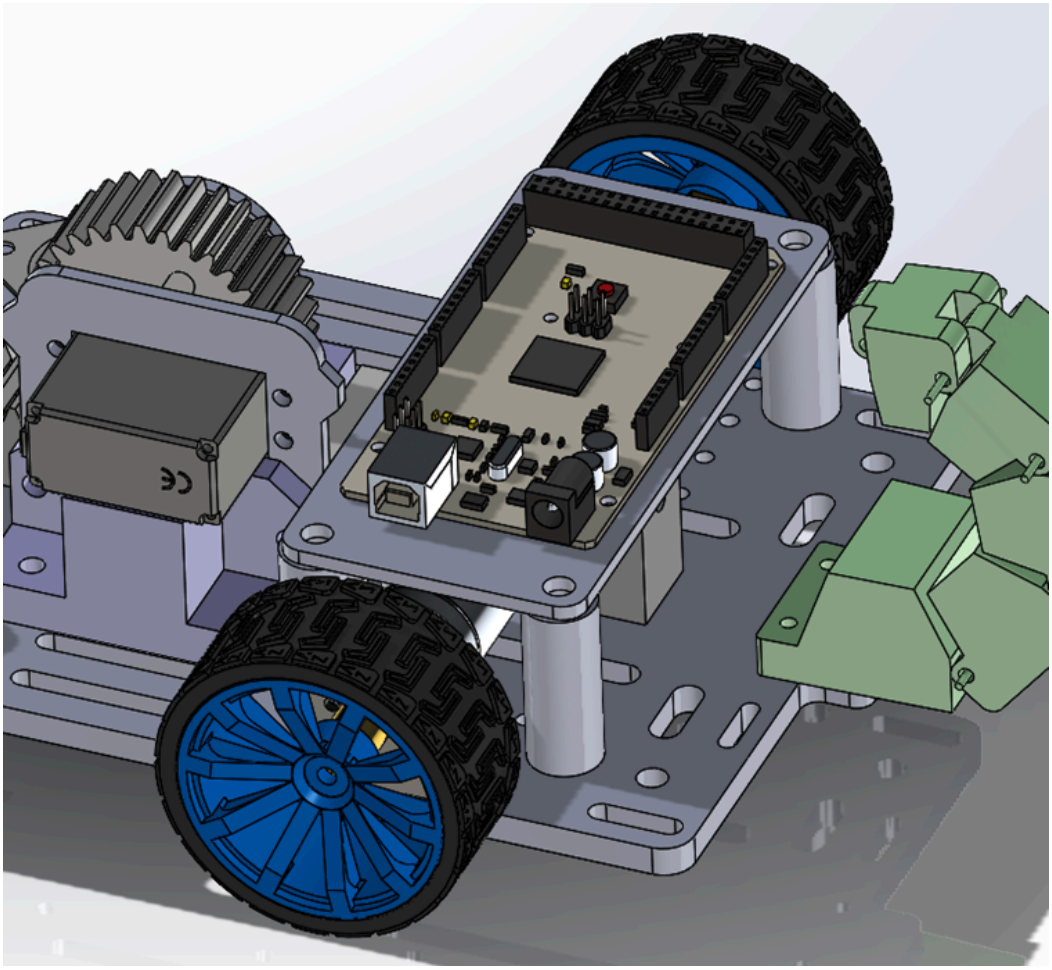

**Figure 8 –** Intermediate Level for Arduino and Motor driver mounting

The second level is mounted to the lower level with standoffs and bolts. The second level has mountings and slots for the camera, the raspberry pi and the battery. There are slots for passing wires and future mounting points The LiDAR is mounted on the third and topmost level. We placed the LiDAR high enough such that there are no components of the bot that restrict its field of view. Since the LiDAR has a horizontal plane as its field of view we made sure no components reached that level. The intermediate level, second level and the topmost level were all 0.125 inch thick acrylic sheets. Almost all components of the structure and mounts were either laser cut acrylic sheets or 3D printed.

