TRASHBOT: Mobile Garbage Collecting Robot

A Major Qualifying Project Report submitted to the faculty of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the degree of Bachelor of Science

Submitted by:

Liliana Loughlin
Robotics Engineering

Cristobal Rincon Rogers
Robotics Engineering, Computer Science

Matthew Sweeney
Robotics Engineering

Yuhan Wu
Robotics Engineering

May 1, 2024

Professor Andre Rosendo, Advisor
Robotics Engineering Department

Professor Fabricio Murai, Co-Advisor
Computer Science Department

Neil Rosenberg, Co-Advisor
Robotics Engineering Department

ABSTRACT

Pollution poses a significant challenge in our modern world, with humans generating vast amounts of plastic waste that endangers the environment and its wildlife. Currently, addressing this issue relies heavily on human interventions, such as clean-up efforts on roads and beaches. Unfortunately, these are typically inconsistent and unreliable. To tackle this problem, the Trashbot Major Qualifying Project at Worcester Polytechnic Institute is developing a prototype robot for the autonomous collection of plastic bottles and aluminum cans. This innovative project aims to create a robot that can navigate WPI's quadrangle independently, detect litter, approach it, collect it, and dispose of it in a designated location. The Trashbot project is a groundbreaking initiative, marking the beginning of a multi-year interdisciplinary endeavor. In the initial phase of the project, the team has achieved several milestones. They have enabled the robot to autonomously navigate using a GPS module and Inertial Measurement Unit, detect trash using a depth camera, and pass raw images to a trained object detection and classification AI model, YOLO. Additionally, the team has designed and fabricated a 4-degree-of-freedom articulated manipulator, implemented a path planning algorithm using a graph state model and installed a LiDAR fixture for future enhancements.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This project was made possible thanks to the help of various people that gave us guidance throughout the MQP journey:

Advisors: Professor Andre Rosendo, Professor Fabricio Murai, Neil Rosenberg

We would like to thank Sparkfun Electronics for supplying our team with the GPS module used in this project.

We would also like to thank Scott Pena and Christopher Chow for helping us calculate the joint trajectories of the initial iteration of the robotic arm and simulate its motion.

1. INTRODUCTION

In response to the need for effective waste management in urban environments, our team embarked on a mission to design and construct a mobile robot capable of autonomously detecting, collecting, and disposing of garbage. Equipped with a robotic arm, the robot was designed to identify trash, approach it, and safely collect it. The system is illustrated below in Figure 1. We successfully developed prototypes of key subsystems, including a four-degree-of-freedom articulated manipulator, a robust deep learning model for detecting aluminum cans, and a navigation algorithm for autonomous GPS navigation. The robot was able to rotate and drive to an object it identified as garbage and retrieve said object with its robotic arm. Despite successfully implementing the navigation algorithm, our robot's navigation performance was inconsistent. This report details our design process, challenges, achievements, and areas for improvement as we strive to enhance the efficiency and reliability of our mobile garbage collection robot.

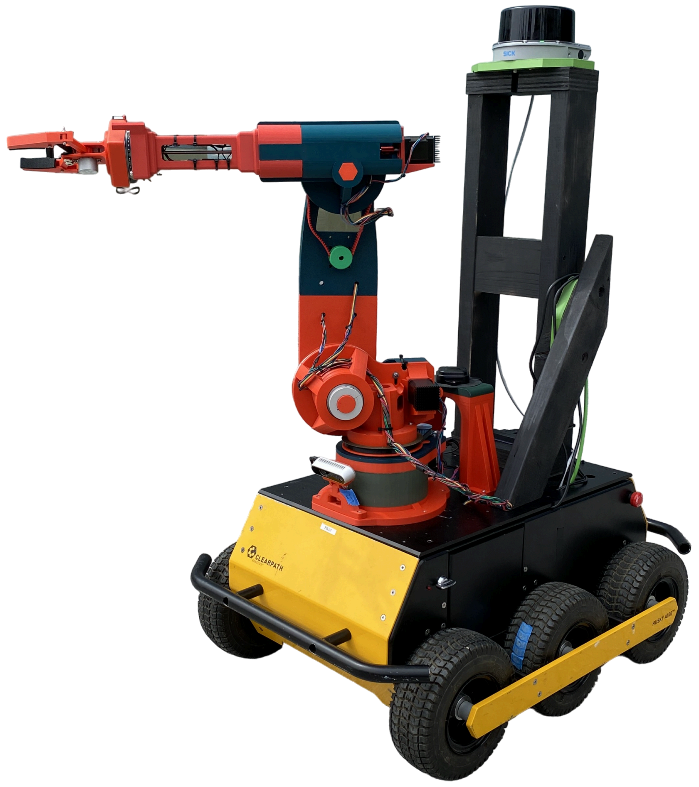


Figure 1: The TRASHBOT.

2. BACKGROUND

The field of autonomous mobile robots for waste collection plays a crucial role in addressing the growing challenges of urban waste management and environmental sustainability. With rapid urbanization and population growth, traditional waste collection methods are becoming increasingly inefficient and environmentally harmful. Autonomous mobile robots offer a promising solution by providing efficient, cost-effective, and environmentally friendly alternatives to manual waste collection.

Our project, focused on the development of a trash-collecting mobile robot with a robot arm, fits within the larger context of the problem area by aiming to improve the efficiency and effectiveness of waste collection processes. By automating the waste collection process, our robot can navigate through urban environments, identify and collect trash, and deposit it in designated locations, reducing the need for manual labor and optimizing waste collection routes.

The field of autonomous mobile robots for waste collection is advancing rapidly, with notable studies such as those by Satav et al. (2023) showcasing the feasibility and effectiveness of autonomous waste collection or sorting robots in various environments. Despite these advancements, significant gaps persist in our understanding of how to scale, enhance robustness, and improve adaptability of these robots across diverse urban settings and waste types.

Key to the development of these robots are the analytical models and system designs that underpin their functionality. These models often integrate sensor technologies for environment perception, path planning algorithms for navigation, and robotic arms for manipulation tasks. Studies by Kulshreshtha et al. (2021) offer valuable insights into the use of grippers, sensors, intelligence models, and manipulation techniques, which can be leveraged and adapted for waste collection robots. The current benchmark in this field is embodied by autonomous robots capable of navigating complex environments, accurately detecting and collecting trash, and dynamically optimizing collection routes in real time. However, these systems still face challenges related to energy efficiency, adaptability to evolving environments, and cost-effectiveness.

In summary, while significant strides have been made in the field of autonomous mobile robots for waste collection, there remain considerable challenges to address. Our project seeks to contribute to this field by developing a robotic system that tackles some of these challenges, thereby pushing the boundaries of autonomous waste collection technology.

3. DESIGN & DEVELOPMENT

Due to the complexity of this project, the design process is discussed in four subsections: an overview of the entire system, robotic arm development, visualization & object detection, and robot navigation.

3.1 System Overview

The core functionalities of the TRASHBOT include navigating around the Quad at WPI, as well as detecting, collecting, and disposing of garbage in the field. The system architecture shown below in Figure 2 discusses its specific functionalities in chronological order.

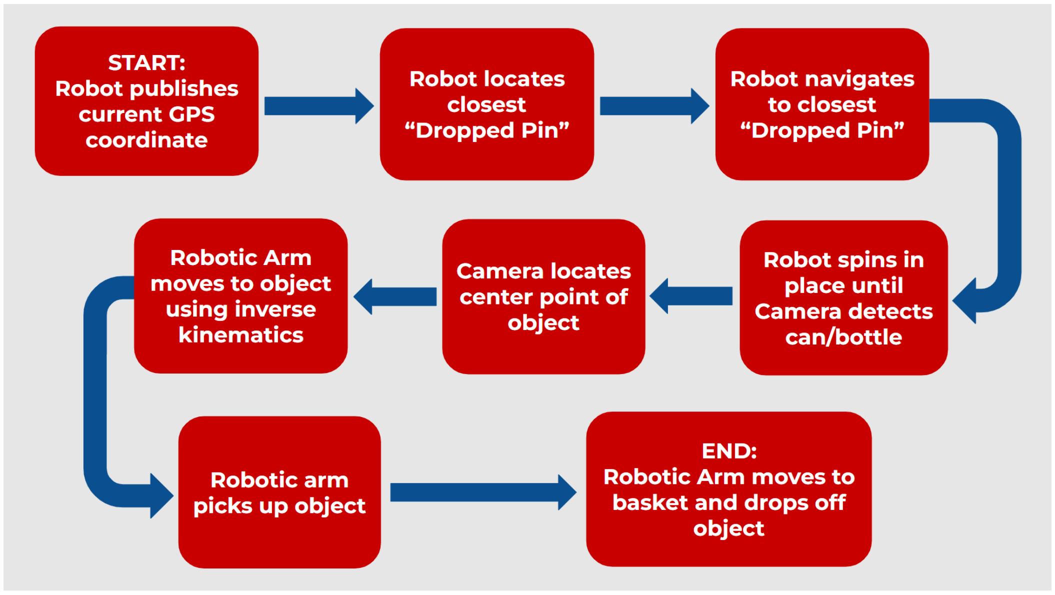

START:
Robot publishes current GPS coordinate
Robot locates closest “Dropped Pin”
Robot navigates to closest “Dropped Pin”
Robot spins in place until Camera detects can/bottle
Camera locates center point of object
Robotic Arm moves to object using inverse kinematics
Robotic arm picks up object
END:
Robotic Arm moves to basket and drops off object

Figure 2: System architecture of the TRASHBOT.

Because our team’s budget was restricted to $1000 between the four of us, we made several cost-effective decisions regarding the system’s overall design. This includes 3D printing the robotic arm, as well as utilizing a recycled Husky A100 and a Nvidia Jetson TX2.

3.1.1 Central Vehicle System

The Clearpath Husky A100, a mobile robot previously used by the Digsafe teams, is a robust system developed by Clearpath Robotics. It features an integrated control and power management system, two-wheel encoders for odometry, a user-accessible fuse panel, and various mounting options for third-party peripherals. The control and power management system includes a control board, a two-channel multi-mode motor driver, a field-swappable battery system, and a CAN network for potential future expansions.

To control the Husky A100, a computing device communicates with the robot using the Horizon Protocol. This communication is facilitated through the USB port in the user space. The Husky A100, as shown below in Figure 3, is a versatile and reliable mobile robot system suitable for a wide range of applications in research, industrial settings, and beyond.

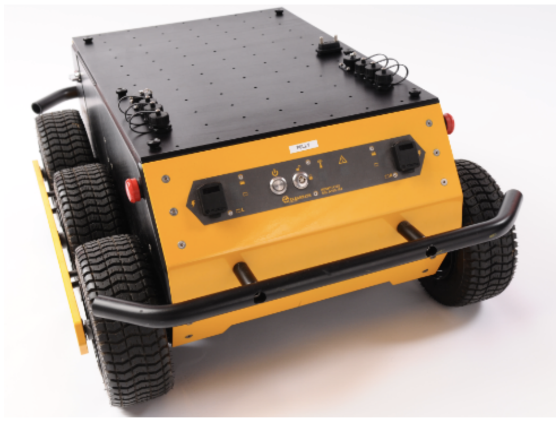


Figure 3: The Husky A100.

3.1.2 Central Computer System

For the robot's central computer, we used the Nvidia Jetson TX2 shown in Figure 4. We chose the TX2 because it has ample processing power and pins for our diverse applications. Additionally, its compatibility with Ubuntu 18.04 and ROS Melodic further aligns with our system requirements.

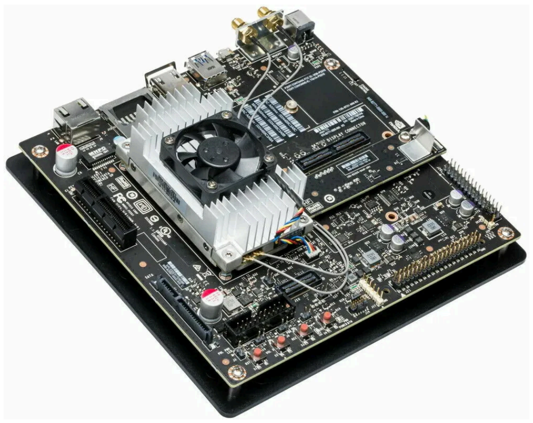


Figure 4: Nvidia Jetson TX2.

The Nvidia Jetson TX2 documentation can be found in Appendices A and B. The J21 connector in Appendix A refers to the pinout section of the TX2. The pins are used to control multiple systems including the arm, limit switches, and the IMU. The J17 connector in APpendix B refers to the serial UART connection interface. Serial devices such as the chassis, camera, and GPS module connect to a USB hub which connects to the serial USB port of the TX2.

3.1.3 ROS Architecture

ROS, or Robot Operating System, is an open-source framework for developing robotic software. It provides a set of libraries and tools that help developers create complex and robust robot applications. ROS is not an actual operating system but rather a middleware layer that runs on top of a traditional operating system (such as Linux) and provides services like hardware abstraction, device drivers, communication between processes, and package management. It is designed to be distributed and modular, allowing developers to create individual software components called nodes that communicate with each other over a network. This modular approach makes it easier to develop, test, and maintain complex robotic systems.

In terms of our project, the NVIDIA Jetson TX2 requires ROS Melodic and Ubuntu 18.04 primarily for compatibility reasons. ROS Melodic is one of the officially supported versions of ROS for Ubuntu 18.04, so utilizing this release of ROS ensures stronger support for Python 3, updated versions of core libraries, and improvements to the build system. Additionally, the Jetson TX2 is designed to perform optimally with Ubuntu 18.04 due to its driver and software compatibility. Both ROS Melodic and Ubuntu 18.04 are Long-Term Support (LTS) versions, ensuring a stable and supported development environment. Furthermore, both ROS Melodic and Ubuntu 18.04 have large and active online communities, which can be beneficial for finding tutorials and sharing knowledge with other developers using similar setups. Overall, using ROS Melodic and Ubuntu 18.04 with the NVIDIA Jetson TX2 provides a reliable environment for developing robotic applications.

3.2 Robotic Arm Design

According to the system architecture discussed in section 3.1, the robotic arm's core functionalities include:

- Moving its end effector to a given point
- Gripping the object located at said point
- Releasing the object at a specified drop point

These actions were achieved by designing and 3D printing several iterations of the system, wiring its actuators to the Jetson TX2, and calculating the arm's inverse kinematics.

3.2.1 CAD Design

To save time and additional expenses, our team decided to develop the project's robotic arm based on smaller scale open-source CAD (Computer-Aided Design) designs online. We determined the arm should have 4 DOF (Degrees of Freedom); including 5 DOF would add unnecessary weight and redundancy, but only incorporating 3 DOF would not allow the arm to reach pieces of garbage in different orientations. We also decided to include limit switches so the arm could "home" itself before it retrieves an object. With these requirements in mind, we identified a CAD design shown below in Figure 5. With its associated Youtube video detailing its development process, we were able to locate its initial Fusion 360 files and modify them accordingly (*3D Printed 6DoF Arduino Robot Arm*, 2019).

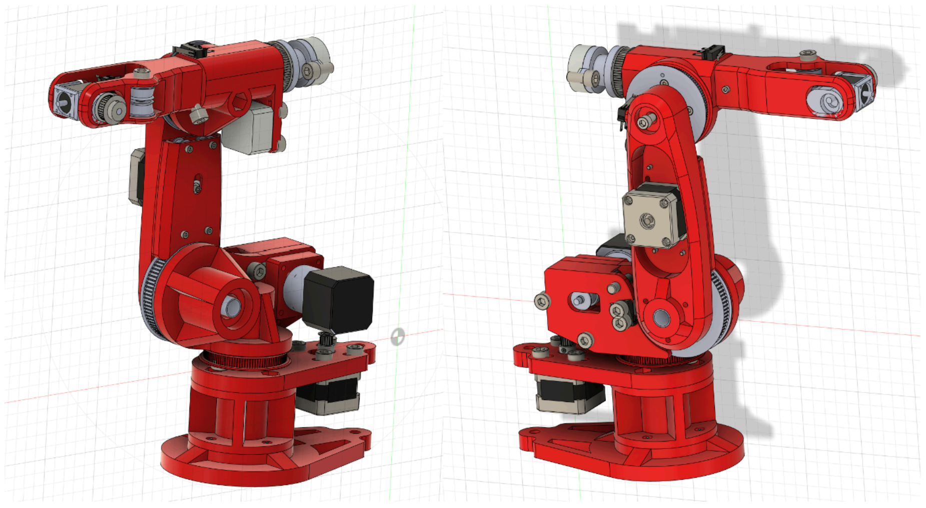


Figure 5: Reference CAD design (*3D Printed 6DoF Arduino Robot Arm*, 2019).

Considering that the arm would be resting on the Husky, its links needed to be long enough so the end effector could reach the garbage on the ground in front of the robot. This involved scaling the reference design by approximately 250%. Due to this drastic size difference, it was necessary to modify several areas of the arm in order to complete our initial CAD design. We split the design into four sections: Base, Link 2, Link 3, and Gripper. The table shown in Appendix C compares the sections from the reference CAD to the sections from our initial design. It also discusses the modifications made and the reasoning behind them.

The robotic arm's final CAD design is shown below in Figure 6. In order to collaborate more fluidly, our team switched from editing our CAD design in Fusion 360 to Onshape. The table shown in Appendix D compares the sections from the initial CAD to the sections from our final design. It also discusses the modifications made and the reasoning behind them.

