Robotic Trash Concierge

George Gao, Jack Girel-Mats, Zachary Mason

Department of Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—The modern workplace has evolved such that many office workers collaborate in the same area. However, this has resulted in many small trash bins scattered across the office. These bins regularly overflow, forcing janitors to make repeated trips to hundreds of tiny trash-bins scattered across an office space. This is not only inconvenient and unhygienic for office workers, but it's a point of great inefficiency for janitorial staff to check hundreds of possibly empty bins. We have devised a solution using an autonomous robot capable of taking out an office worker's trash can by bringing it to a centralized dumping area, where janitorial staff will empty dirty bins and place clean bins for the robot to bring back to their original positions. This method streamlines the trash collection process and eliminates the need for janitors to check hundreds of tiny bins across the office space.**

***Index Terms*—Autonomous, LiDAR, Robot, Roomba, ROS, SLAM, Trash Bin**

I. Introduction

The work environment has continuously evolved from era to era, and along with it problems and solutions. The modern workplace is no exception. Offices have become increasingly open, and collaborative, with tech campuses at the forefront of such changes. They have introduced architectures with open seating and shared desk space. It however introduces a copious amount of trash bins into the office, one for each desk. This places an excess burden on janitorial staff to accomplish the dull, monotonous work to take out all the scattered bins.

We've noticed that the scale of the problem and its monotonous nature, this was an appropriate application of automation. Thus we have designed a solution as such.

Our proposed solution introduces a robot designed for the flexible nature of open office spaces, with a goal of greatly reducing the workload on custodial staff. Our robot will be able to map out an open office area, and track where trash bins are located. Using a tracking system, the robot will be able to navigate to the trash bins, and bring them to a centralized dumping ground to be taken out by janitorial staff. Janitorial staff will then return clean trash bins to their original locations. This reduces the running around custodians have to do as now the trash is centralized in a common area.

While there are no competing products on the market, we realize that some technologies can be retrofitted to solve the same problem, notably other robotic delivery systems such as Proteus Robotics warehouse robots [1]. However, each such solution was not designed for an office space specifically, and would be clunky to use.

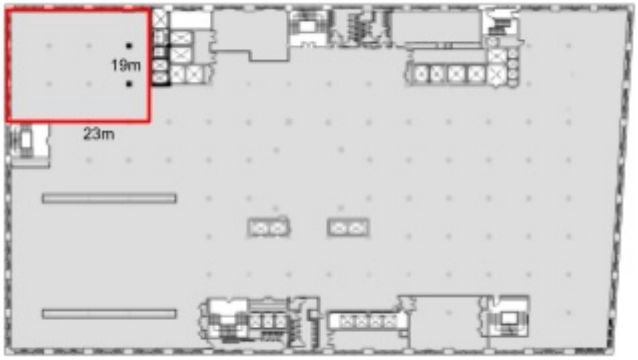


Fig. 2.1. Floorplan of Meta's 770 Broadway office in NYC [2].

II. Use-Case Requirements

We first introduce unit metrics that our project must meet. We estimate that the average worker will be out of the office work area by 6 p.m., so we chose a one hour buffer for the starting time of our robot, 7 p.m. We also take note that night custodians should still be in the office by 12 a.m and thus our robot must be finished by then, giving us a working time of five hours. Within these five hours a robot must travel at a speed of 0.21m/s. This comes from our use-case study into the dimensions of the average working area of a Meta office space (Fig. 2.1). The room we looked at came from Meta's 770 Broadway NY office, which had a working area of 19m x 23m, and assuming a common starting ground to the middle of the room (as the average distance for all trash cans) we calculated a round distance time of 42m. For a working area with 90 trash cans for 5 hours; we reached our 0.21m/s average speed after adding some buffer time.

The next use-case requirement we want to meet is to have an 85% trip success rate, with a successful trip defined as successfully navigating to a trash can and bringing it back without spillage or human collisions. The 85% comes from two sub requirements in series: a 90% bin docking success rate, 95% non-collision rate, and a 99% no spillage during transport. We think 90% is viable for bin docking due to the numerous ways a bin could be moved or mishandled by humans in between cleaning periods. A 95% non human collision rate is also acceptable due to the work period of the robot being designed for an after hours setting, alongside its menial movement speed of 0.21m/s, meaning collisions are both unlikely to happen and inconsequential if they were to happen. Nonetheless we don't want to inconvenience workers, thus we decided 95% would be acceptable in this case. For a given bin collection trip, there should be at least a 99% rate of transport without the bin tipping over. We deem this to be acceptable since spilled bins nullify our system, and our work area of 90 trash cans means 1 bin could tip over, in

which case it wouldn't be too burdensome for a custodial staff member to take care of manually, since they may already be performing a separate task nearby. Considering these success rates in series, we follow the derivation

$$R_s = R_{bin\ docking} \times R_{non-collision} \times R_{bin\ spillage} \quad (1)$$

to reach the final success rate of being 85%. We deem 85% success rate to be acceptable due to the still drastic reduction in labor that it entails, cutting the number of trips to be a trivial amount. Furthermore, due to how 85% encompasses human collisions, the number of trips that succeed in purely bringing back trash cans would be much higher. Following the same equation as (1), except getting rid of the non-collision rate, we achieve a success rate of 89%. In terms of our use case study this means around 10 left-over bins, a drastic reduction from 90.