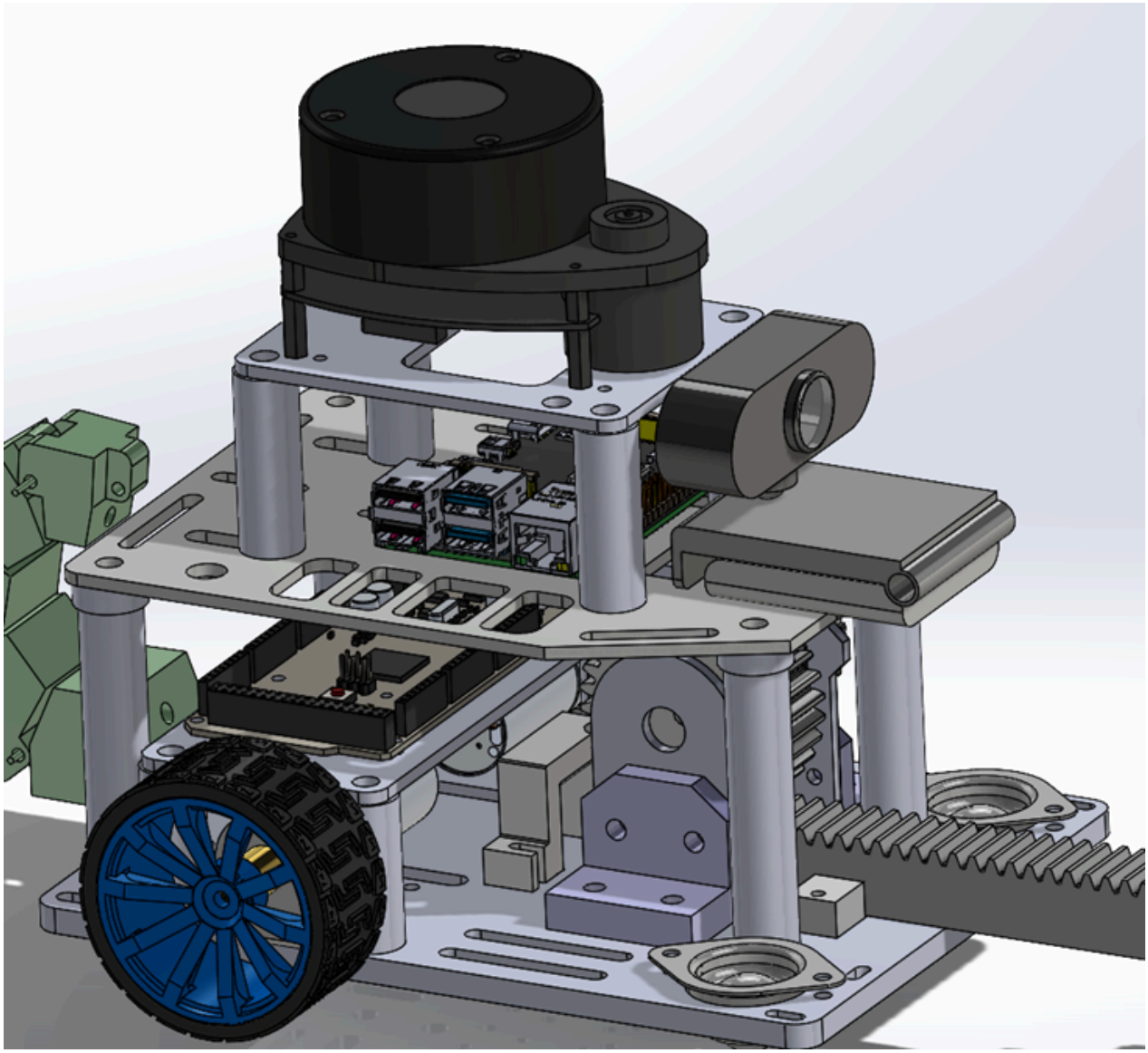

**Figure 9 –** Second and third level

The full chassis is depicted in figures 10a and 10b, within these photos, the clearance and integration of all of our systems on a physical scale is showcased. The overall chassis was relatively pretty compact and was designed for tight clearances to allow for a more realistic look of a chameleon body. Additional aesthetics features were designed to attach to the chassis but with difficulties of integration, it was decided to be left out to focus on the functionality of the design.