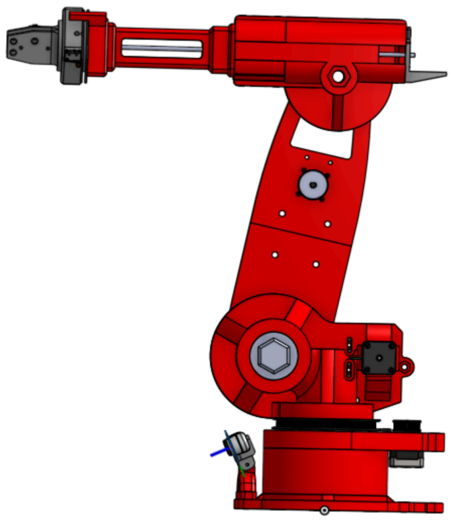


Figure 6: The final CAD design for the robotic arm.

3.2.2 Material Choices

To stay within our team budget, we completely 3D printed the robotic arm to create a light and cost-effective design. All of the linkages and pulleys were initially printed with generic PLA filament, but we reprinted them with Bambu PLA Tough filament. We made this decision because PLA Tough is more likely to deform under stress, rather than crack/break like generic PLA. The arm's gripper is mostly printed from Bambu PLA Tough, but some of its pieces were replaced with Bambu PA6-CF. This ensured that the gripper would not break if the arm experienced failures and made an impact on the ground. Lastly, we concluded that instead of buying several iterations of gearbelts, we could save money and time by 3D printing them. Most of them were printed with TPU filament, as it provides flexibility and high shore hardness. The torque required to control the second joint of the arm is much higher compared to the other joints, so we 3D printed its gear belt with a combination of TPU and PETG filament. As illustrated below in Figure X, the flexible TPU was used for the teeth, and the spine was constructed out of PETG to maintain its shape under stress.

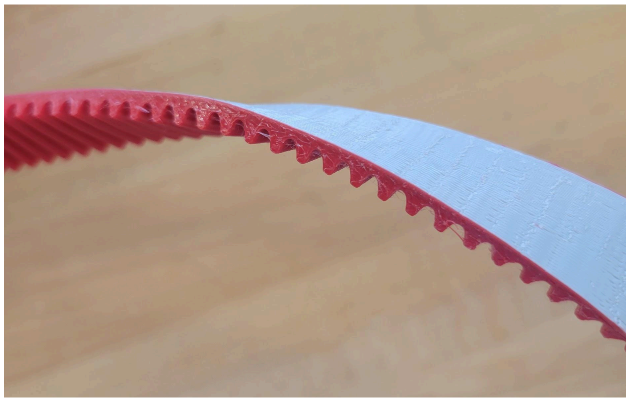


Figure X: Gearbelt for the robotic arm's second joint.

3.2.3 Electrical Design

Motors and Motor Drivers

For the robotic arm system, we used a multitude of stepper motors. We chose the NEMA-17 stepper size because it is extremely common and sites like Stepper Online had several motors of this size with varying features. The motors and their gearboxes were chosen based on static analyses discussed in section 4.1.1. The data sheets for each of these motors can be seen in Appendix E. The motor specifications for the robotic arm's joints are as follows:

- Joint 1: 48 mm motor without a gearbox
- Joint 2: 60 mm motor with a gearbox
- Joint 3: 38 mm motor with a gearbox
- Joint 4: 48 mm motor without a gearbox

We chose to use the DM542T stepper drivers to control the motors. The DM542T drivers provided a smooth signal to the steppers without microstepping. We chose not to micro-step as it could potentially reduce our motor output torque. Unfortunately, these drivers require 5V logic, but the Jetson TX2 only requires 3.3V logic. In order to address this, we needed to convert the logic voltage from the drivers to a tolerable logic voltage for the Jetson TX2.

Logic Voltage Conversion Solutions

We pursued several solutions to the differing logic voltage problem, including utilizing bi-directional level shifter boards by Adafruit. Then, we attempted to create our own unidirectional level shifters to increase the voltage of our system. One of the boards and its corresponding schematic is shown below in Figures 7 and 8.

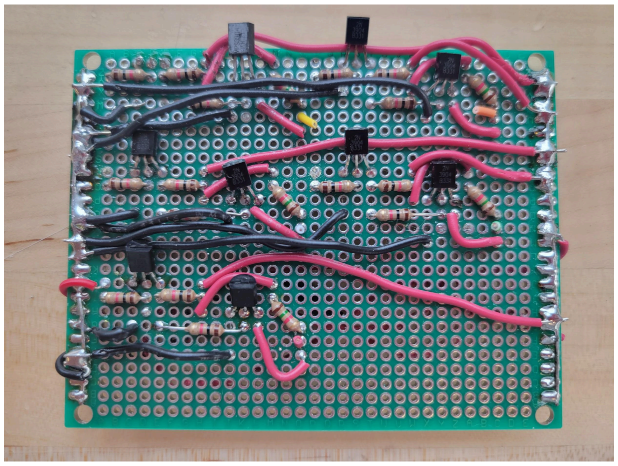


Figure 7: Unidirectional level shifter board.

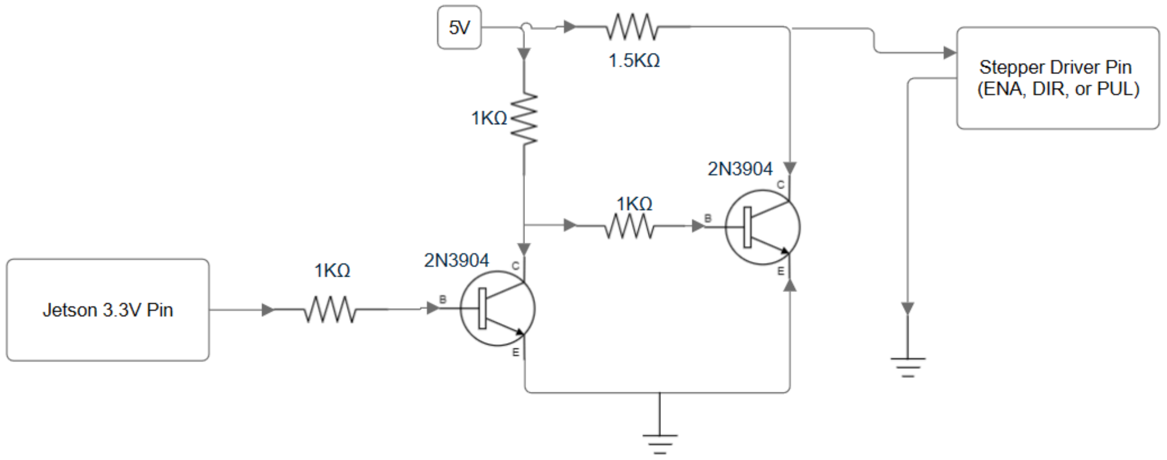


Figure 8: Unidirectional level shifter schematic.

The final circuit we designed consisted of a transistor that allowed us to enable and disable a 5V circuit for each of the inputs to the driver. The transistor was placed in between the negative connection of each input and the ground. This allowed us to deactivate the circuit by cutting off the ground when the base pin of the transistor was low. The base pin was engaged and disengaged by the 3.3V logic from the TX2. The schematic is shown below in Figure 9.

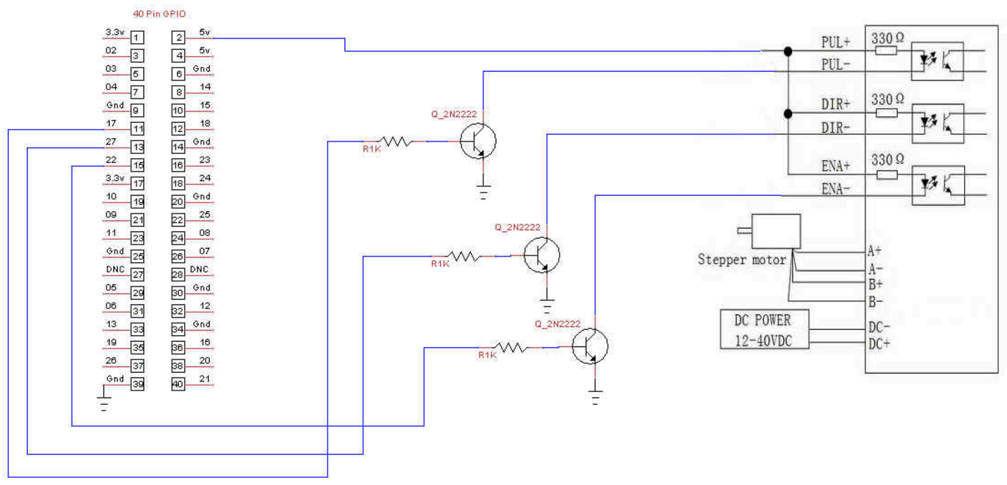


Figure 9: Transistor circuit schematic.

Gripper Circuitry

The gripper utilizes a 28BYJ-48 stepper motor and a ULN2003 driver shown below in Figure 10. The driver is controlled by an ESP32 board, connected to the Jetson through serial.

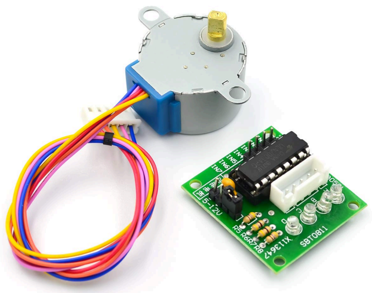


Figure 10: 28BYJ-48 stepper motor and ULN2003 driver.

We chose the 28BYJ-48 because of its small form factor, which allowed us to place it directly on the gripper mechanism. Because the driver was meant to be used with the Arduino Stepper library, it was difficult to wire it directly to the Jetson TX2. We chose to mitigate this problem by including an ESP32 board. We determined that the ESP32 could function in this system because the motion of the gripper requires little computational power. Once we connected the ESP32 to the USB hub on the Jetson TX2, we could communicate between the processors and actuate the gripper.

Electrical Schematic

Figure 11 below is an electrical schematic of the robotic arm. It includes the connections between the Jetson TX2, the joint limit switches, the joint stepper motors, and the joint stepper motor drivers. The full page view of this schematic is shown in Appendix F.

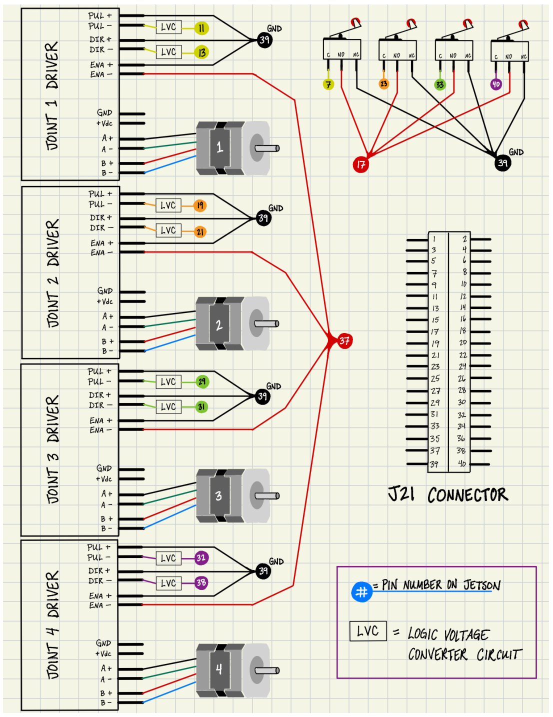


Figure 11: Electrical schematic of the robotic arm.

Motor Control Software

To control the motors, we developed a Python stepper motor library. By default, the library assumes a Counter Clockwise (CCW) positive pulse increment. However, users have the option to invert the motor direction, considering Clockwise (CW) as the positive pulse increment. This feature was implemented to accommodate situations where certain twists were defined opposite to the axis of rotation of a motor. This inversion allowed for alignment with twists and preferred joint axis configurations.

Moreover, our library incorporated considerations for the minimum pulse width necessary to set direction, movement steps, and enable motors as required by our stepper drivers. To control the GPIO pins on the Nvidia Jetson TX2, we utilized the Jetson.GPIO library. However, it is important to note that specific UDEV rules must be configured to allow all users to read and write to the memory-mapped devices when using this library.

Finally, we employed the following equation to map angles to pulses, taking into account the gear reductions for each joint:

$$pulse\ count = round((\Theta\ /\ step\ angle)\ *\ gear\ reduction)$$

This mapping ensured accurate and precise movement control of the robotic arm. We rounded the value since we could only have a discrete number of steps.

3.2.4 Robotic Arm Inverse Kinematics

In order to control the arm's movement based on a given point, we calculated its inverse kinematics. Before completing this, it was necessary to find the transformation matrix from the camera frame to the reference frame on the robotic arm. It was also key to include the additional height difference created by the Husky in the translation portion of the matrix. This transformation is visually demonstrated in Figure 12.

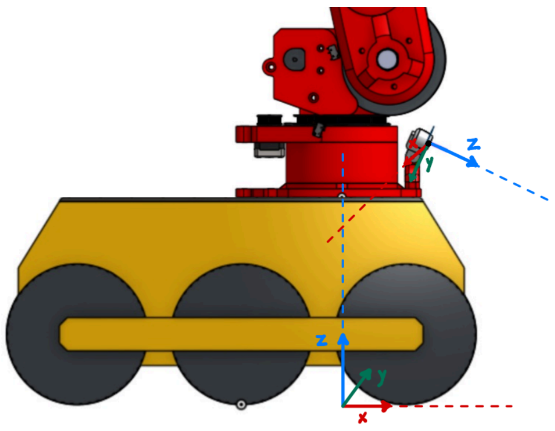


Figure 12: The camera frame and the reference frame on the robotic arm.

The camera transformation matrix is as follows:

```
np.array([[ 0,-0.4067, 0.9135, 0.122497],
          [-1, 0,      0,      0.032445],
          [ 0,-0.9135,-0.4067, 0.416466],
          [ 0, 0,      0,      1]])
```

Initial Solution

Our team initially decided to calculate the arm’s inverse kinematics using the Newton-Raphson method. We solved for the home configuration and the joint twists shown below in Figure 13. We intended to use functions derived from a code library accompanying the textbook *Modern Robotics: Mechanics, Planning, and Control* (Lynch & Park, 2017).

$$S_1 = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$S_2 = \begin{bmatrix} 0 & 1 & 0 & -0.53477 & 0 & 0.07 \end{bmatrix}$$

$$S_3 = \begin{bmatrix} 0 & 1 & 0 & -0.9054 & 0 & 0.07 \end{bmatrix}$$

$$S_4 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0.935512 & -0.0196 \end{bmatrix}$$

$$M = \begin{bmatrix} 1 & 0 & 0 & 0.55912576 \\ 0 & 1 & 0 & 0.0196 \\ 0 & 0 & 1 & 0.935512 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 13: Joint twists and home configuration of the robotic arm.

Unfortunately, we could not use the IKinSpace() function from the code library because it needed the desired orientation of the end effector as an input. Because we only had the desired position of the end effector, we decided to solve the inverse kinematics with a different approach.

Successful Solution

Upon further exploration on different approaches to solve inverse kinematics, we discovered that the geometric decoupling method is the most successful for our project. By separating the problem into smaller sub-problems, geometric decoupling allowed us to efficiently compute the joint angles needed to reach the desired end effector position. Since joint 4 controls the rotation of the gripper, it relates to the orientation of the detected object. Therefore, we only calculated angles θ1, θ2, θ3 for their respective joints based on the object’s position. The procedure of this solution is addressed below.

1) Set the arm to its home configuration. Assign labels to the joints and their corresponding links, as shown in Figure 14.

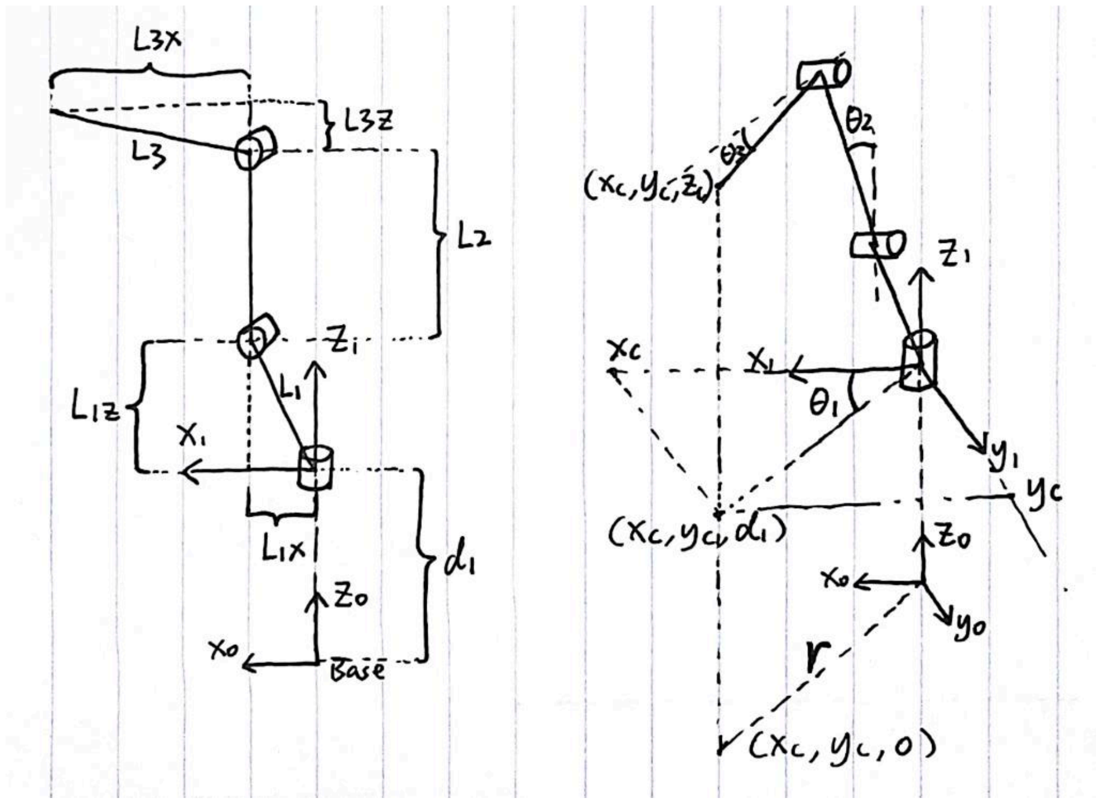


Figure 14: Home configuration and joint label of the robotic arm.