Another requirement for our system is it should be able to lift a 10 pound, 7-gallon standard trash bin. Given the office environment where bins are expected to be emptied nightly, bins won't fill up too much throughout the day, as it's mostly going to be food waste and packaging that gets thrown out, along with the occasional document, none of which are that heavy. Bins themselves weigh 2-3 pounds, so this allows for 7–8 pounds of waste to be removed at a time.

We also want to be mindful of the health and safety implications of our project. Our robot is going to be working alongside other humans in an office environment, so it's important to keep the safety of the office staff central to our project's goals. This includes adding obstacle detection, emergency stop procedures, and making the robot easy to see.

III. Architecture and/or Principle of Operation

The high level architecture of our solution is laid out in the block diagram below in Fig. 3.1. We will now describe the overall principle of each subsystem in detail.

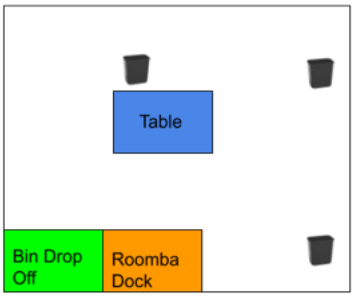


Fig. 3.2. A simplified model demonstrating a high level representation of our robot's operating environment

A. Bin Pickup System

The pickup apparatus consists of an arm powered by motors for a stable and reliable system that minimizes the janitorial staff's workload. The trash bins are placed on specialized bases which allow for a stable and consistent docking interface with our robot.