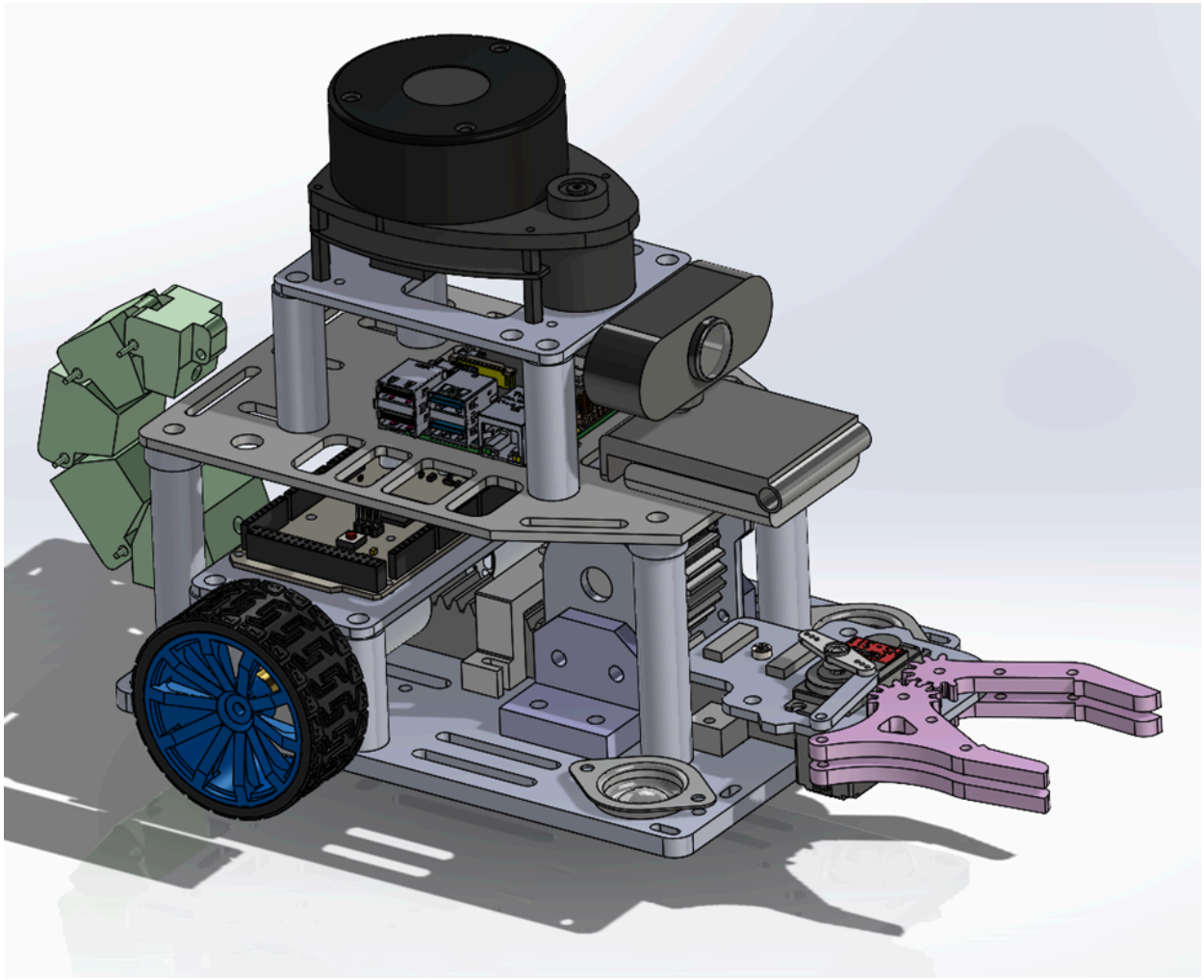

**Figure 10 (a) –** Chassis model with all components mounted

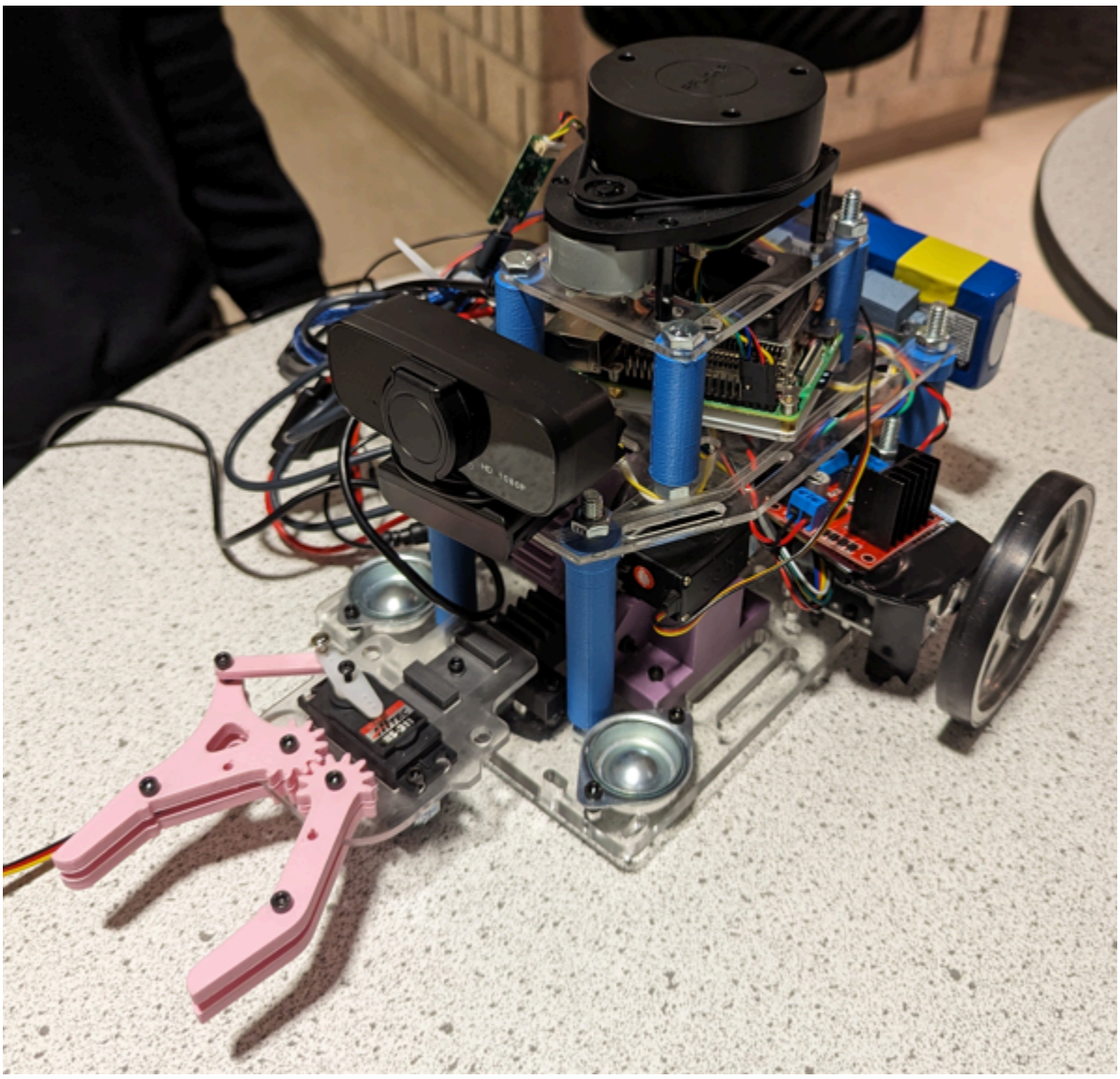

**Figure 10 (b) –** Chassis with all components mounted

**2) Tongue and Tail:**

We designed our chameleon bot to have tongue and tail functionalities. The forward motion of the rack depicts the chameleon launching the tongue and the claw clamping onto objects depicts the chameleon catching prey with the tongue. There are guides mounted to the lower level that ensure linear motion of the rack. The pinion and the claw motion are both powered by separate servos. We have the spiraling tail of the chameleon mounted at the rear of the base level. A string connects the rear of the rack with the tail. The forward motion of the rack causes the tail to spiral up and the rearward motion causes the tail to spiral back down.