2) Calculate θ1:
Joint 1 is calculated using atan2 using the following equations:

$$r = \sqrt{(xc^2 + yc^2)}$$

$$cos(\theta 1) = \frac{xc}{r} = D$$

$$\theta 1 = atan2(\pm\sqrt{1 - D^2}, D)$$

It returns a positive angle and a negative angle with the same absolute value. We determine which value is correct based on the detected object's Y coordinate: if $yc < 0$, then θ1 is positive; otherwise, θ1 is negative.

3) Calculate θ2:

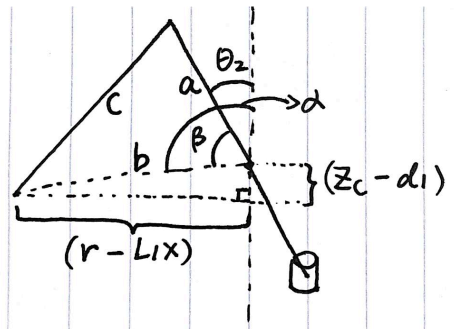


Figure 15: Joint 2 illustration.

As shown in Figure 15 above, joint 2 is calculated using the Law of Cosines and the following equations:

$$\alpha = tan(\frac{r - L1x}{Zc - d1})$$

$$a = L2$$

$$b = \sqrt{(r - L1)^2 + (Zc - d1)^2}$$

$$c = \sqrt{(L3x)^2 + (L3z)^2}$$

$$\beta = arccos(\frac{a^2 + b^2 - c^2}{2ab}) \quad \text{[Law of Cosines]}$$

$$\theta 2 = \beta - \alpha$$

If *Zc* is below Z0, $Zc - d1$ will return a negative value.

4) Calculate θ3:

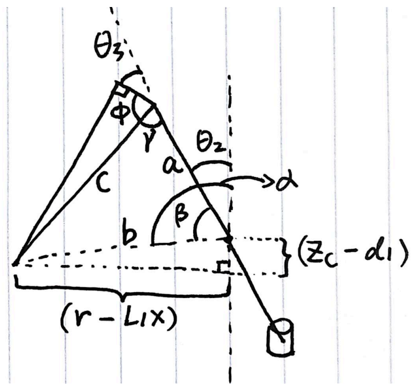


Figure 16: Joint 3 illustration.

As shown in Figure 16 above, joint 3 is calculated using atan2, the Law of Cosines, and the following equations:

$$\gamma = arccos(\frac{a^2+b^2-c^2}{2ab}) \text{ [Law of Cosines]}$$

$$\phi = atan2(L3x, L3z)$$

$$\theta 3 = \pi - \gamma - \phi$$

After we successfully calculated the joint angles needed to reach a given point, we utilized the JointTrajectory() function from the Modern Robotics package. This function allowed us to set the number of points in the discrete representation of the trajectory, as well as the time scaling method. JointTrajectory() successfully returned a trajectory as an N x n matrix, where each row is an n-vector of joint variables at an instant in time.

3.3 Visualization & Object Detection

In this section, we discuss how we've incorporated advanced visualization and object detection methods into our project. By using the YOLOv5 model with a depth camera, we are able to accurately detect objects and extract their 3D coordinates. The depth camera we selected utilizes Stereo Vision technology, which offers improved performance in various lighting conditions, and is not affected by the sunlight when performing outdoors. The integration of these technologies supports our goal to develop a system that can handle complex tasks in changing environments.

3.3.1 Choice of Camera

Consumer-grade depth cameras have found widespread use in robotics, unmanned vehicles, and deep-learning-based intelligent systems. Originally developed for home entertainment and gaming, cameras like Microsoft's Kinect V1 have become popular in robotics and computer vision due to their lightweight design, affordability, high frame rates, and reasonable resolution (Heinemann et al., 2022). While the original Kinect (2010) used structured light (SL) technology, other technologies like Time-of-Flight (ToF) and stereo vision (SV) have become prevalent in-depth camera systems.

ToF cameras emit pulses of infrared light and measure the time it takes for the light to return from the target to the sensor to estimate depth. This technology is used in cameras like Microsoft's Azure Kinect and Kinect V2. The uncertainty of depth measurements typically scales linearly for ToF, providing better measurements for longer ranges (Heinemann et al., 2022). On the other hand, SL cameras, first introduced in the first generation of Kinect, emit an irregular pattern of infrared light, with depth information calculated from the distortion of this projected pattern. SV technology, which uses triangulation to determine depth from two images captured from different viewpoints, is also popular. Active SV systems project an additional infrared pattern on the scene to create artificial features for depth calculation (Heinemann et al., 2022). While SL and ToF technologies struggle under direct light and are thus not reliable in outdoor environments, SV systems are known for their higher reliability under varying conditions in robotics and computer vision applications, despite only sometimes being as precise as SL.

For our project, which required a camera robust and adaptive to varying light conditions for outdoor use, we chose to focus on systems with Stereo Vision. Heinemann's research indicated that the Intel RealSense D455 had the best performance and documentation, with the Intel RealSense D435 being the second-best option and the Oak-D Pro as the fourth-best option. Our team sought to balance our priorities between documentation and performance. Therefore, our main contenders were the Oak-D S2, Oak-D Pro, and the Intel RealSense D435f.

We also conducted an additional comparison between the Oak-D Pro and the RealSense D435 provided by Luxonis. Figure 17 shows the reference image used for comparison, while Figures 18 and 19 show the performance using inactive laser dot projection (infrared light) and active laser dot projection (infrared light). We found the output of both cameras satisfactory; however, once again, the documentation of the RealSense cameras stood out as superior.

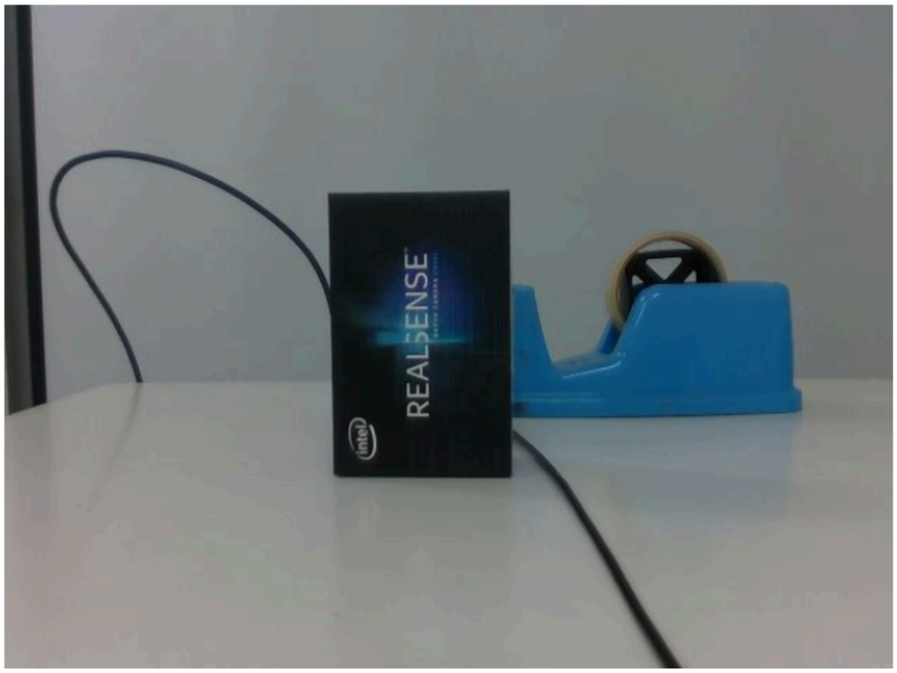


Figure 17: Reference image for comparing Intel RealSense D435i and the Oak-D Pro (*RealSenseTM Comparison*, 2023).

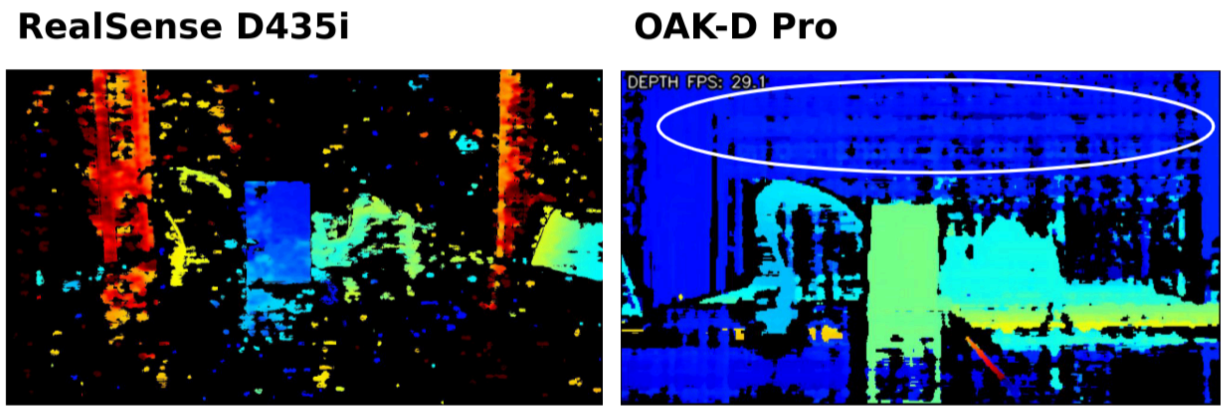


Figure 18: Laser dot projector disabled: Passive Stereo (*RealSenseTM Comparison*, 2023).

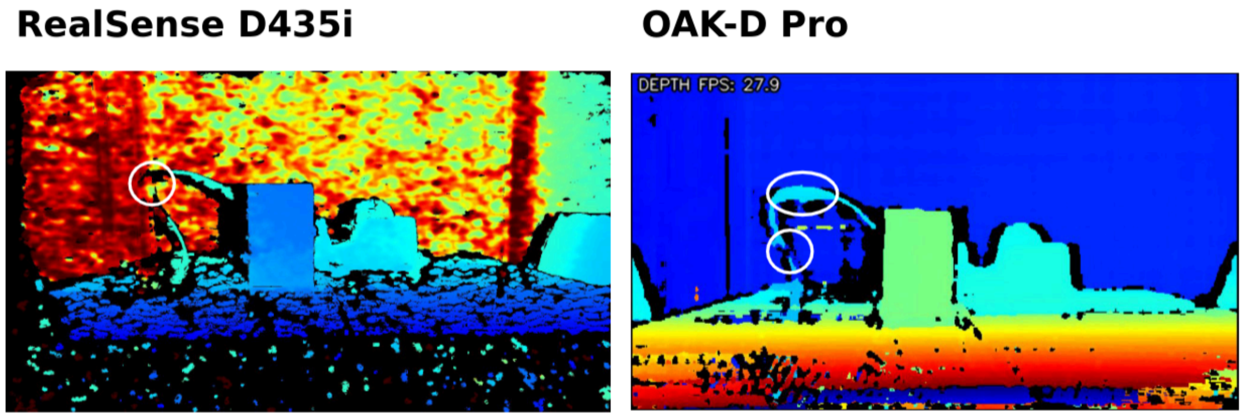


Figure 19: Laser dot projector enabled: Active Stereo (*RealSenseTM Comparison*, 2023).

As a result, we chose the Intel Realsense 435f, shown in Figure 20, which incorporates a 750nm near-infrared filter for simple out-of-the-box integration.

Figure 20: Intel RealSense D435f.

3.3.2 YOLO Training & Data Collection → Generalize YOLO

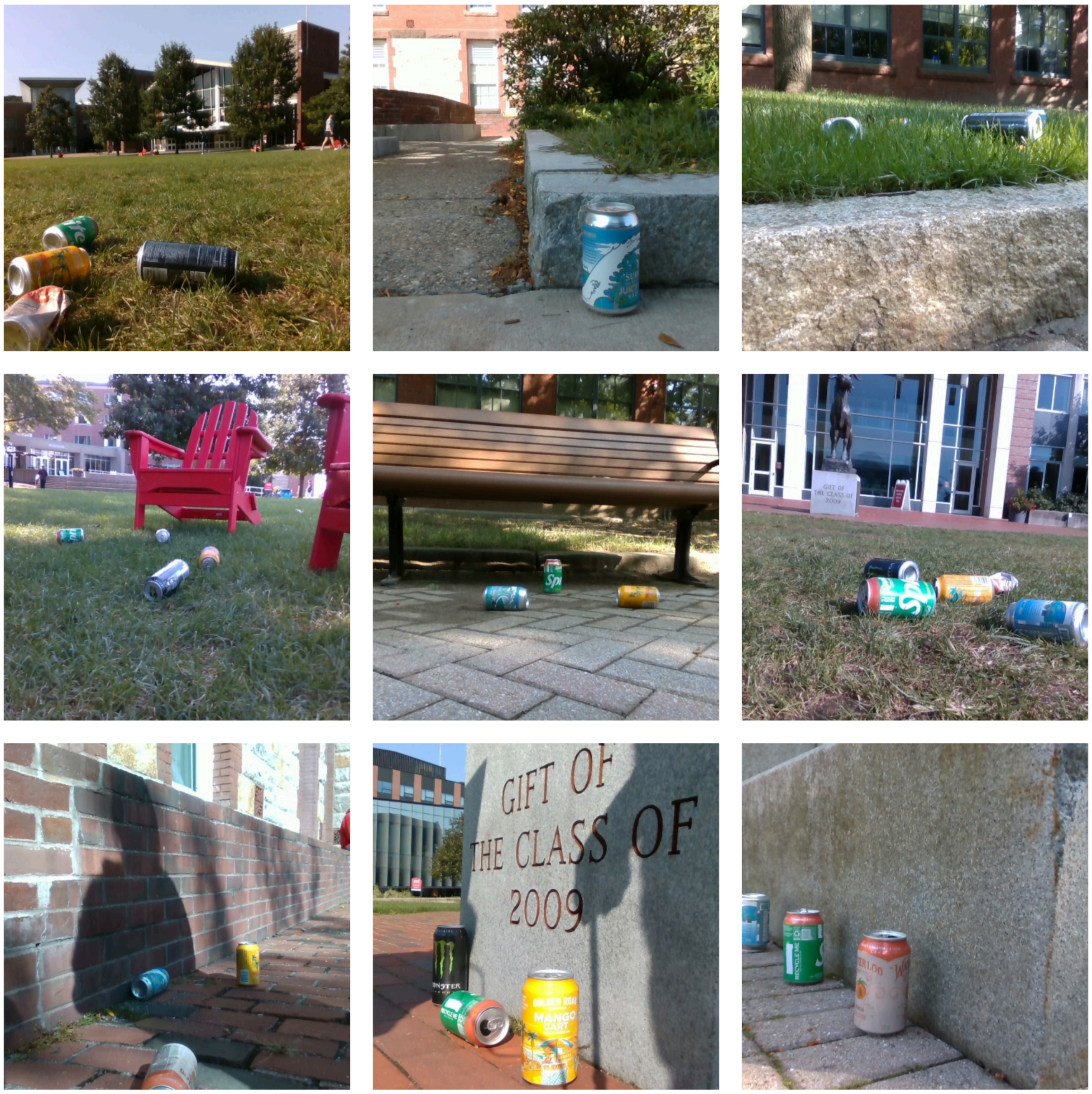


Figure 21: Custom dataset collected for training YOLO.

To train our deep learning model, YOLO, we gathered a custom dataset specifically tailored to our needs (Figure 21). YOLO, which stands for "You Only Look Once," is a state-of-the-art object detection system that extracts contextual information from images to precisely locate objects within them. To ensure our model's generalization capability, we focused on collecting images of aluminum cans in various orientations and placed on different surfaces. We omitted plastic transparent PET bottles since YOLO had difficulty locating the images in the scene. Initially, we captured images using our smartphones. However, recognizing the importance of training our model with images taken from the robot's mounted camera height, we switched to using images captured from this perspective. This adjustment aimed to account for any distortions introduced by the camera lens, ensuring the model would be more accurate in real-world scenarios. Our dataset was divided into two subsets: 80% for training the model and 20% for testing its performance. This division allows us to evaluate how well the model generalizes to unseen data, helping us gauge its effectiveness and reliability in practical applications.

3.3.3 Object Detection

Because we developed our system within the ROS environment, we utilized a ROS wrapper for Intel RealSense devices. The realsense2-camera ROS package was used to manage camera streams and to extract depth information efficiently.

While testing the Intel RealSense SDK on our laptop, we noticed that the depth frame and the RGB color frame did not match; the camera had a larger FOV(field of view) on its depth frame than the color frame. This is because the 2D color picture is generated through a separate RGB module, and it is using two stereo visions to calculate the depth frame. In order to ensure that each pixel in the color image corresponds to a direct depth point, we needed to align the depth image to the color image by mapping the depth to the smaller color FOC size. The exact overlay of the depth on color images is essential for getting the accurate distance between the detected object and the camera. We used realsense2-camera ROS package to align depth images to rgb images and publish the aligned depth frame to the rostopic "/camera/camera/aligned_depth_to_color/image_raw". Furthermore, we generated a point cloud database on the aligned depth image, which enabled us to complete future point cloud processing.