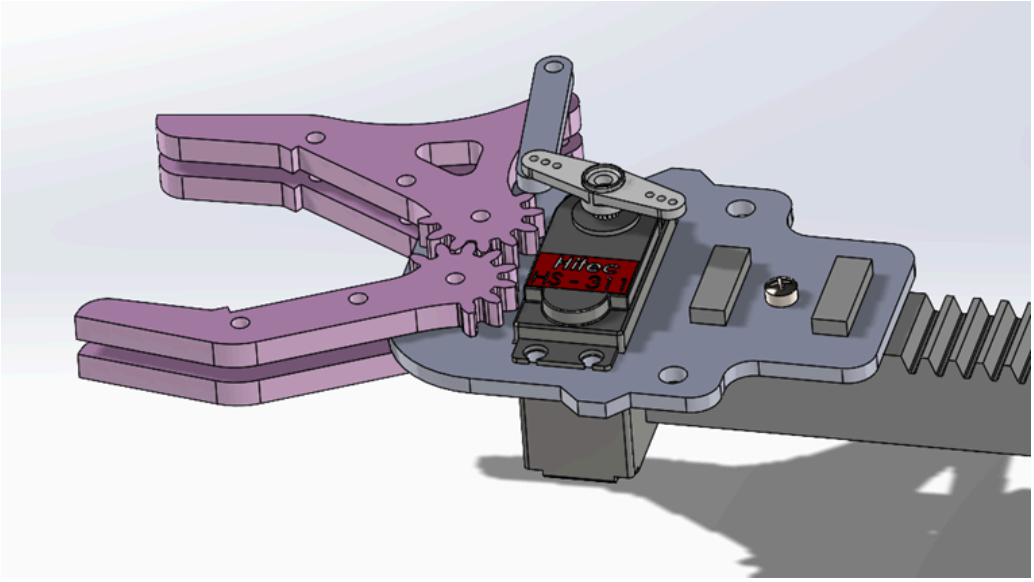


**Figure 11** – Tongue (claw)

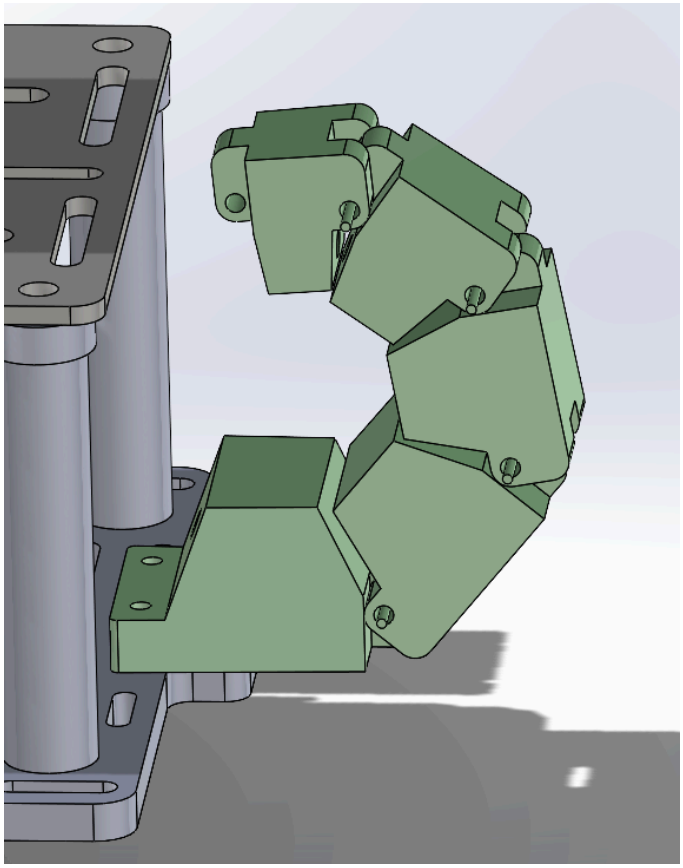

**Figure 12** – Tail

## C. Robot Operating System (ROS)

To enable integration of advanced robot algorithms, we utilized the Robot Operating System (ROS). Specifically we used ROS2 Humble with the Ubuntu 22.04 OS in order to accommodate the drivers our sensors required. ROS is a set of open source software libraries providing functions for everything from low level hardware control to advanced robotic localization and pathfinding algorithms and more. In a robotic system, ROS takes the form of a network of nodes and topics, which are conceptually similar to walkie talkies (the nodes) that receive and send messages on pre-programmed channels (the topics). By assembling this network of nodes and topics in an intelligent way, a ROS user can code specific functions and capabilities for their robot.

For testing of robot code, ROS is built to work with an associated simulation program called Gazebo. The Unified Robot Description Format (URDF) includes libraries and plugins that allow roboticists to imitate the values of a robot's sensors in a simulated environment and pass those values to the functional robot code. This allows for rapid and accurate simulation of robot behavior in a virtual environment where crashes and dangerous behavior have little consequence, and improves the safety and speed with which the real robot can be developed.

In order to achieve our EAP goals of intelligently navigating a space and playing fetch, we wanted to run a SLAM algorithm to create a map and accurately determine the robot’s position, a pathfinding algorithm to select a route through the map without crashing into obstacles, and computer vision algorithm to recognize and track the object we intended to fetch. This design means we have a pathfinding system, fetch system, and teleoperation system that are each sending different command velocities to the differential drive controller. In order to handle these conflicting commands, we used a node called twist_mux to prioritize the command velocities. Below is a simplified version of the ROS node graph to illustrate our program architecture (Figure 13).

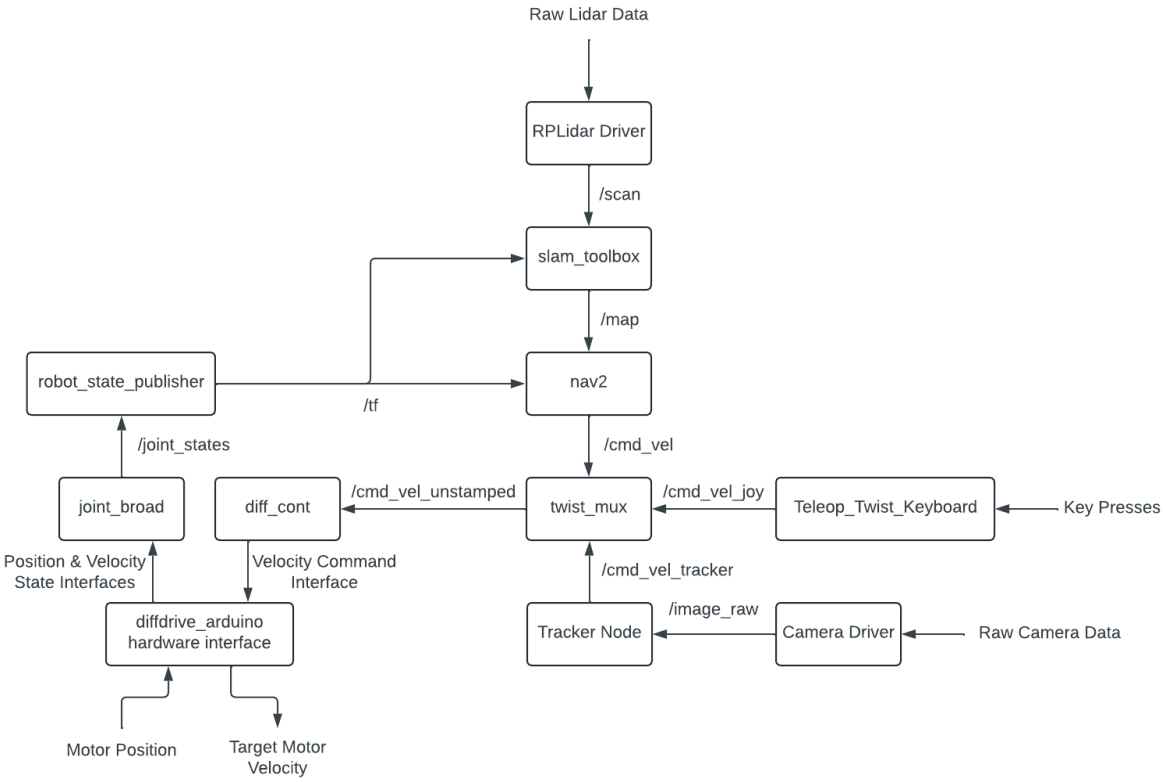


**Figure 13 –** Simplified ROS Node Diagram

### 1) ROS Lidar

The ROS system integrates with the RPLidar through the RPLidar Driver node as shown in Figure 13 when operating onboard the real robot. In the simulation, a simulated lidar built with Gazebo-provided sensor types and plugins takes the place of the RPLidar and lidar driver node. Using gazebo tags under the lidar.xacro description file we are able to construct a ray sensor that samples distance in 360 degrees at a 10hz update rate. This distance data is then published as a sensor_msgs/LaserScan message on the /scan output topic, imitating the output of the lidar driver node it replaces. This allows the remaining ROS system which subscribes to the /scan topic to function as it would if the RPLidar and lidar driver node were still running. The slam_toolbox is the node through which the ROS system listens to and makes use of the lidar data, producing a map of the robot's surroundings and an updated estimate of the robot's position within the map to correct the error inherent in the robot odometry.

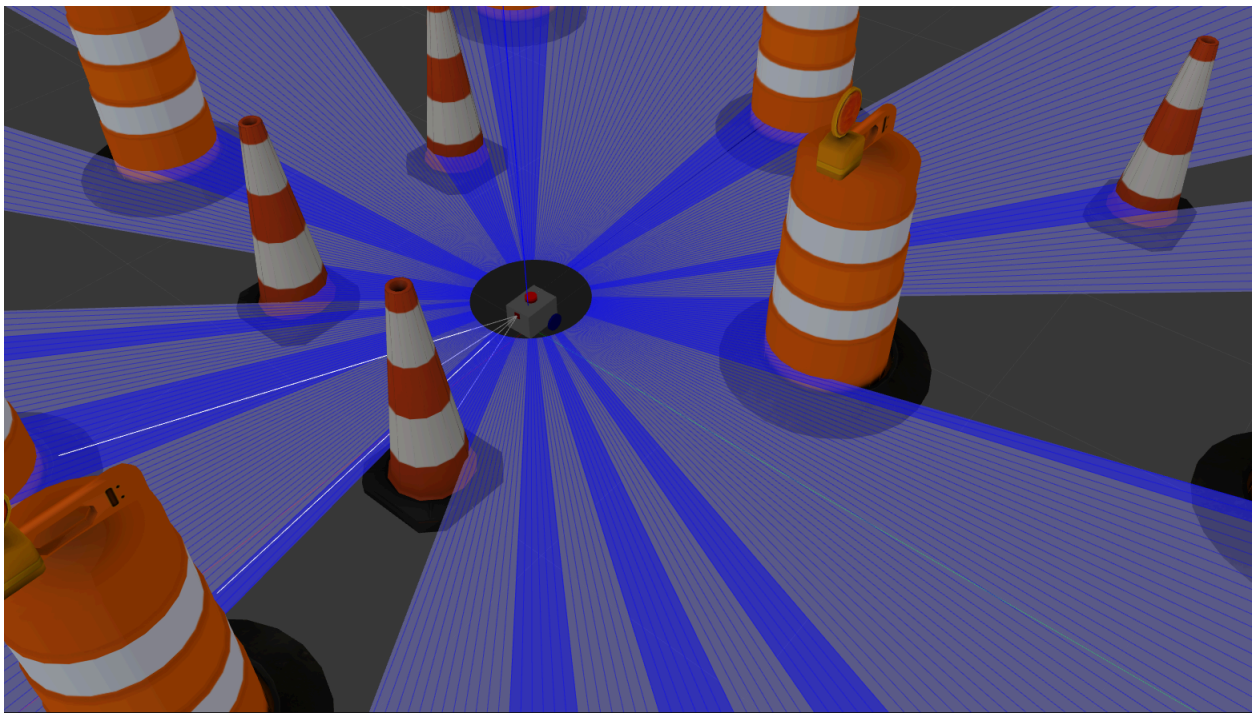

**Figure 14 –** Visualization of Simulated Lidar in Gazebo

**2) ROS Camera**

With Gazebo and Rviz2 you are able to simulate a camera in the ROS environment. This allows us to test the object tracking functionality of the robot with a simulated robot and object. After implementing a camera module in Gazebo, linking in Rviz2 displays the camera image that is being posted by the node. Running an image tacking filter [3] program allows us to adjust multiple sliders that affect the camera's min and max values for the view box, hue, saturation, and brightness. In order to determine the correct values to use, a spectrum plane was imposed behind the object in gazebo. This allowed us to see the right color needed during configuring the parameters. This then allows us to record the filter parameters into a separate file that the tracking launch file will utilize. Running a ball_tracking topic, we were able to implement multiple nodes that returned the X,Y,Z coordinates of the object that was in frame. These values provided us with real time data of where the object was located with respect to the center of the image frame.