Once we received a stable stream of ROS images through the depth camera, it was necessary to perform object detection through the YOLO v5 algorithm. For our purposes, the following software dependencies needed to be addressed:

- Framework Requirements: YOLO v5 requires PyTorch and Torchvision for its operation. Based on the limitations imposed by our current JetPack version, the most advanced compatible versions are PyTorch 1.8.0 and Torchvision 0.9.0. Due to the limitation of PyTorch and Torchvision versions, we chose the release version v5.0 of YOLO v5.
- Python Version Compatibility: Our system initially had Python 3.7 installed; however, the v5.0 release version of YOLO v5 runs most stable on Python 3.6. To resolve this, we utilized a Conda environment that allows the installation of the required Python version without affecting the system-wide installation.

In order to combine YOLO v5 with the ROS environment, we utilized a package called ROS-CV_bridge to convert between ROS image messages and OpenCV image messages. This conversion enabled the application of the YOLO v5 algorithm for image processing and object detection.

Using the YOLO v5 algorithm, a bounding box was placed around each detected target object in the image. From this, the X and Y coordinates of the center of the bounding box were extracted, representing the central point of the object within the image frame. Once we extracted the 2D coordinate of the object in the image frame, we proceeded to use camera intrinsic parameters to map the 2D coordinates to the 3D coordinates based on the camera's coordinate frame.

3.3.4 Coordinate Extraction

Using the depth data and the $(x,\ y)$ coordinates from the image frame, we calculated the 3D coordinates of the object in the camera coordinate system. This transformation relies on the camera's intrinsic parameters, which include the focal lengths $(fx,\ fy)$ and the optical center $(cx,\ cy)$ of the camera.

$$Z\ =\ depth\ data$$

$$X\ =\ \frac{(x - cx) \times Z}{fx}$$

$$Y\ =\ \frac{(y - cy) \times Z}{fy}$$

The equations above project 2D image points onto 3D space based on the recorded depth and the intrinsic parameters of the camera. Once we retrieved the 3D coordinate of the detected object, we published it as a PoseStamped message data through ROS publisher. This allowed for the further usage of the position data in the inverse kinematics calculation and enabled the future addition of orientation data of the detected object.

3.3.5 RANSAC Point Cloud Processing

For our orientation calculation of this project, we utilized the RANSAC (random sample consensus) method to perform a cylinder fit for the camera's point cloud data. However, due to limitations in the current segmentation method, which lacks robustness and responsiveness, we didn't include RANSAC in our final iteration. That being said, extracting the object's orientation remains a significant area of interest for potential future teams to enhance or redevelop our initial attempts with RANSAC segmentation. The progress we made regarding this method is discussed below.

To reduce computational load and enhance the effectiveness of the segmentation process, we implemented a filter based on the distance between the point and the XYZ coordinate of the detected object. We only kept the points that were within a 20 cm radius from the detected object's XYZ coordinates. Then, we computed the surface normals for each point in the filtered point cloud, which were used in the subsequent plane and cylindrical segmentation process. Utilizing the RANSAC algorithm, a plane model—the floor and ground—was identified and segmented. The inliers represented the planar surface, while the points that did not belong to the plane were set as outliers. In our case, we wanted to perform a further cylindrical segmentation for the outliers to retrieve the orientation of the detected aluminum can.

The result of the cylindrical segmentation returned a XYZ coordinate of a point on the centerline of the cylinder, and a direction vector along the centerline of the cylinder in the camera frame. Since the orientation aspect of the detected object is only associated with the turning angle of the gripper, we only needed to retrieve the angle between the Z axis of the camera and the direction vector. We extracted the x and z aspect of the direction vector, then used atan2 to calculate the angle that could be used in future inverse kinematics calculations.

3.4 Husky Kinematics

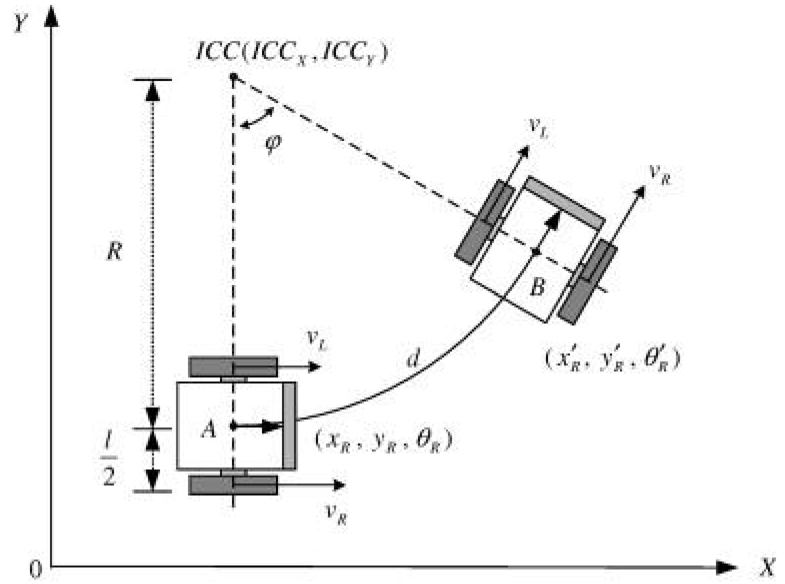


Figure 22: Differential drive kinematic model (Han, Choi, & Lee, 2008).

Our odometry was based on the odometry model for a differential drive robot. A differential drive robot is a type of mobile robot that moves by rotating its wheels at different speeds. This differential motion allows the robot to change its direction and move forward or backward. The key feature of a differential drive robot is its ability to turn in place by rotating its wheels in opposite directions. The kinematic model for a differential drive robot is illustrated above in Figure 22. The robot's position and orientation are described by its x and y coordinates and its heading angle theta. Its linear velocity and angular velocity determine the robot's motion. These velocities are related to the wheel speeds of the robot's left and right wheels.

3.4.1 Husky Modifications

Released in 2010, the Husky A100 is the predecessor of the Husky A200, making it outdated and lacking current software packages. Consequently, we had to rely on the packages available on Clearpath Robotics' GitHub for the Husky A200. We modified several of these packages because the Husky A100 is incredibly different from the Husky A200. For example, the A200 uses a form of skid steering where all four wheels are powered. Comparatively, the A100 uses a 6-wheel drive train chassis. The inner wheels on each side of the A100 are slightly lower than the outer wheels, causing the robot to operate more similarly to a differential drive robot than a skid steering robot. Additionally, other than the physical size differences of the wheels, the two Huskies have entirely different encoders and speed controllers.

To address these discrepancies , we created new packages for the Husky A100's motion. We developed a ROS package named "motion," consisting of two nodes: "odometry" and "kinematics." The odometry node updates the robot's state using data from the encoders and IMU, while the kinematics node handles rotation and linear movement. Despite these changes, challenges remained.

3.4.2 IMU

To improve our heading information, we purchased an IMU, shown in Figure 23 below. Specifically, we opted for the 9DOF Adafruit BNO055, which offers a yaw angle accuracy of approximately 1 degree. The IMU communicates with the Jetson TX2 using I2C. We utilized a ROS library that creates a publisher for this sensor, which allowed us to publish heading data. The Odometry node subscribes to this topic and updates the robot’s heading information within the limits of [-π, 0] to [0, π). The Github repository for this ROS library can be found here: https://github.com/BytesRobotics/bno055

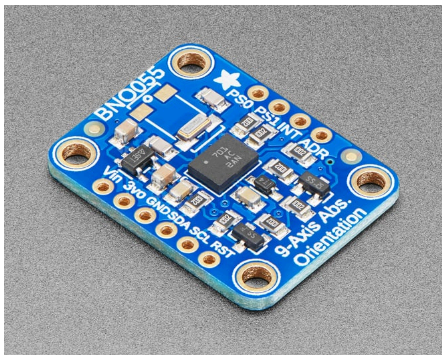


Figure 23: The BNO055 9-DOF IMU.

Connecting the IMU to the Jetson TX2

Table 1 below displays the pin connections between the IMU and the Jetson TX2.

Table 1: Connecting between IMU and Jetson TX2.

IMU Pins	Their Corresponding Jetson TX2 Pins
GND	6
3.3V	1
SCL	28
SDA	27

3.4.3 Movement Function

For the Husky's movement function, we employed a straightforward approach known as turn-drive. This method involves the robot first rotating in the direction of the desired point of travel, then driving the corresponding distance to reach that point.

3.5 Navigation

Our mission was to design and construct a mobile robot that can detect, collect, and dispose of garbage on WPI's Quad. To navigate autonomously in an outdoor environment, we favored using a GPS module for localization and navigation.

3.5.1 Choice of GPS Module

GPS (Global Positioning System) plays a crucial role in outdoor navigation for mobile robots due to its ability to provide accurate and real-time location information. This information is essential for the robot to know its position on Earth's surface, allowing it to plan its path effectively and navigate to its destination. GPS also enables the robot to avoid obstacles and hazardous areas by providing precise location data. Additionally, GPS data can be used to create maps of the robot's environment, aiding in navigation and exploration tasks. Overall, GPS enhances the robustness and efficiency of outdoor navigation for mobile robots, making it an indispensable tool for autonomous operation in outdoor environments.

For outdoor navigation, our team decided between using the NEO-6M or the ZED-F9P. The NEO-6M and the ZED-F9P are both GPS modules, but they differ in terms of features and capabilities. The NEO-6M is a basic and cost-effective GPS module suitable for simple navigation applications. It provides standard GPS functionalities, including accurate positioning and time information. One of its main advantages is its low cost, making it a popular choice for applications where budget is a concern. However, the NEO-6M has limitations in terms of accuracy and update rate compared to more advanced GPS modules. It may struggle in environments with poor satellite visibility, such as urban canyons or dense foliage, leading to reduced performance in such conditions. On the other hand, the ZED-F9P is a high-precision GPS module known for its exceptional accuracy and reliability. It supports multiple satellite systems, including GPS, GLONASS, Galileo, and BeiDou, which allows for faster and more reliable positioning, especially in challenging environments. The ZED-F9P also offers features such as RTK (Real-Time Kinematic) support, further enhancing its accuracy and making it suitable for applications requiring precise positionings. However, the main drawback of the ZED-F9P is its higher cost compared to the NEO-6M, which may be prohibitive for some projects.

Our team favored the ZED-F9P (Figure 24) to achieve accurate positioning, but it was not in our allotted budget. To combat this, we got creative and emailed several companies who could potentially sponsor our project. We received a response from Sparkfun Electronics offering to send a variety of its products, including navigation modules. As a result, our team acquired the ZED-F9P from SparkFun Electronics without charge.

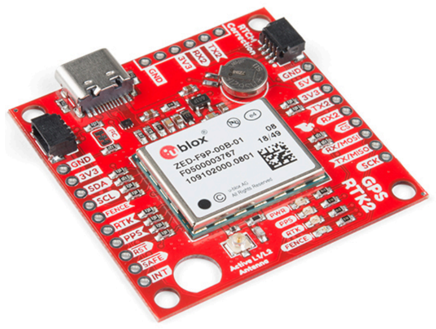


Figure 24: The ZED-F9P from Sparkfun Electronics.

3.5.2 RTK Correction

Real-time Kinematics (RTK) is a GPS technique that enhances the precision of position data derived from satellite-based positioning systems. By comparing the signals received from our GPS unit to that of a fixed base station, RTK corrects for factors that can degrade the GPS signals, such as atmospheric disturbances and satellite orbit errors. This correction significantly reduces positional errors, enabling our system to achieve an accuracy of approximately 3 cm. To implement this high-precision positioning, we subscribed to the Massachusetts CORS (MACORS) network. MACORS provides RTK correction data from its fixed base stations, which is essential for achieving the enhanced accuracy needed for our application. The difference between GPS data before and after RTK correction implementation is displayed below in Figure 25.

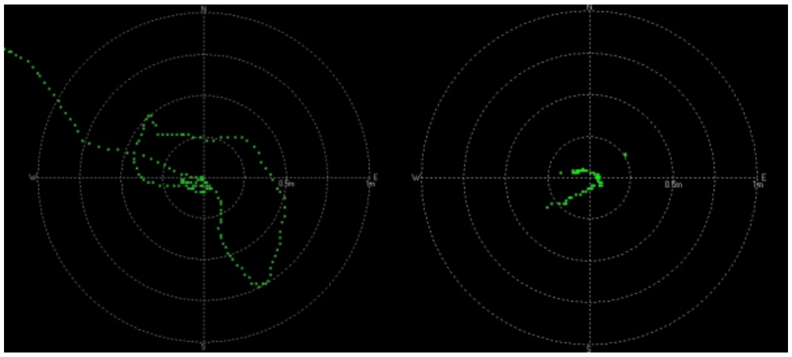


Figure 25: GPS data before (left) and after (right) the implementation of RTK correction (*What is GPS RTK?*).

3.5.3 Dropped Pins

Our team used smartphones to collect GPS coordinates around the quad, marking locations with Google Maps' dropped pins. We recorded each point's latitude and longitude, storing them in a CSV file. The GPS points are illustrated in Figure 26 below.

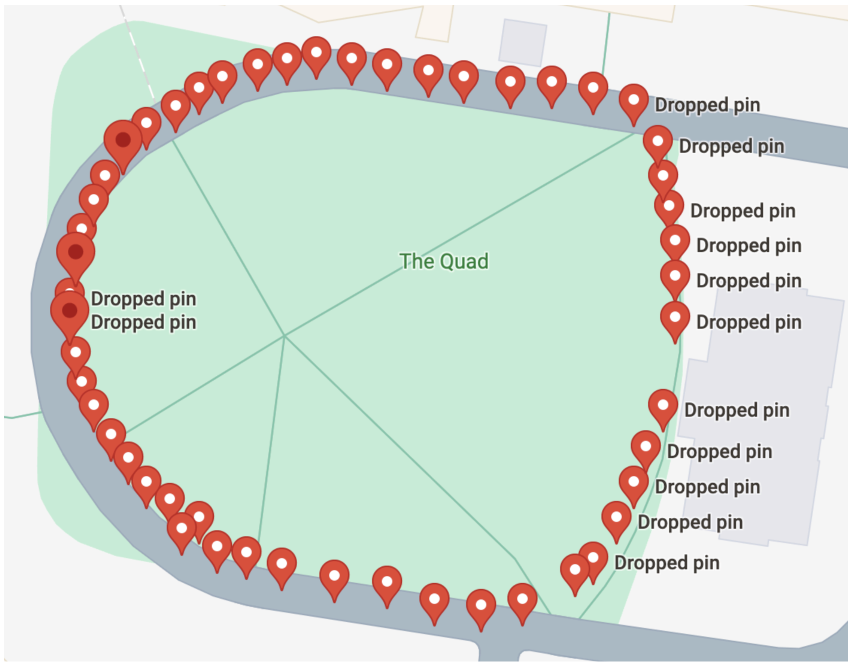


Figure 26: GPS points (Latitude, Longitude) around the quad for navigation.

The accuracy of dropped pins in Google Maps can vary based on several factors, such as the strength of the device's GPS signal, the environment (e.g., urban canyons or open areas), and the quality of the GPS receiver. Typically, under ideal conditions, dropped pins can be accurate to within a few meters. However, in challenging environments with limited GPS visibility, accuracy may decrease, leading to slightly less precise pin placements. Overall, dropped pins in Google Maps offer a reliable estimation of location, though they may not always be exact, particularly in areas with poor GPS reception. To improve our navigation accuracy, we supplemented our data collection with GPS module readings. Due to a lack of resources, we updated a small number of points.

3.5.4 Algorithm Discussion

We developed a navigation algorithm that utilizes geometric formulas and triangular proofs. The procedure of this solution is addressed below.

1. Set a fixed point (F) inside of the WPI Quad.
2. Drive the robot 1 meter directly east to reach a point (Z) with a purely longitudinal offset.
3. The robot determines the closest "Dropped Pin" (Y) from point (Z).
4. Use the Haversine formula, referenced in Appendix G, to calculate the following distances illustrated in Figure 27:
    - The fixed point (F) to the offset point (Z)
    - The offset point (Z) to the desired point (Y)
    - The fixed point (F) to the desired point (Y)

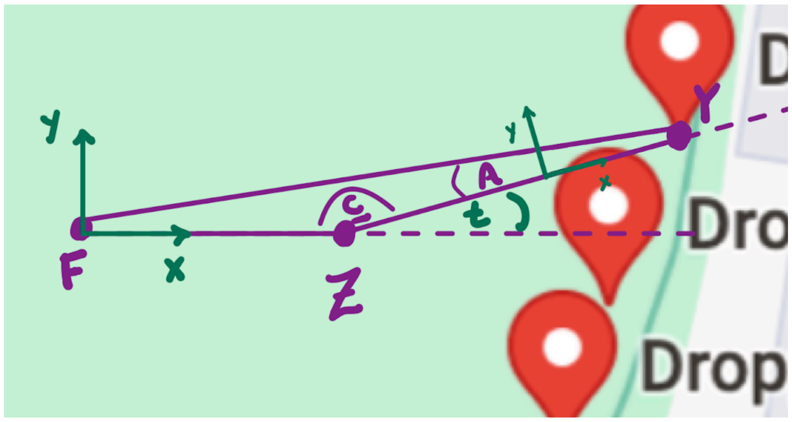
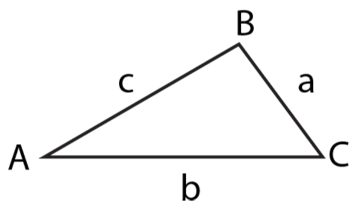


Figure 27: Initial triangular solution for the geometric algorithm.

5. These distances create a purple SSS triangle. Calculate angle A and angle C using the Law of Cosines formula shown below:

$$\boxed{\begin{array}{l} a^2=b^2+c^2 - 2bc\cos A \\ b^2=a^2+c^2 - 2ac\cos B \\ c^2=a^2+b^2 - 2ab\cos C \end{array}}$$



6. Calculate the turning angle: t = 180 (degrees) - angle C
7. The robot turns t degrees and drives straight until it reaches point (Y).