With the ball tracking filters set, and the ball coordinates known, another node can be ran that will tell the robot to follow the object. This program directs the robot to scan the area till it can find the predetermined object. Once the object is in the camera view, it will begin driving towards it until the object has reached a predetermined length away.

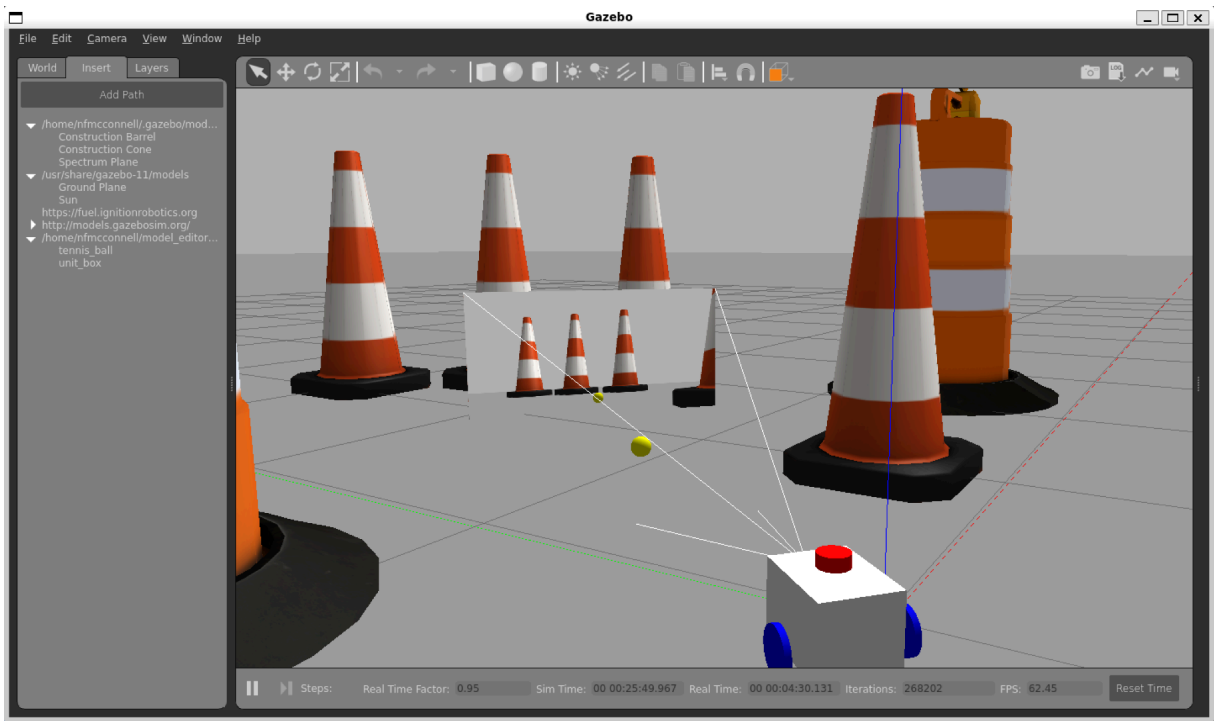


**Figure 15 –** Camera view in Gazebo

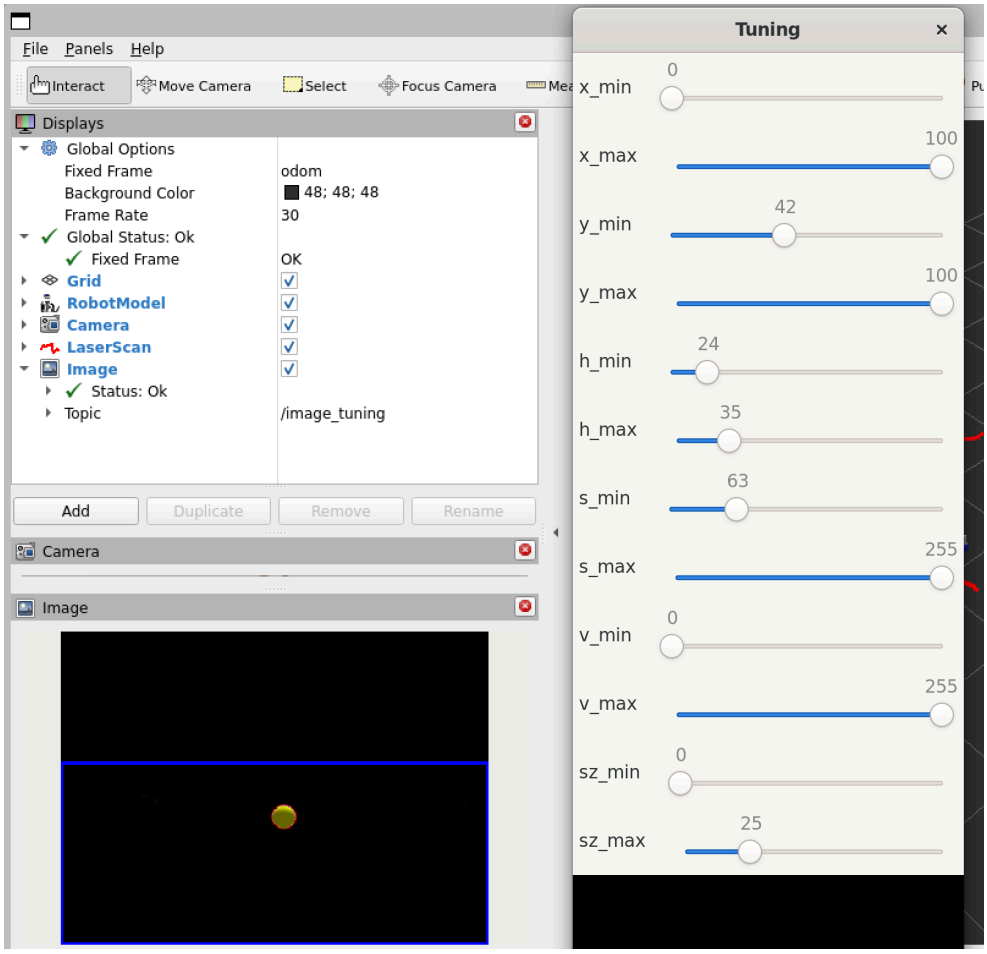


**Figure 16 –** Camera filtering of the object in Rviz2

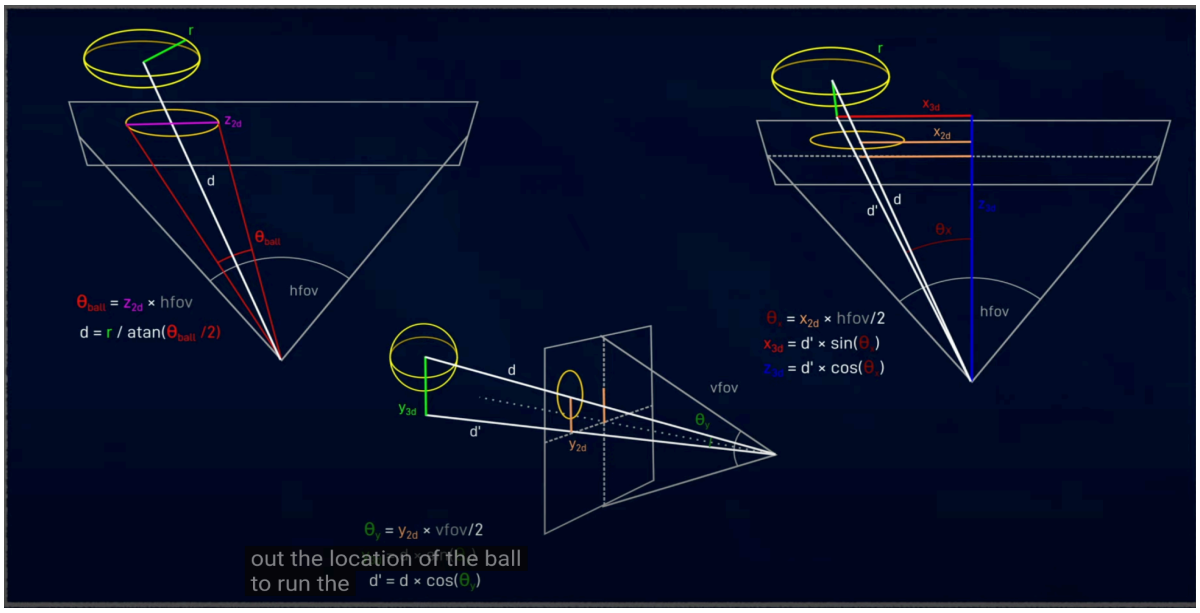


**Figure 17 –** Object tracking coordinate breakdown

# VI. Budget Breakdown

The material listed below highlights all components of the EAP robot that was used to create the finished product. This bill of materials includes components such as the Raspberry Pi 4 and lidar that were provided in class. Some items were also supplied by team members who had them already in their possession and were not needed to be purchased. All items supplied to us via class or teammate are marked with an "*". As broken down in our budget table, our team allowance of $250 was met with a total of $177.84 being spent outside of the provided materials.

| **Type** | **Quantity** | **Price** | **Total** |
|---|---|---|---|
| Raspberry Pi 4* | 1 | $35.00 | $35.00 |
| Arduino Nano | 1 | $20.00 | $20.00 |
| Slamtec RPLIDAR A1M8* | 1 | $99.00 | $99.00 |
| Polulu Motors*& L-Brackets | 2 | $17.78 | $35.56 |
| Hi-Tech Servo | 2 | $13.49 | $26.98 |

| | | | |
|---|---|---|---|
| Motors | | | |
| Aluminum Wheels* | 2 | $7.50 | $15.00 |
| Webcam* | 1 | $50.00 | $50.00 |
| LiPo Battery* | 1 | $0.00 | $0.00 |
| Motor Driver | 1 | $5.42 | $5.42 |
| 5V Regulator | 1 | $4.50 | $4.50 |
| Casters | 2 | $1.88 | $3.76 |
| Wires | 100 | $0.00 | $0.00 |
| Metric M3 Screws and bolts | 1 | $10.00 | $10.00 |
| Acrylic Plates | ⅛”x12”x12”<br>¼”x12”x12” | $12.99 | $26.00 |
| 3D Printed Parts | 6 | $0.00 | $0.00 |
| Total (w/ Provided Parts) | | | $326.84 |
| Total (w/o Provided Parts) | | | **$177.84** |

# VII. Summary and Lessons Learned

**Understanding Software/Hardware Compatibility**

Early in this project, the most troublesome challenge was learning all the various ROS systems and hardware systems we need and which versions of each do and do not function with each other properly. For instance, the RPLidar firmware version impacts which version of the lidar driver node you can use, which is only compatible with certain versions of ROS, which are only compatible with certain versions of Ubuntu, which run only on specific versions of the Raspberry Pi. The robot systems are like a fragile stack of dominos which must be assembled in a very specific way to stand up and if anything is wrong, the entire stack comes falling down. When assembled correctly and with all the right dependencies in place, the system functions robustly and reliably. But, as we learned, if anything is wrong– you have the wrong version of a node, the wrong version of a specific piece of hardware– the entire thing comes crumbling down.

This doesn’t mean you can’t get versions of our system working on new hardware and ROS versions, but doing so necessitates a deep understanding of ROS and how to modify/create your own ROS nodes, drivers, controllers, and hardware interfaces. These are skills nobody in our group yet has, and so we had to spend a significant amount of time discovering which versions of every system we needed in order for our systems to integrate properly. ROS is powerful and exciting to learn, but the price of that power is a large entrance-fee in the form of the skills and knowledge required just to get started.