8. The robot begins to rotate in place.
    a. If the camera recognizes an object on the ground as a piece of garbage, the robot stops rotating and navigates to the object.
    b. If the camera does not recognize an object, the robot makes a complete 360 degree rotation in place.
9. (Continuation of 8b) The robot determines the desired point (E).
10. Use the Haversine formula to calculate the following distances illustrated in Figure 28:
    - a: The fixed point (F) to the current point (Y)
    - b: The current point (Z) to the desired point (Y)
    - c: The fixed point (F) to the desired point (Y)

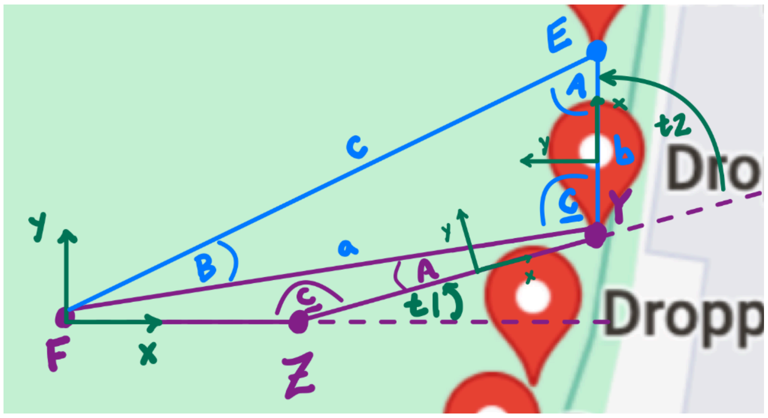


Figure 28: General triangular solution for the geometric algorithm.

11. These distances create a blue SSS triangle. Calculate angle A and angle C using the Law of Cosines formula mentioned above.
12. Calculate the new turning angle: t = 180 (degrees) - current angle C - previous angle A
13. Repeat steps 7-12 until the camera recognizes an object as a piece of garbage or the robot traverses every "Dropped Pin" on the Quad.

4. SYSTEM TESTING & VALIDATION

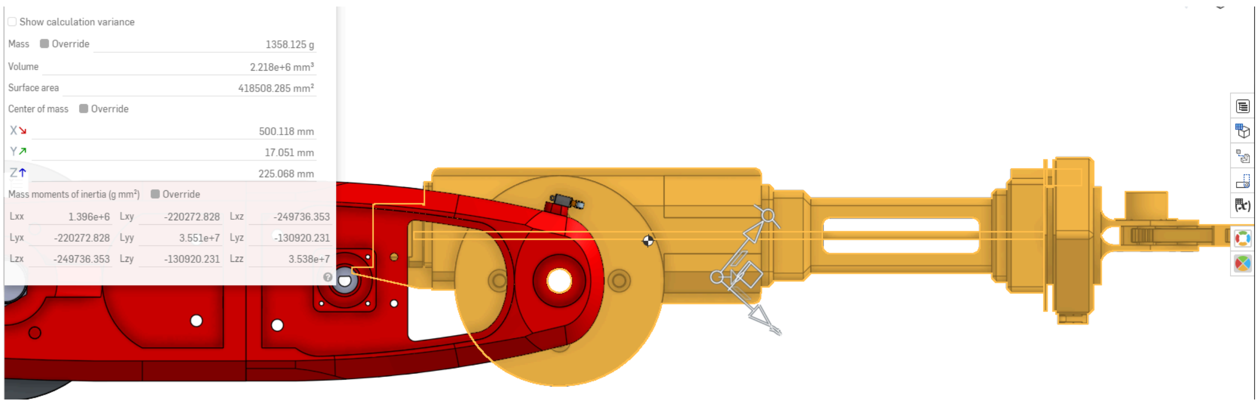
Testing every aspect of our designs mentioned in the previous section was fundamental to the success of this project. This involved executing successive trials on the robotic arm to ensure that its end effector reached a given point, testing the accuracy of the measurements retrieved from the visualization model, and validating the robot navigation algorithms by driving the robot around the Quad.

4.1 Robotic Arm Testing

Once our team had a complete design of the robotic arm constructed, we ran various tests on the system to ensure that the arm can approach, grip, relocate, and drop an object at given points. These tests include static analyses on the motors controlling the joints, electrical assessments on level shifters, and repeatedly executing the entire system.

4.1.1 Static Analyses

In order to ensure that each motor is strong enough to control its respective link, we performed static analyses on all four links. The first and fourth link had a fairly limited load imposed on them, as gravity had no effect on them. Because of this, we were able to use the basic 48mm NEMA 17's for these links. Links 2 and 3 had much more significant loads on them due to gravity, so we needed to perform calculations to find proper motors. Figures 29 and 30 below contain the COM of the mass on each of the two motors.



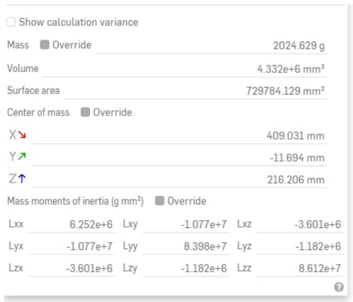
Show calculation variance

Mass (Override)	1358.125 g
Volume	2.218e+6 mm³
Surface area	418508.285 mm²
Center of mass (Override)	
X	500.118 mm
Y	17.051 mm
Z	225.068 mm

Mass moments of inertia (g mm²) (Override)

Lxx	1.396e+6	Lxy	-220272.828	Lxz	-249736.353
Lyx	-220272.828	Lyy	3.551e+7	Lyz	-130920.231
Lzx	-249736.353	Lzy	-130920.231	Lzz	3.538e+7

Figure 29: COM of the mass on the motor actuating link 3.



☐ Show calculation variance

Mass	▨ Override	2024.629 g			
Volume		4.332e+6 mm³			
Surface area		729784.129 mm²			
Center of mass	▨ Override				
X		409.031 mm			
Y		-11.694 mm			
Z		216.206 mm			
Mass moments of inertia (g mm²)	▨ Override				
Lxx	6.252e+6	Lxy	-1.077e+7	Lxz	-3.601e+6
Lyx	-1.077e+7	Lyy	8.398e+7	Lyz	-1.182e+6
Lzx	-3.601e+6	Lzy	-1.182e+6	Lzz	8.612e+7

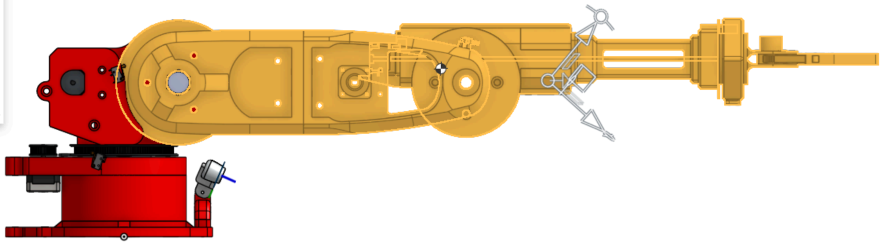


Figure 30: COM of the mass on the motor actuating link 2.

The calculations to determine the torque requirement and provided torque for each motor is displayed below. The motor used for link 3 is 38mm with a gearbox and the motor used for link 2 is 60 mm with a gearbox.

Torque Requirement Calculations for Motor Actuating Link 3:

$m \ = \ 1.3581247\, kg$

$a \ = \ 9.81\, m/s$

$r \ = 0.05947534\, m$

$F \ = \ ma$

$T \ = \ Fr$

$T \ = 0.79240204657\, N/m$

Torque Provided from Motor Actuating Link 3:

$T\ without\ Gear\ Ratios \ = \ 0.39\, N/m$

$Ratio\ of\ Gearbox \ = \ 5.18:1$

$Ratio\ of\ Pulleys \ = \ 5:1$

$Torque\ with\ Gearing\ Ratios \ = \ 10.101\, N/m$

The torque provided from the motor actuating link 3 is approximately 12.75 times more than the torque required to lift the arm. This proves the arm can account for its own weight as it retrieves objects.

Torque Requirement Calcu for Motor Actuating Link 2:

$m \ = \ 2.02462927\, kg$

$a \ = \ 9.81\, m/s$

$r \ = 0.33903086\, m$

$F \ = \ ma$

$T = Fr$

$T = 6.7336997834\ N/m$

Torque Provided from Motor Actuating Link 2 with Gear Ratio:

$T\ without\ gear\ ratios = 0.65\ N/m$

$Ratio\ of\ Gearbox = 5.18:1$

$Ratio\ of\ Pulleys = 5:1$

$Torque\ with\ Gearing\ Ratios = 16.835\ N/m$

The torque provided is at least 2.5 times more than the torque necessary to lift the arm. This, again, allows the arm to account for its own weight as it retrieves objects.

4.1.2 Electrical Testing

As previously mentioned in section 3.2.3, we attempted several solutions to convert the logic voltage from the drivers to a tolerable logic voltage for the Jetson TX2. After wiring each of our solutions into the robotic arm's electrical system, we repeatedly ran every joint motor to determine whether they were receiving enough power to actuate. When we reached testing failures, we utilized an oscilloscope to determine the voltage being received from the Jetson TX2.

4.1.3 Inverse Kinematics Testing

To determine whether the robotic arm was able to reach a given point, we simply engaged in iterative system testing. First, we placed a can at a random point in the robotic arm's workspace. The camera recognized the can and retrieved its X,Y,Z world coordinates. The coordinates were multiplied by the transformation matrix discussed in section 3.2.6, then inputted into the geometric decoupling method. Once the method outputted the joint angles needed to reach this point, we commanded the motors to actuate. We performed this process repeatedly to address slight offsets that occur during real-life use of the arm.

4.2 Visualization Testing

In this section, we focus on testing the visualization capabilities of our system to ensure that the object detection and coordinate extraction functionalities are accurate and reliable. We aimed to validate the integration of the YOLOv5 model with our depth camera setup, assessing how well the system can identify and locate objects.

4.2.1 Inference Testing Validating YOLO

We conducted inference testing to validate the performance of the YOLO model. This involved running the trained model on a test dataset to evaluate its detection capabilities. The test dataset is 20% of our collected data, so our overall confidence value can be described as the mean value of the confidence value of all detected cans in our test dataset.

4.2.2 Accuracy Measurement of Extracted Point

To evaluate the precision and accuracy of our detection algorithm, we positioned the target object at a fixed point and continuously ran the algorithm for 30 seconds. The precision is described by the standard deviation of the extracted point. Then, we measured the object's position with a measuring tape and compared these measurements to the positions detected by the algorithm. This process was repeated three times, and the accuracy is described by the mean of the difference between measured value and the output of the detection and point extraction algorithm.

4.3 Kinematic Testing

To validate our differential kinematic model implementation, we conducted three tests. Firstly, we marked a meter in distance using painter's tape and noted the Husky's starting position on the tape. Then, we commanded the robot to drive a specific distance and compared the actual distance traveled to the desired distance. This test was repeated for distances of 1m, 0.5m, and 0.25m, with multiple trials conducted for each distance to ensure consistency. Secondly, we tested the rotation capabilities of the robot. Using a protractor, we marked angles relative to the robot's starting heading and commanded the robot to rotate to these angles. We tested angles of π/4, π/2, -π/4, -π/2, and π, running three trials for each angle to validate the accuracy of our model in predicting rotational motion. The final test was a combination of the first and second tests: commanding the robot to rotate and then drive a certain distance. These tests were crucial to confirm that the Husky's encoders were accurately calculating the robot's heading and position. We were particularly concerned about the accuracy of the encoders and conducted these tests to ensure that our kinematic model was correctly interpreting the encoder data.

4.4 Navigation Testing

To validate our navigation algorithm, we conducted a crucial test at the Gompei statue on the Quad. We initiated several nodes to commence navigation, including the IMU, Odometry, Kinematics, GPS, and Navigator nodes. The Navigator node assumes responsibility for all navigation tasks, including issuing commands to the Husky's kinematics node. It accesses the CSV file mentioned earlier to generate a map and subscribes to the ROS topic /gps_coordinate to update the node's current position for further calculations.

In our test, the Husky rotated to the specified angle as defined by our algorithm (refer to section 3.5.3). We repeated this test multiple times to assess its accuracy. Our evaluation focused on whether the Husky rotated to the correct heading and traveled the expected distance. To ensure consistency, we started each test from a relatively similar point to determine the trajectory.

5. RESULTS & DISCUSSION

The results of the project's subsystems – object visualization, robotic arm joint trajectories, robot kinematics and robot navigation – are detailed below. The entire system in action can be seen here: https://youtu.be/9_974heO8I0. The github repository can be found here: https://github.com/terracris/trashbot.

5.1 Visualization Results

The subsection 5.1.1 includes the result for object recognition from the testing dataset as well as our inference testing result when running the object recognition algorithm. The result for 3d position extraction compared with our measured distance are detailed in the 5.1.2 subsection.

5.1.1 Object Recognition Results

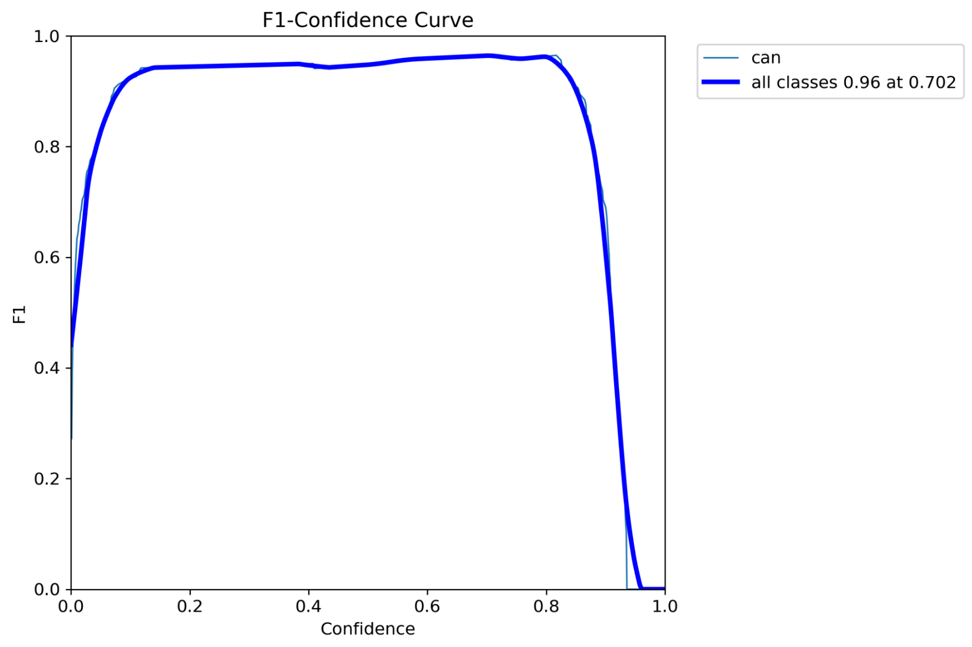


Figure 31: F1-Confidence curve of our trained weight.

Based on the feedback from our object recognition training, on average, the model has an 96% precision at a confidence threshold of 0.702 for all classes (see Figure 31). Furthermore, in our evaluation of the YOLOv5 model during inference testing, we achieved a high confidence level of 0.90, indicating strong reliability in the model's predictions. This indicated that the model consistently identified and classified objects with great certainty. However, we noticed that when testing indoors, the object detection algorithm constantly detected the wheel of a cart at a can (see Figure 32), with confidence level > 0.85. We concluded that this was because we collected all of our training data for object detection in an outdoor environment, thus it could not handle a complicated indoor background.

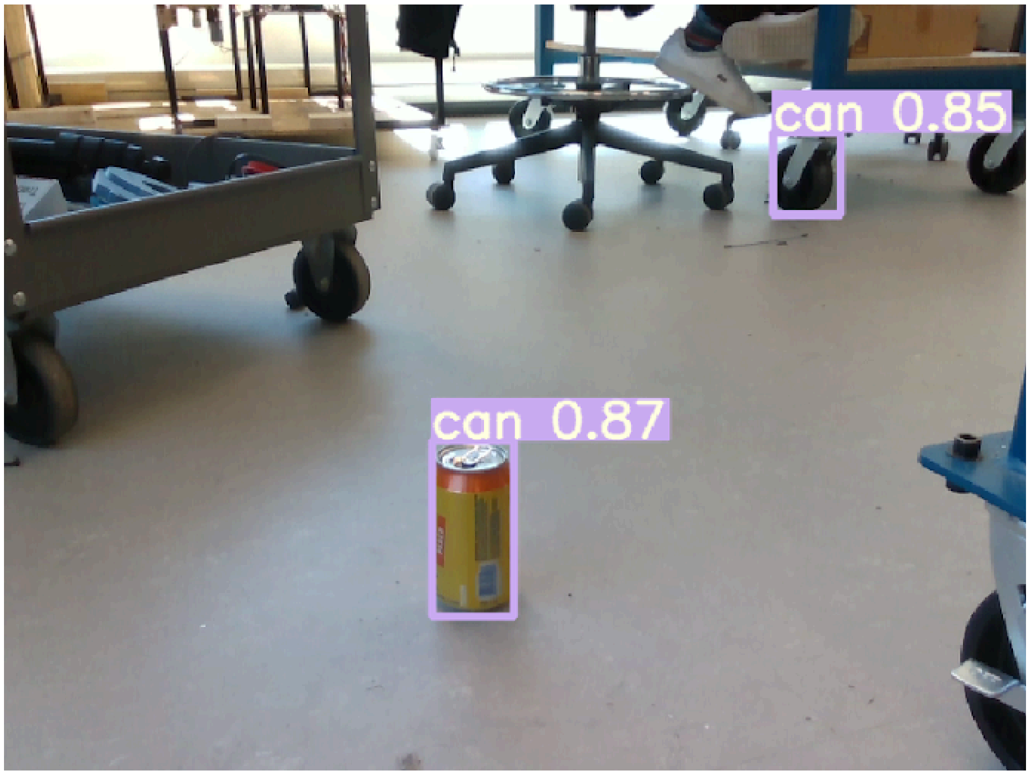


Figure 32: Camera detects the wheel as a can.

5.1.2 Point Extraction Results

Furthermore, our point extraction algorithm was precise and accurate, with errors less than 1 cm in all XYZ directions. However, it is important to note that since we used a measuring tape to retrieve the distance between the detected object and the camera, there was a potential for measurement errors to occur.

Table 2 displays the recorded values for two test positions, the X Y Z values are measured from the base position of the robot. The test images received for positions 1 and 2 from the camera can be seen in Figures 33 and 34, respectively.

Table 2: Point extraction X Y Z values.

	Point Extraction Result for Position 1	Measured Value for Position 1	Point Extraction Result for Position 2	Measured Value for Position 2
X value	0.6184 m	0.623 m	0.5689	0.574 m
Y value	-0.1031 m	-0.112 m	0.2579	0.251 m
Z value	0.0502 m	0.055 m	0.0523	0.055 m

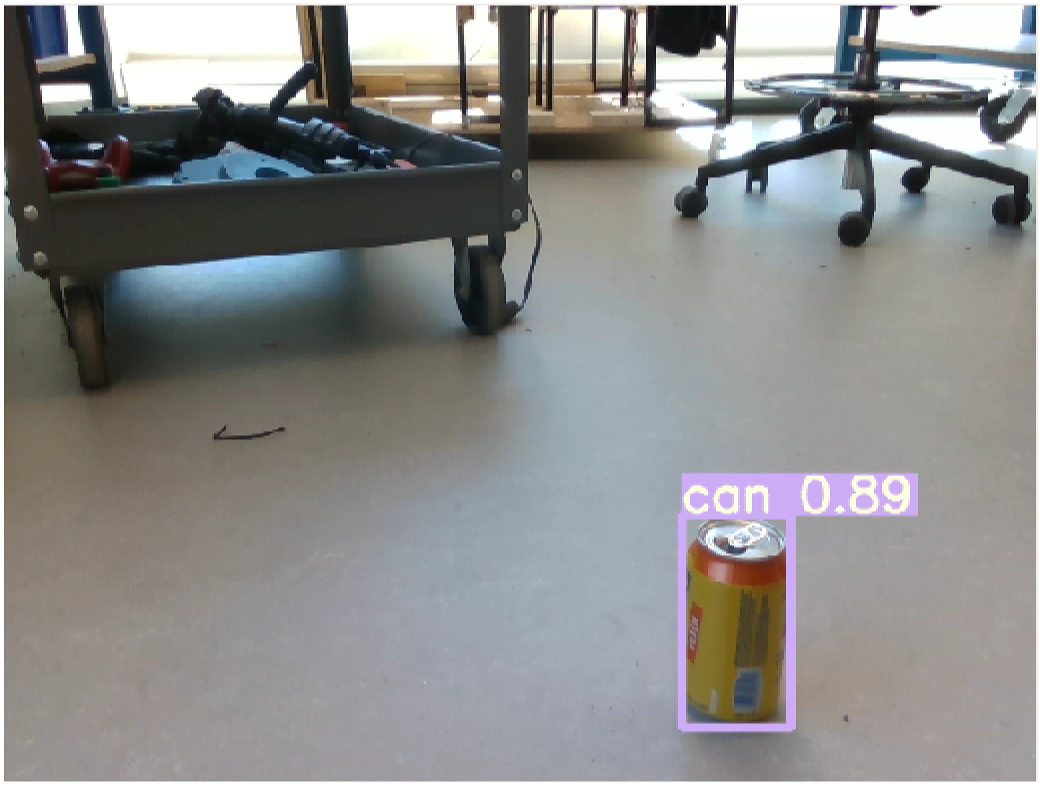


Figure 33: Camera feed for test position 1.

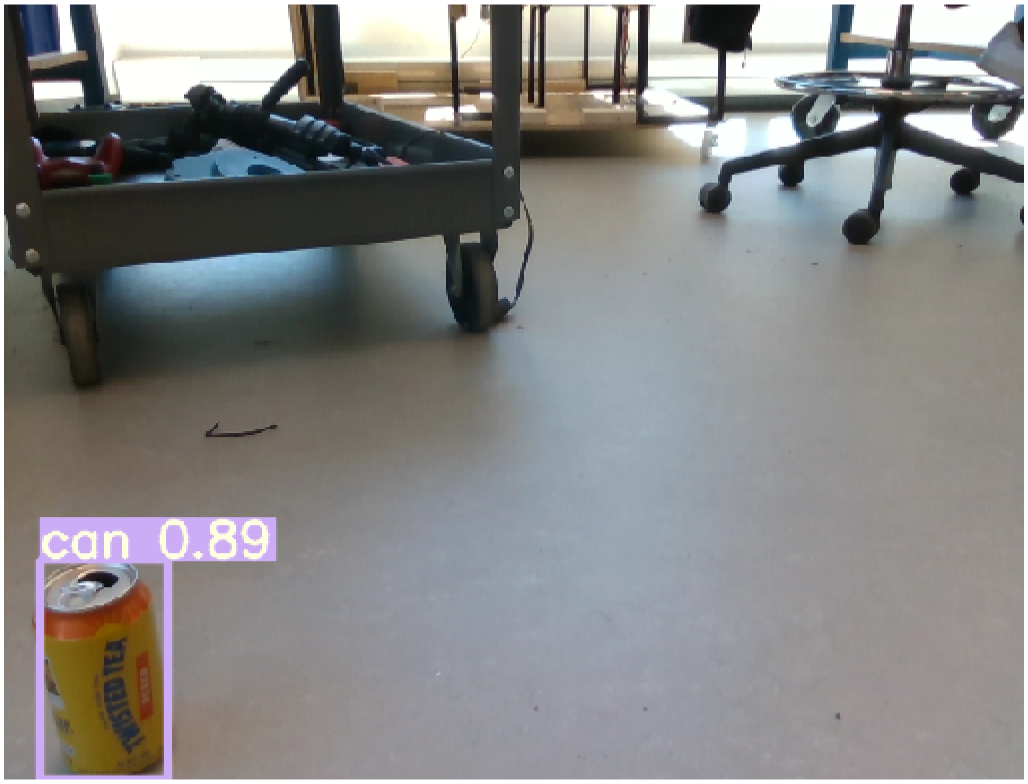


Figure 34: Camera feed for test position 2.

5.2 Robotic Arm Results

Ultimately, the robotic arm was able to successfully navigate to a given point, grip an object, relocate it, and release it at a drop-off point. Figure 31 below exemplifies the operational capabilities of the robotic arm.

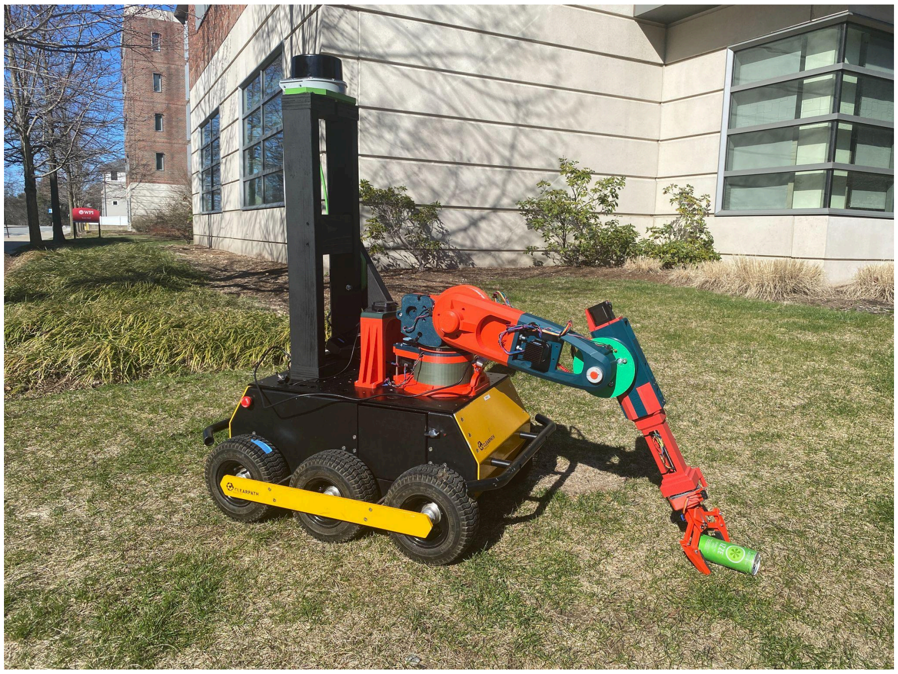


Figure 35: The robotic arm in operation.

5.2.1 Motor & Workspace Results

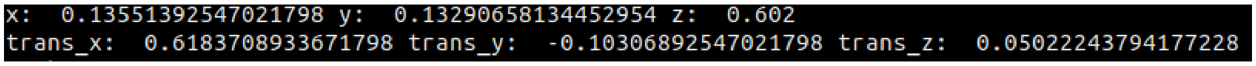
After extensive static analyses and testing on the motors and gears belts, the arm was able to hold its own weight when it was fully extended outward. We defined a realistic workspace for the arm by giving each joint angular limits based on the arm's home configuration. This was to ensure that the arm did not collide with the GPS antenna mount or the LiDAR mount on the Husky. The following Table 3 displays the negative and positive limit angles for each joint on the arm.

Table 3: Joint limit angles.

Joint Number	Negative Limit Angle (Degrees)	Positive Limit Angle (Degrees)
1	-90	60
2	-10	115
3	-75	75
4	-40	90

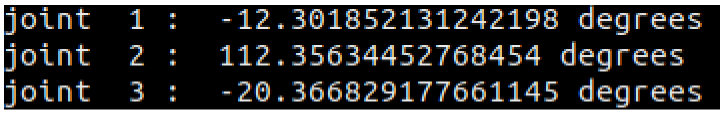
5.2.2 Joint Trajectory Calculation Results

The results of our inverse kinematics testing are illustrated in the following figures. Figure 36 displays the point received from the camera in meters. This point is then transformed using the matrix defined in section 3.2.4, resulting in the coordinates trans_x, trans_y, and trans_z in meters, which represent the position defined by the base of the robot arm. Subsequently, these coordinates are input into our inverse kinematics solution, also outlined in section 3.2.4, yielding the angles shown in Figure 37. To enhance clarity, the joint angles have been converted to degrees for the figures; however, it is important to note that our trajectory planning function requires angles to be in radians. The trajectory is shown in Figure 38. The same process is repeated for the second test position in Figures 39, 40 and 41.



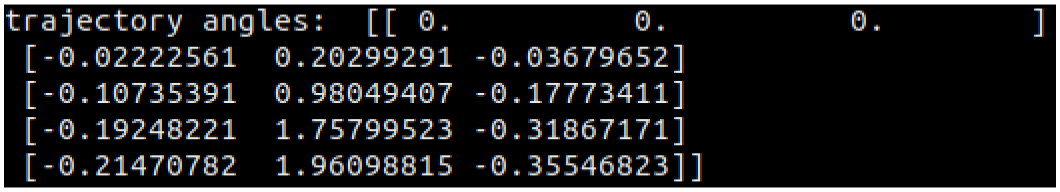
```
x:  0.13551392547021798 y:  0.13290658134452954 z:  0.602
trans_x:  0.6183708933671798 trans_y:  -0.10306892547021798 trans_z:  0.05022243794177228
```

Figure 36: Point received from camera transformed to the robot base frame in meters.



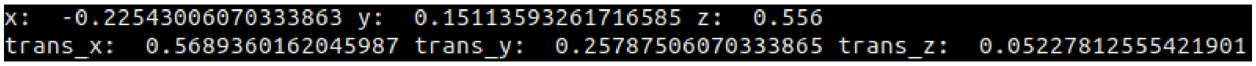
```
joint  1 :  -12.301852131242198 degrees
joint  2 :  112.35634452768454 degrees
joint  3 :  -20.366829177661145 degrees
```

Figure 37: Joints angles required in degrees.



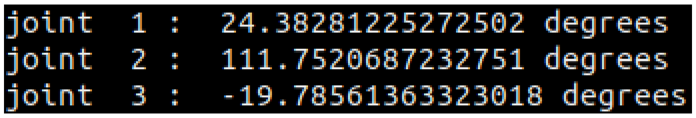
```
trajectory angles:  [[ 0.          0.          0.        ]
 [-0.02222561  0.20299291 -0.03679652]
 [-0.10735391  0.98049407 -0.17773411]
 [-0.19248221  1.75799523 -0.31867171]
 [-0.21470782  1.96098815 -0.35546823]]
```

Figure 38: Joint trajectory associated with testing point 1 in radians.



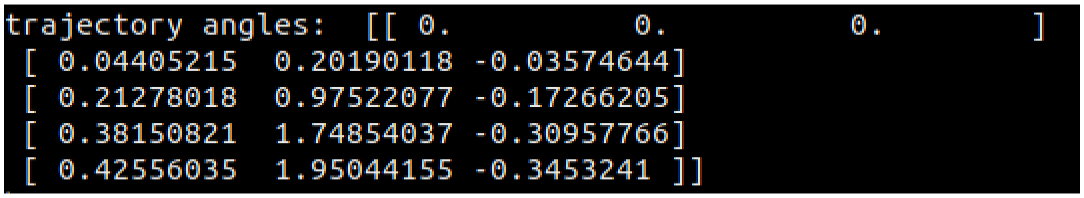
```
x:  -0.22543006070333863 y:  0.15113593261716585 z:  0.556
trans_x:  0.5689360162045987 trans_y:  0.25787506070333865 trans_z:  0.05227812555421901
```

Figure 39: Point received from camera transformed to the robot base frame in meters.



```
joint  1 :  24.38281225272502 degrees
joint  2 :  111.7520687232751 degrees
joint  3 :  -19.78561363323018 degrees
```

Figure 40: Joints angles required in degrees.



```
trajectory angles:  [[ 0.          0.          0.        ]
 [ 0.04405215  0.20190118 -0.03574644]
 [ 0.21278018  0.97522077 -0.17266205]
 [ 0.38150821  1.74854037 -0.30957766]
 [ 0.42556035  1.95044155 -0.3453241 ]]
```

Figure 41: Joint trajectory associated with testing point 2 in radians.

5.2.3 Logic Voltage Conversion Results

As previously mentioned in section 3.2.3, we attempted several solutions to convert the logic voltage from the drivers to a tolerable logic voltage for the Jetson TX2. While testing the bi-directional level shifter boards by Adafruit, we noticed other channels activating randomly. Due to this, it became incredibly difficult to consistently control the stepper motors, so the boards were deemed as unreliable. The handmade unidirectional level shifters returned more promising results, but they were not optimal. On occasion, the boards would create spastic signals that led to incorrect direction switching and less smooth motion. Finally, the last solution involving the basic transistor circuits was implemented. This approach created proper pulse width modulation signals to consistently control our stepper drivers. The simple design of the successful solution made it easy to understand, implement, and verify.

5.3 Kinematic Results

Using the encoders on the Husky A100 with the packages designed for the Husky A200 posed a significant challenge due to inaccuracies. While linear motion was relatively adequate with an error of about 2-3 centimeters, rotation based on encoder values proved to be a major challenge. The Husky A200, which features four-wheel drive (4WD), provided values through its Joint State publisher that were not suitable for our robot's differential drive system. While testing and validating our differential drive kinematic model, we learned that the joint state publisher did not provide us with the most accurate information to track our rotation. As a result, we had to purchase the BNO055. We chose this IMU because of its simplicity and it provided us with 9-DOF. Once we added the IMU to our system, our heading had an accuracy of up to 1 degree.

5.4 Navigation Results

Our navigation algorithm exhibited inconsistencies, particularly in instances where we expected the robot to rotate 45 degrees but it rotated 60 degrees instead. This discrepancy often led to the robot approaching walls within the quad. Moreover, we observed that the robot tended to travel a greater distance than calculated. We attribute these discrepancies to errors in the data points collected using smartphones. While we collected some points near our test area using the GPS module, we acknowledged the need to increase our tolerance for point identification.

To address these issues, we propose incorporating geofencing into our navigation algorithm. Geofencing is a virtual perimeter for a real-world geographic area, which can be dynamically generated or predefined. We suggest adding a parameter for geofencing to the GPS coordinates class, specifying a fence or radial boundary for a point. This addition would require creating a new message type for the Husky to receive coordinates to travel, as well as dynamically setting a tolerance for available points.

Additionally, we recognize the need for significant iteration to enhance the robustness of our navigation. One proposed improvement is to integrate a digital compass to obtain a more absolute heading. This would allow us to account for magnetic declination and use the bearing angle, which is the angle between two GPS coordinates concerning true north, to determine the required robot rotation. While our IMU includes a magnetometer, its results were not accurate enough due to testing inside the RBE MQP Lab (UH 150), known to be a Faraday cage. We suggest testing the magnetometer outdoors for better performance.

We encourage other teams to explore these algorithms further, particularly the SSS triangle method and the bearing angle algorithm, to improve navigation accuracy and consistency.

6. RECOMMENDATIONS

Although we successfully designed and implemented a mobile robot that can detect, collect, and dispose of garbage on WPI's Quad, there are still several areas of development in this project. If future teams are interested in continuing this MQP, we have compiled recommendations regarding current and future installments of the TRASHBOT to aid them in their engineering process.