**Team Communication and Delegation**

As a team we relied heavily on navigation, specifically ROS2 control to systematically integrate servo actuators, object recognition, and object tracking. However, due to the complexity of integrating SLAM and the issues of implementing transforms which correlate the simulation positions of parts of our robot to a physical representation of our robot, we ran into several runtime errors. The lessons learned from our errors led us to believe as a team that we should’ve created tasks that did not depend on one overarching feature within our robot and focus on the overall picture of the project which was to make a robot pet. A lot of the time spent was tackling future parts of navigation. However, small features such as speech recognition and physical aesthetics such as eyes and color changing would've helped us develop something that would've been more fitting to the project. Overall, making independent features would've been a better approach. Additionally, this would help delegate tasks to team members.

Furthermore, collaboration with other teams would have helped progress the software of our robot as well as inspiration. In contrast, due to the overall complexity of the integration of the RPLidar, there weren't many other teams to collaborate with which made it difficult. Regardless of this, we were able to produce a great navigation system despite having areas to improve on and are extremely proud with what was produced at the end. Taking things piece by piece was a big focus but the process probably could have benefited from collaborating with other teams.

# References

[1] Reamon, D. (2024). *Project Draft - DOG or EAP.pdf*. MCEN 4228-5228 Files. https://canvas.colorado.edu/courses/100582/files/74016462?module_item_id=5295702

[2] Joshnewans. "GitHub - Joshnewans/Ball_Tracker: ROS 2 Ball Tracking Software." *GitHub*, github.com/joshnewans/ball_tracker.

[3]Joshnewans. "GitHub - Joshnewans/Articubot_One." *GitHub*, github.com/joshnewans/articubot_one.

# Appendix:

[Link to Robopet ROS Package]
[Link to RPLidar Driver Package]
[Link to Hardware Interface]
[Link to Arduino Motor Controller Program]
[Link to Encoder Motor Datasheet]

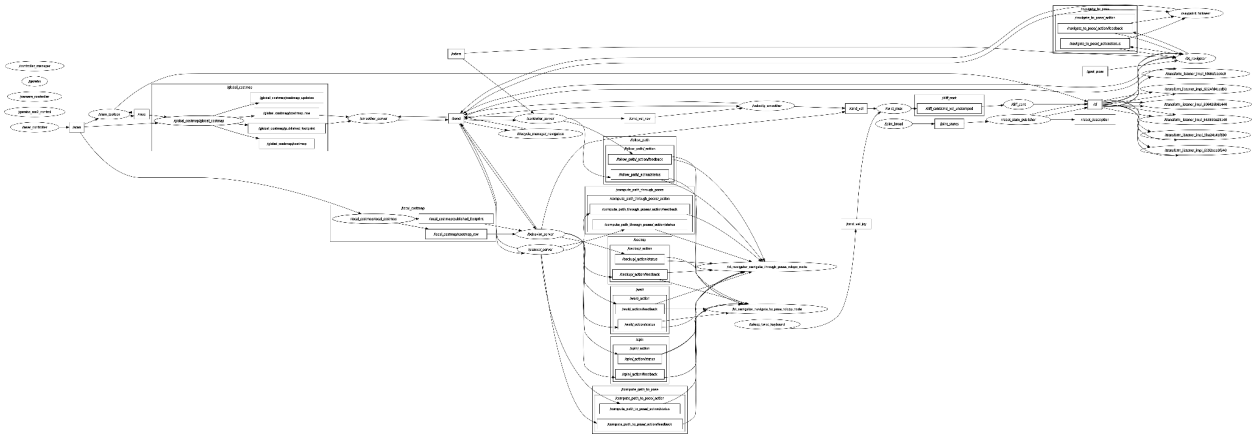

**Figure 18 –** ROS RQT Graph for Simulation

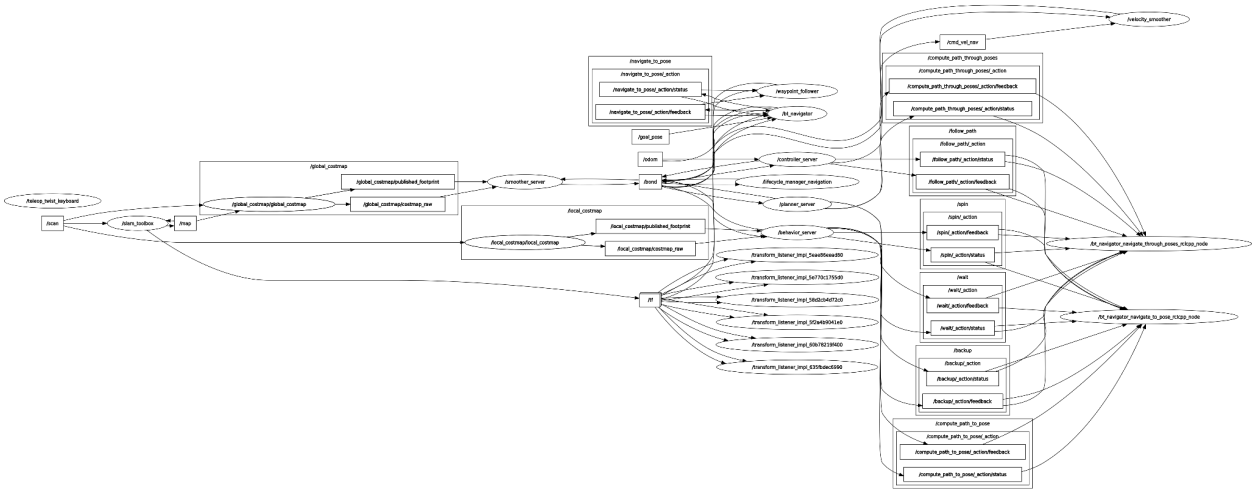

**Figure 19 –** ROS RQT Graph for Real Robot

Note – Apologies for the incomprehensibility of the RQT graphs. This is why we included a simplified node graph. We hope by zooming in or exporting the images these graphs may be readable.