6.1 Robotic Arm

6.1.1 HTD Belts

We recommend switching from 3D printed pulleys and belts to HTD (High Torque Drive) pulleys and belts due to their superior strength, durability, and precision. HTD belts are made from materials like neoprene or rubber with fiberglass reinforcement, specifically engineered to withstand high torque and maintain durability under stress. This ensures they can handle the demanding movements of robotic arms more effectively than 3D-printed parts, which may lack the necessary strength. Additionally, HTD components are manufactured to precise specifications, guaranteeing accurate motion transmission crucial for robotic arm precision. In contrast, 3D-printed parts may have inconsistencies that lead to inaccuracies in movement. HTD belts also offer superior wear resistance, maintaining consistent performance over time compared to 3D-printed belts, which may wear out more quickly, especially under high loads. While 3D printing can be cost-effective for prototyping, the long-term cost and availability of replacement parts may be higher compared to standardized HTD components, making HTD pulleys and belts the preferred choice for the overall functionality and reliability of a robotic arm. If further design modifications are made to the arm, it is imperative that one prints belts to prototype, then proceeds to purchase HTD belts and their respective pulleys.

6.1.2 Adaptive Gripper

We strongly recommend replacing the motor currently controlling the gripper with a more powerful alternative. The end effector would be able to gain a stronger grip on objects, which would greatly decrease the likelihood of said objects slipping out of the gripper as the arm moves to its drop-off point.

In addition to replacing the motor on the arm's gripper, we also recommend redesigning the gripper entirely. The current gripper was designed to only hold cylindrical items like bottles or cans. If future teams decide to incorporate different kinds of garbage into the collecting system, we recommend designing or purchasing a 3-Finger Adaptive Gripper. According to Robotiq, this gripper "is the best option for maximum versatility and flexibility. It picks up any object of any shape" https://robotiq.com/products/3-finger-adaptive-robot-gripper. The following Figure 42 compares the current gripper to the 3-Finger Adaptive Gripper.

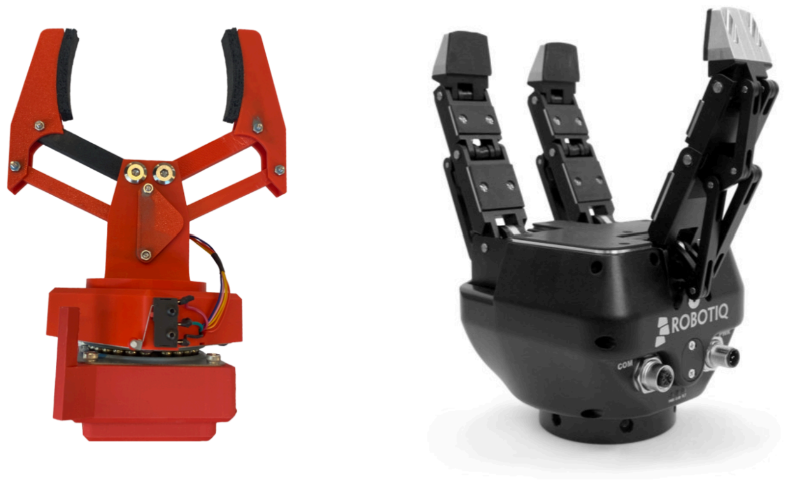


Figure 42: The current gripper and a potential iteration of the future gripper.

6.1.3 Utilize an Arduino Driver

We recommend transitioning away from using the Jetson as the primary driver for the robotic arm. While the Jetson has limited USB ports, you can expand connectivity by using USB hubs to add additional devices. By connecting an Arduino Uno or Mega to the Jetson via USB and utilizing it as the driver for the stepper motors, you can simplify wiring and reduce the margin for error in your system architecture. Although we initially used the Jetson to maintain a consistent programming language (Python), leverage ROS, and utilize Python's threading library for concurrency, we encountered challenges with GPIO pin management onboard the Jetson, particularly when conducting basic arm tests.

Using an Arduino for motor control allows for a form of pseudo-parallelism. This approach involves allocating fixed time slots for each joint to advance, ensuring smooth movement. For instance, when commanding joint 1, joint 2, and joint 3 to move simultaneously, the system cycles through each joint, pulsing the respective stepper motor until it reaches the desired position. Alternatively, each joint can be assigned dedicated CPU time on the Arduino, allowing each joint to move as many pulse counts as possible within its time slot before the next joint moves. This sequential movement mimics the scheduling of processes in operating systems, ensuring efficient and controlled motion of the robotic arm.

6.2 Point Cloud Processing and Grasping Mechanism

For this project, we already have a basic segmentation method through RANSAC. However, it is not very robust and constantly fails to segment out the aluminum can from the background. For future teams, it is recommended to add adaptive thresholding techniques for the RANSAC algorithm, or develop other point cloud processing techniques to segment out the detected object more efficiently especially if the future teams decided to add more trash type to collect. Furthermore, future teams are encouraged to develop grasp detection algorithms that use the point cloud data to analyze the optimal way to orient the gripper. This approach would enable the robotic system to dynamically determine the most effective gripping strategy based on the shape, size, and orientation of objects, providing a more efficient and secure grasp based on current or potential iteration of the future gripper.

6.3 Navigation

In section 5.4, we proposed enhancing our navigation system by integrating additional sensors. Our first recommendation is to conduct outdoor testing of the IMU's magnetometer to ascertain if it can provide accurate azimuth magnetic compass readings. If this test yields favorable results, there would be no need to acquire an additional sensor. However, if the test indicates that the IMU's magnetometer is insufficient, we propose replacing the onboard IMU with a high-precision magnetometer or digital compass. This upgrade would provide the Husky with an absolute heading relative to magnetic north. Utilizing a database containing magnetic declination, we can estimate or calculate the robot's heading in relation to true north. This information enables us to use bearing angles to determine the necessary angle for the robot to rotate to reach a specified point. Furthermore, to enhance the robustness of our algorithm, we recommend implementing geofencing, as discussed in section 5.4. This entails conducting multiple tests to determine an appropriate fencing radius, ensuring the algorithm's effectiveness in various scenarios.

6.4 LiDAR

One of the many systems we wanted to implement on the robot is obstacle avoidance with the SICK MRS1104c LiDAR. As displayed in previous figures, the LiDAR has already been mounted on the Husky. If the robot is utilized in dynamically changing environments in the future, some type of obstacle avoidance must be implemented. The robot needs to avoid people and objects it encounters on its given path. A recommended method to achieve this would be using the LiDAR to collect a point cloud that saves a map of its surroundings, and rapidly uses a fast motion planning algorithm such as RRT* (Rapidly-exploring Random Trees) to find a path around the obstacle.

6.5 New Chassis and Computer

We recommend any team following our work to search for a new chassis. The Husky A100 is extremely outdated and comes with significant setbacks when implementing it in ROS. Such setbacks include the lack of a given URDF file, incorrect kinematics based on the newer Husky A200, and inaccurate encoders. We feel that the addition of a new chassis, whether purchased or made, would significantly improve the overall quality of the project.

Alongside the many setbacks of the outdated Husky A100, the Nvidia Jetson TX2 is also unsupported and old. The TX2, although a powerful processor, only works with Nvidia Jetpack OS 4.6.4. The Jetpack 4.6.4 OS only works with Ubuntu 18.04, meaning we can only use ROS Melodic with the robot. Both Ubuntu 18.04 and ROS Melodic are generally unsupported, which makes debugging difficult. This problem also contributes to the numerous problems we faced when trying to implement newer ROS packages made for ROS Noetic or newer. As a team, we would recommend that the TX2 be replaced with a newer Nvidia Jetson module. The newer modules are far more powerful and reliable than the deprecated Nvidia Jetson TX2.

REFERENCES

Satav, A. G., Sunidhi Kubade, Chinmay Amrutkar, Arya, G., & Pawar, A. (2023). A state-of-the-art review on robotics in waste sorting: scope and challenges. *IJIDeM*, 17. https://doi.org/10.1007/s12008-023-01320-w

Kulshreshtha, M., Chandra, S. S., Randhawa, P., Tsaramirsis, G., Khadidos, A., & Khadidos, A. O. (2021). OATCR: Outdoor autonomous trash-collecting robot design using yolov4-tiny. *Electronics*, *10*(18), 2292. https://doi.org/10.3390/electronics10182292

3D Printed 6DoF Arduino Robot Arm - Overview of the Arm [Part 1]. (2019, November 1). Www.youtube.com. https://www.youtube.com/watch?v=wI4Jh-T0Tlo

Lynch, K. M., & Park, F. C. (2017). *Modern Robotics : Mechanics, Planning, and Control*. University Press.

Heinemann, M., Herzfeld, J., Sliwinski, M., Johannes Hinckeldeyn, & Jochen Kreutzfeldt. (2022). A metrological and application-related comparison of six consumer grade stereo depth cameras for the use in robotics. *IEEE*. https://doi.org/10.1109/rose56499.2022.9977421

RealSenseTM Comparison. (2023). Docs.luxonis.com; Luxonis. Retrieved April 23, 2024, from https://docs.luxonis.com/projects/hardware/en/latest/pages/guides/realsense_comparison/

Han, S., Choi, B., & Lee, J. (2008). A precise curved motion planning for a differential driving mobile robot. *Mechatronics*, 18(9), 486–494. https://doi.org/10.1016/j.mechatronics.2008.04.001

What is GPS RTK? (n.d.). Learn.sparkfun.com; Sparkfun Learn. Retrieved April 23, 2024, from https://learn.sparkfun.com/tutorials/what-is-gps-rtk?_ga=2.147108723.591115304.1713787680-1596351493.1710347125&_gac=1.148314309.1713787680.Cj0KCQjwlZixBhCoARIsAIC745An2XcwC_l03cwwsvv8rE8CTJo56q0muxesDcrkdCH1olVIBJ8rcrgaAlSGEALw_wcB

APPENDIX A: J21 Connector on the Jetson

Optional	Default	Signal voltage	Pin		Signal voltage	Default	Optional
	3.3V Supply		1	2	5.0V Supply		
	I2C1 SDA	3.3V	3	4	5.0V Supply		
	I2C1 SCL	3.3V	5	6	Ground		
Audio I2S MCLK	GPIO	3.3V or 1.8V	7	8	3.3V	UART TXD	GPIO
	Ground		9	10	3.3V	UART RXD	GPIO
UART RTS	GPIO	3.3V	11	12	3.3V or 1.8V	GPIO	Audio I2S CLK
	GPIO	3.3V or 1.8V	13	14	Ground		
	GPIO	3.3V	15	16	3.3V or 1.8V	GPIO	Digital Mic Input
	3.3V Supply		17	18	3.3V or 1.8V	GPIO	
SPI1 MOSI	GPIO	3.3V or 1.8V	19	20	Ground		
SPI1 MISO	GPIO	3.3V or 1.8V	21	22	3.3V	GPIO	
SPI1 SCLK	GPIO	3.3V or 1.8V	23	24	3.3V or 1.8V	GPIO	SPI1 CS0
	Ground		25	26	Not Used		
	I2C0 SDA	3.3V	27	28	3.3V	I2C0 SCL	
	GPIO	3.3V or 1.8V	29	30	Ground		
	GPIO	3.3V	31	32	3.3V or 1.8V	GPIO	Digital Mic Clock

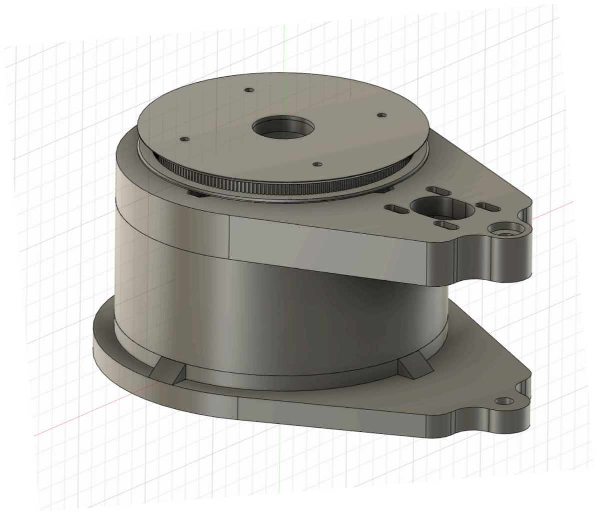
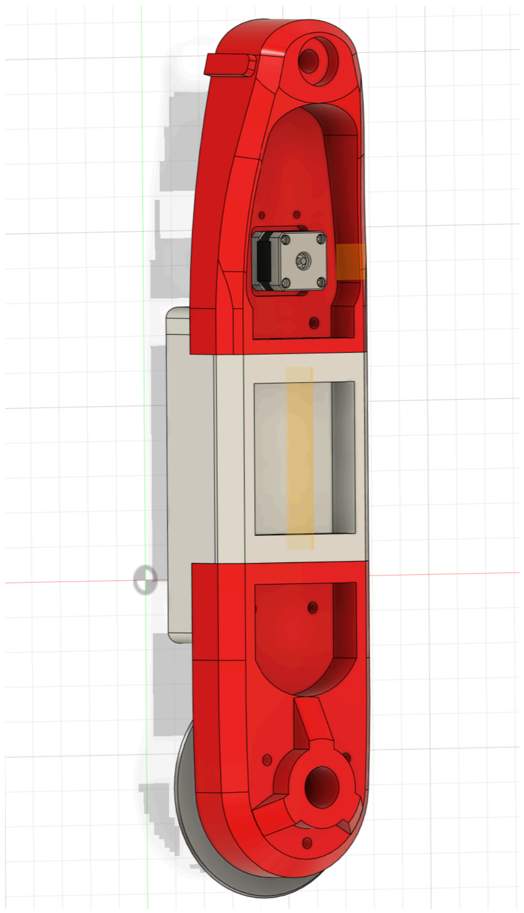
Optional	Default	Signal voltage	Pin		Signal voltage	Default	Optional
	GPIO	3.3V or 1.8V	33	34	Ground		
Audio I2S LRCK	GPIO	3.3V or 1.8V	35	36	3.3V	GPIO	UART CTS
	GPIO	3.3V	37	38	3.3V or 1.8V	GPIO	Audio I2S DIN
	Ground		39	40	3.3V or 1.8V	GPIO	Audio I2S DOUT

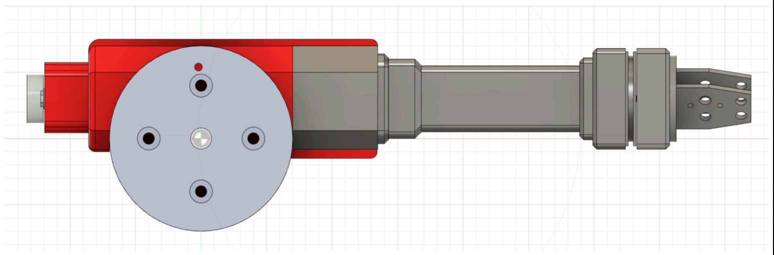
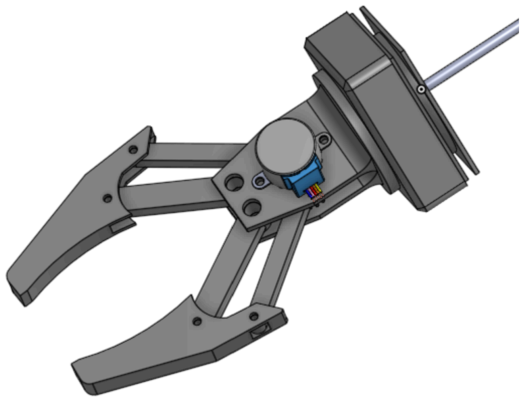
APPENDIX B: J17 Connector on the Jetson

Optional	Default	Signal voltage	Pin		Signal voltage	Default	Optional
	GPIO	3.3V	1	2	3.3V Supply		
	Not Used			3	4	1.8V Supply	
	CAN0 RX	3.3V	5	6	-	GPIO	
	CAN0 TX	3.3V	7	8	5.0V Supply		
	GPIO	3.3V	9	10		Ground	
	Ground		11	12	1.8V	I2C2 CLK	GPIO
	GPIO	3.3V	13	14	1.8V	I2C2 SDA	GPIO
	CAN1 RX	3.3V	15	16		WDT RESET OUT	GPIO
	CAN1 TX	3.3V	17	18	1.8V	I2C3 CLK	

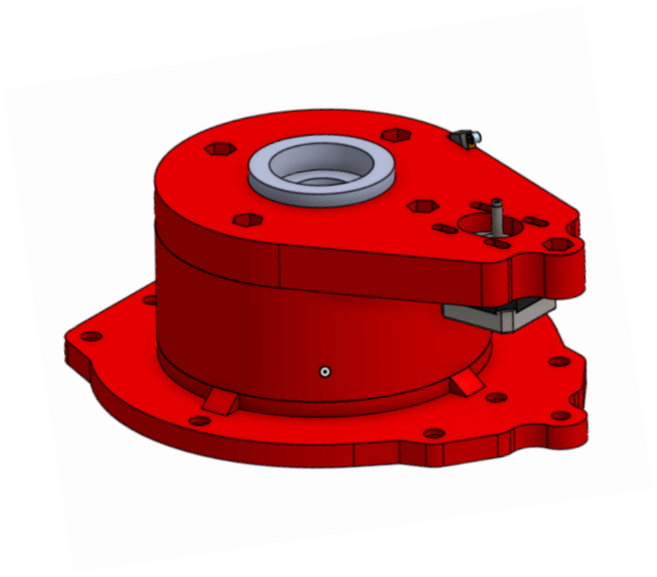
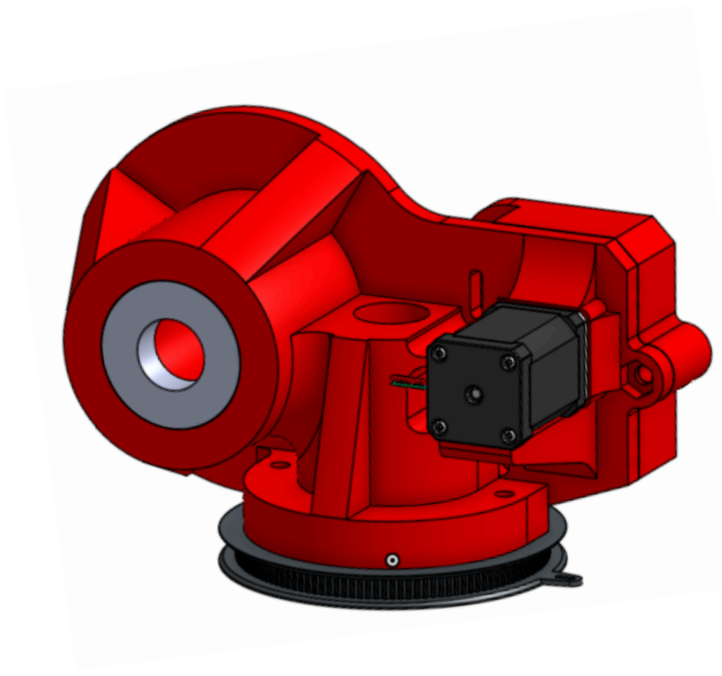
Optional	Default	Signal voltage	Pin		Signal voltage	Default	Optional
	GPIO	3.3V	19	20	1.8V	I2C3 DAT	
	Ground		21	22	**1.8V**	GPIO	
	I2S1 CLK	1.8V	23	24	1.8V	I2S1 SDOUT	
	I2S1 SDIN	1.8V	25	26	1.8V	I2S1 LRCLK	
SPDIF OUT or GPIO	DSPK OUT CLK	1.8V	27	28	Ground		
SPDIF IN or GPIO	DSPK OUT DAT	1.8V	29	30	Not Used		

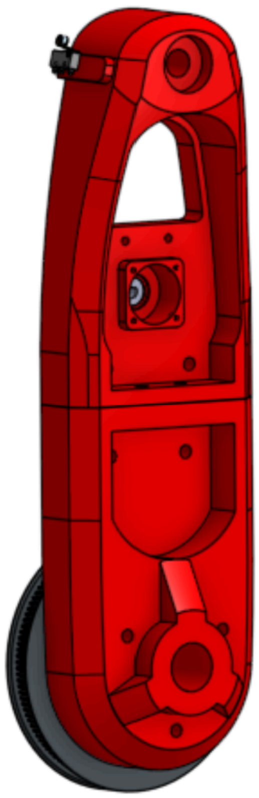
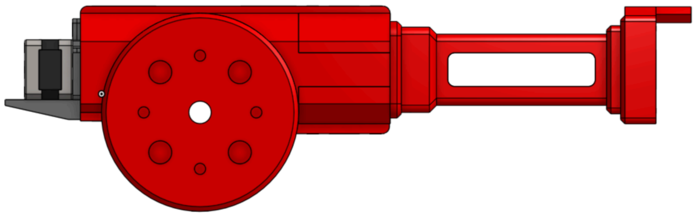
APPENDIX C: Our Initial CAD Design and Modifications from Reference Arm CAD Design

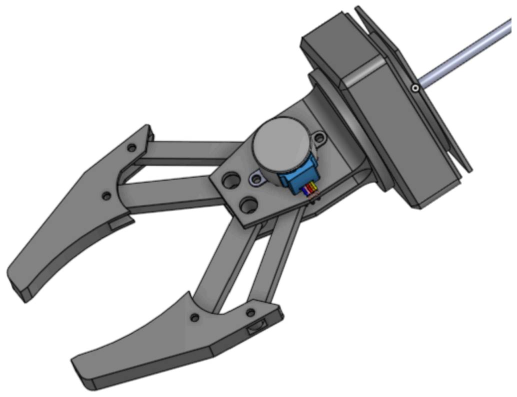
	Our Initial CAD Design	Modifications from Reference Arm CAD Design
Base		● Scaled by 250% ● Increased base tube width for structural integrity
Link 2		● Scaled by 250% ● Added removable center piece to extend the link

Link 3		● Scaled by 250% ● Redesigned the motor mount to properly hold the stepper in place ● Redesigned to be able to detach into two pieces if adjustments needed to be made ● Redesigned the gripper-end of the link so a lazy susan bearing could be attached
Gripper		● Completely redesigned gripper – original gripper would not be strong enough to retrieve and relocate objects

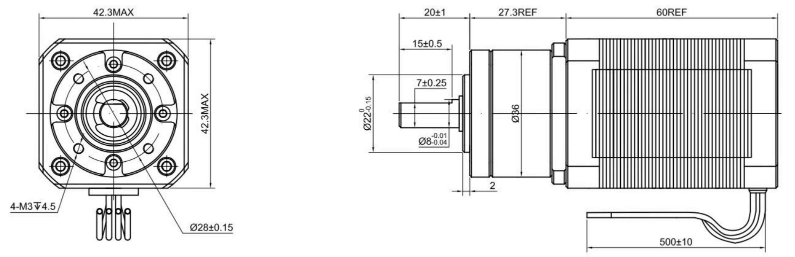
APPENDIX D: Our Final CAD Design and Modifications from Initial CAD Design

	Our Final CAD Design	Modifications from Initial CAD Design
Lower Base		● Redesigned base plate and added screw holes so the arm could be attached directly to the Husky's top plate ● Added limit switch position
Upper Base		● Added several supports for the stepper motor controlling joint 2 – stepper kept overheating and melting the PLA before adding supports ● Added two tensioner pulleys to increase the number of gearbelt teeth making contact with the gear teeth ● Added limit switch position

Link 2		● Added limit switch position ● Reinforced area around the stepper motor controlling joint 3 so no overheating takes place
Link 3		● Redesigned the motor mount to hold a stronger motor in place ● Reinforced the connection between the two pieces ● Hollowed out the second piece to make the arm lighter and easier to manipulate ● Added extension by the gripper-end of the link so it could hit the limit switch placed on the gripper

Gripper		● Added limit switch position ● Reinforced areas that originally had thin, perpendicular pieces

APPENDIX E: Data Sheets for the Robotic Arm’s Joint Motors

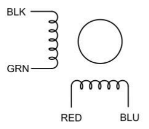


SPECIFICATION \ CONNECTION	BIPOLAR
AMPS/PHASE	2.10
RESISTANCE/PHASE(Ohms)@25°C	1.60±10%
INDUCTANCE/PHASE(mH)@1KHz	3.00±20%
HOLDING TORQUE w/o GEARBOX(Nm)[lb-in]	0.65[5.75]
GEAR RATIO	$5\frac{2}{11}$
EFFICIENCY	90.00%
STEP ANGLE w/o GEARBOX(°)	1.80
BACKLASH@NO-LOAD	<=3°
MAX.PERMISSIBLE TORQUE(Nm)	3.00
MOMENT PERMISSIBLE TORQUE(Nm)	5.00
SHAFT MAXIMUM AXIAL LOAD(N)	50.00
SHAFT MAXIMUM RADIAL LOAD(N)	100.00
WEIGHT(Kg)[lb]	0.80[1.76]
TEMPERATURE RISE:MAX.80°C（MOTOR STANDSTILL;FOR 2PHASE ENERGIZED）	
AMBIENT TEMPERATURE -10°C~50°C[14°F~122°F]	
INSULATION CLASS B 130°C[266°F]	

TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A —	BLK	A
2	A\ —	GRN	A\
3	B —	RED	B
4	B\ —	BLU	B\

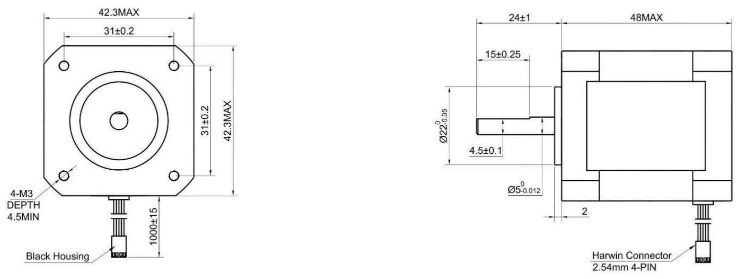
FULL STEP 2 PHASE-Ex. ,
WHEN FACING MOUNTING END (X)

STEP	A	B	A\	B\		
1	+	+	-	-	↓ CW	CCW ↑
2	-	+	+	-		
3	-	-	+	+		
4	+	-	-	+		



STEPPERONLINE®

	APVD		1.4.2023	STEPPER MOTOR
	CHKD			
1 : 1	DRN			17HS24-2104S-PG5
SCALE	SIGNATURE		DATE	

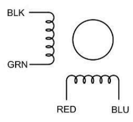


SPECIFICATION \ CONNECTION	BIPOLAR
AMPS/PHASE	2.00
RESISTANCE/PHASE(Ohms)@25°C	1.40±10%
INDUCTANCE/PHASE(mH)@1KHz	3.00±20%
HOLDING TORQUE(Nm)[lb-in]	0.59[5.22]
STEP ANGLE(°)	1.80
STEP ACCURACY(NON-ACCUM)	±5.00%
ROTOR INERTIA(g-cm²)	82.00
WEIGHT(Kg)[lb]	——
TEMPERATURE RISE:MAX.80°C（MOTOR STANDSTILL;FOR 2PHASE ENERGIZED）	
AMBIENT TEMPERATURE -10°C~50°C[14°F~122°F]	
INSULATION RESISTANCE 100 Mohm（UNDER NORMAL TEMPERATURE AND HUMIDITY）	
INSULATION CLASS B 130°C[266°F]	
DIELECTRIC STRENGTH 500VAC FOR 1MIN.(BETWEEN THE MOTOR COILS AND THE MOTOR CASE)	
AMBIENT HUMIDITY MAX.85%(NO CONDENSATION)	

TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A+ —	BLK	A+
2	A- —	GRN	A-
3	B+ —	RED	B+
4	B- —	BLU	B-

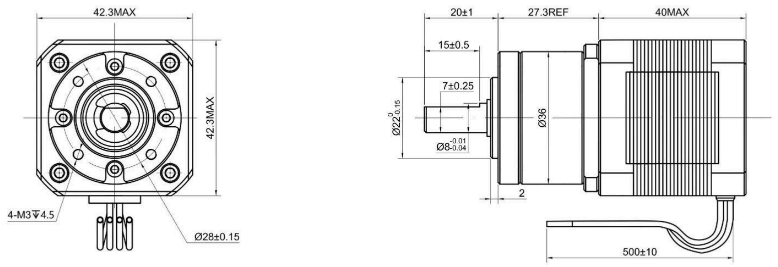
FULL STEP 2 PHASE-Ex. ,
WHEN FACING MOUNTING END (X)

STEP	A+	B+	A-	B-		
1	+	+	-	-	↓ CW	CCW ↑
2	-	+	+	-		
3	-	-	+	+		
4	+	-	-	+		



STEPPERONLINE®

	APVD		10.30.2020	STEPPER MOTOR
	CHKD			
1:1	DRN			17HS19-2004S1
SCALE	SIGNATURE		DATE	

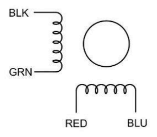


SPECIFICATION \ CONNECTION	BIPOLAR
AMPS/PHASE	1.68
RESISTANCE/PHASE(Ohms)@25°C	1.60±10%
INDUCTANCE/PHASE(mH)@1KHz	3.10±20%
HOLDING TORQUE w/o GEARBOX(Nm)[lb-in]	0.39[3.45]
GEAR RATIO	$5\frac{2}{11}$
EFFICIENCY	90.00%
STEP ANGLE w/o GEARBOX(°)	1.80
BACKLASH@NO-LOAD	<=3°
MAX.PERMISSIBLE TORQUE(Nm)	3.00
MOMENT PERMISSIBLE TORQUE(Nm)	5.00
SHAFT MAXIMUM AXIAL LOAD(N)	50.00
SHAFT MAXIMUM RADIAL LOAD(N)	100.00
WEIGHT(Kg)[lb]	——
TEMPERATURE RISE:MAX.80°C（MOTOR STANDSTILL;FOR 2PHASE ENERGIZED）	
AMBIENT TEMPERATURE -10°C~50°C[14°F~122°F]	
INSULATION CLASS B 130°C[266°F]	

TYPE OF CONNECTION (EXTERN)		MOTOR	
PIN NO	BIPOLAR	LEADS	WINDING
1	A+ —	BLK	A+
2	A- —	GRN	A-
3	B+ —	RED	B+
4	B- —	BLU	B-

FULL STEP 2 PHASE-Ex. ,
WHEN FACING MOUNTING END (X)

STEP	A+	B+	A-	B-		
1	+	+	-	-	↓ CW	CCW ↑
2	-	+	+	-		
3	-	-	+	+		
4	+	-	-	+		



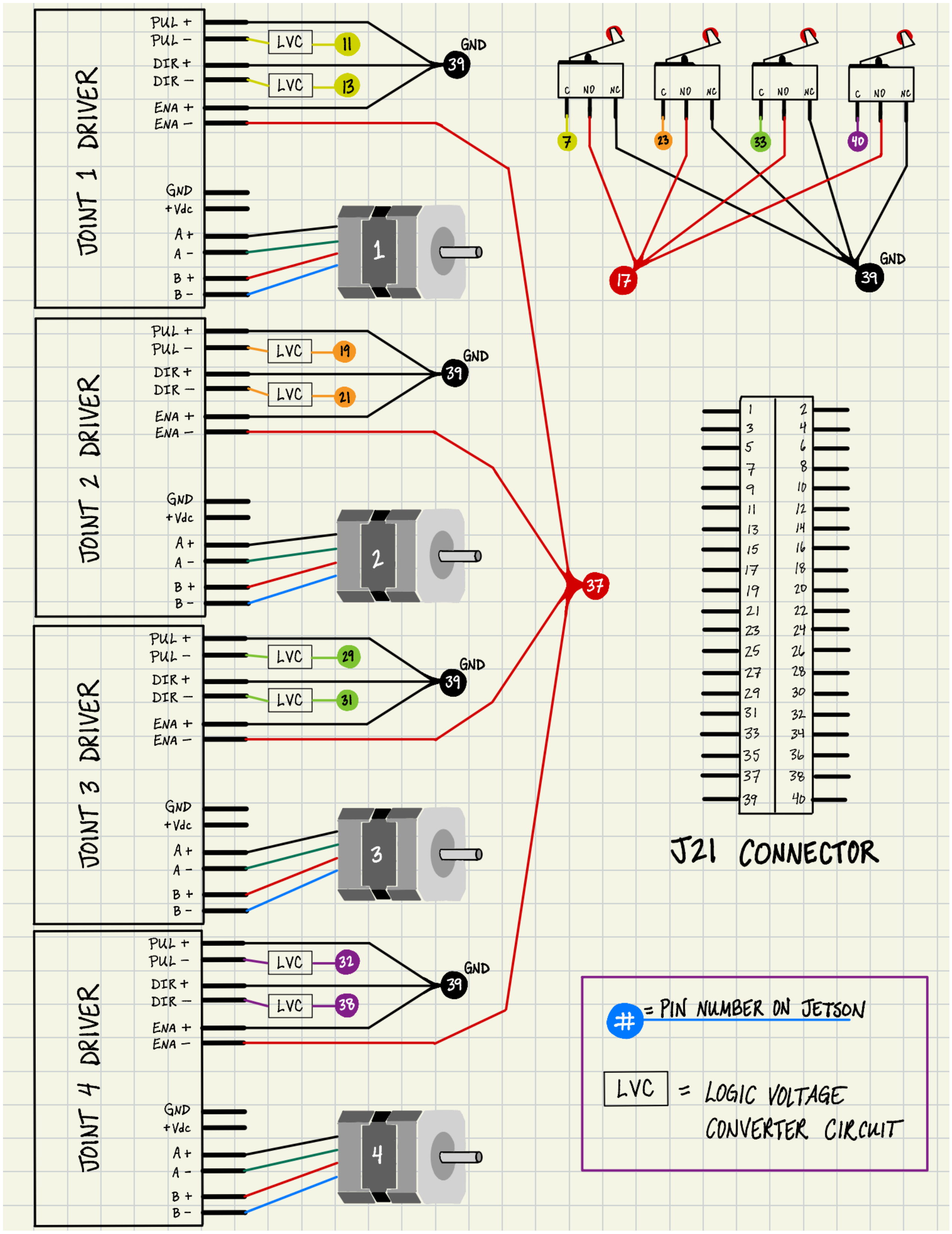
STEPPERONLINE®

	APVD		1.5.2023
	CHKD		
1:1	DRN		
SCALE	SIGNATURE		DATE

STEPPER MOTOR

17HS15-1684S-PG5

APPENDIX F: Electrical Schematic of the Robotic Arm



APPENDIX G: Haversine Formula

φ_1, φ_2 are the latitude of point 1 and latitude of point 2

λ_1, λ_2 are the longitude of point 1 and longitude of point 2

$$\operatorname{hav}(\theta) = \sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2}$$

$$\begin{aligned} d &= 2r\arcsin\left(\sqrt{\operatorname{hav}(\varphi_2-\varphi_1) + (1-\operatorname{hav}(\varphi_1-\varphi_2)-\operatorname{hav}(\varphi_1+\varphi_2))\cdot\operatorname{hav}(\lambda_2-\lambda_1)}\right) \\ &= 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + \left(1-\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right)-\sin^2\left(\frac{\varphi_2+\varphi_1}{2}\right)\right)\cdot\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \\ &= 2r\arcsin\left(\sqrt{\sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + \cos\varphi_1\cdot\cos\varphi_2\cdot\sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right). \end{aligned}$$