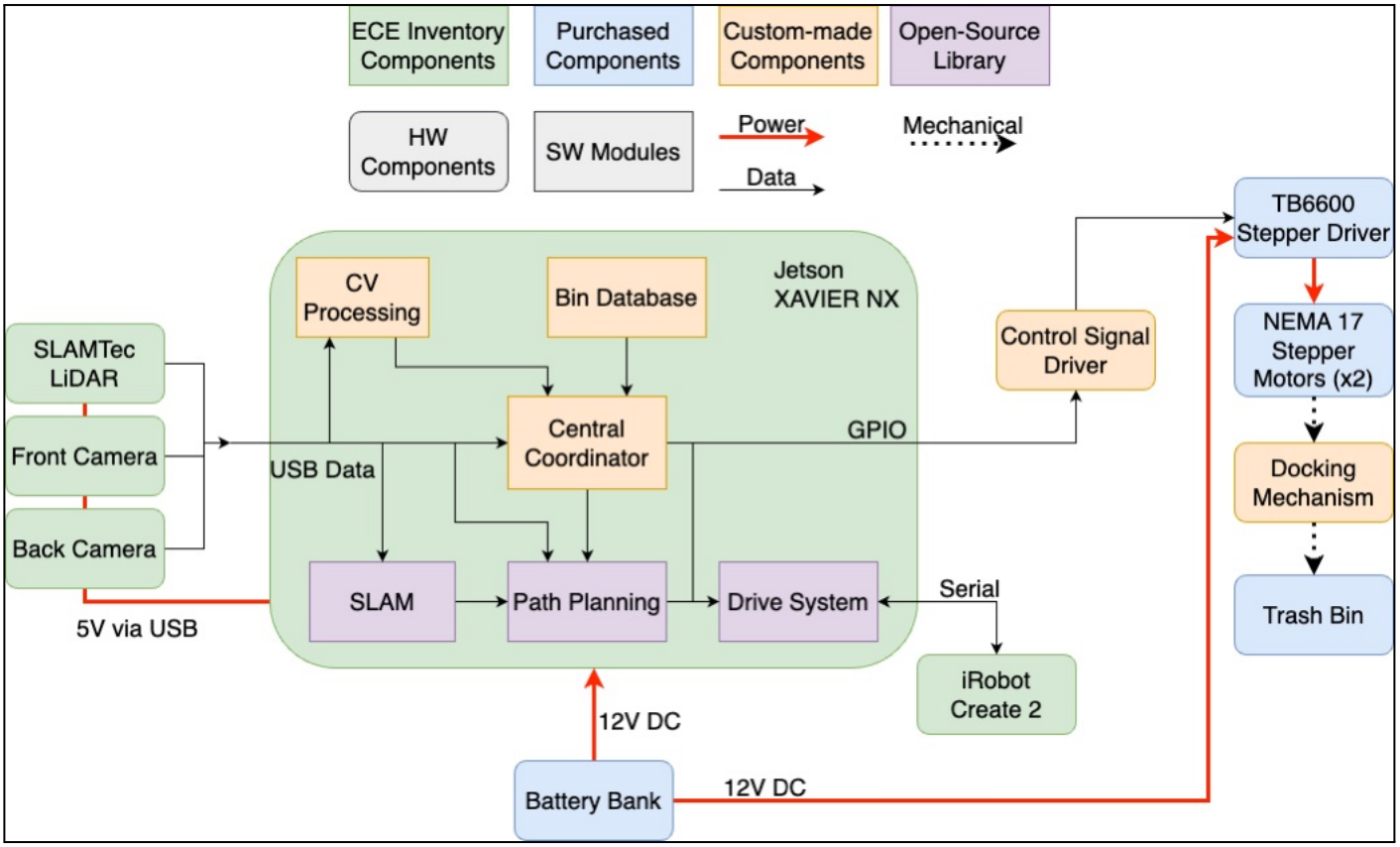


Fig. 3.1. System block diagram

B. Mapping

In order to create an autonomous system, we need to have knowledge of the office space that the robot will be operating inside. To accomplish this we will use SLAM to precreate a map of the office environment. This map will contain elements such as desks, chairs, walls, and bin locations.

C. Bin Tracking

In order to keep track of and locate the bins in an office environment, each bin will be given a unique tag. During mapping, every bin's location will be recorded into a database. This database will then be queried during pickup for the bins' statuses and coordinates.

D. Path Planning System

Once our robot has created a map of its environment, and enters the pickup routine, it will need to plan a path from its location to its desired bin pickup location. The path planning system will be responsible for creating a path from the map of the office environment. This system will also implement a local path planning strategy to accommodate new obstacles.

E. Drive System

For the robot, we will be using an iRobot Create2 (Roomba). This will allow for a solid base on which we can mount various components, as well as being supported by interfaces to send custom movement commands.

F. Computer Vision System

Our camera feeds will be piped into OpenCV to accomplish bin identification. We will calibrate our camera to be able to calculate the distance and orientation of bins from our robot. These will be used during mapping for bin location acquisition, and during pickup for bin alignment.

A depth-sensing system will be used for mapping of the office environment, as well as obstacle avoidance and navigation during the bin pickup routine.

G. State Coordinator

Our robot has many tasks: mapping, path planning, bin identification, bin alignment, bin pickup, and so on. The state coordinator will have the final say on signals sent to the drive system and bin pickup system. It will also be responsible for identifying situations in which janitorial staff are needed, such as in the case of the Roomba getting stuck.

IV. Design Requirements

Our project's design requirements are derived from our use case requirements, and specify the performance of various subsystems that will meet our use case requirements after system integration.

To design a robot that fits within the constraints of open office environments, we used the Meta NYC office to estimate the size restrictions for our robot. Based on our findings the robot's height needs to be less than 30 inches and it needs to be less than three feet wide and and 3 feet long to fit between and under desks.

During the robot's pickup routine, it may encounter obstacles and humans, which could result in trash spills and collisions. We determined that a system for obstacle detection and avoidance was required. Additionally, it is important for the robot to be easily identifiable by office and janitorial staff.

Given the use cases' movement speed requirement and cleanup time period, we need to ensure the robot has enough power to complete its tasks. The Roomba has an advertised runtime of 2 hours, but this is with the cleaning motor active, which uses a significant amount of power. Since we're only using the drive motors, the overall power use is significantly reduced and should allow us to meet the 5 hour requirement. However, the robot's computation system, bin pickup system, and sensing systems will require an additional battery attached to the Roomba's body.

Our camera system should be able to process data with at least 15fps (frames per second), as this is recommended for applications that move at less than 2m/s for proper CV applications [3].

For accurate bin identification and environment tag usage, we specify that the computer vision subsystems must be able to correctly identify with 90% accuracy from greater than 2 meters away in distance. 2 meters away would mainly be used for sanity checking the location that the robot is moving towards is correct, and is not that important. Thus lower accuracy is acceptable. The accuracy must reach 95% within 1 meter away but greater than 0.15 meters away from the camera, barring any obstructions when in proper office lighting conditions, as this would be the distances the docking procedure would be working within. We want to be as accurate as possible due to the inconvenient nature of fixing such mistakes of moving the wrong bin. However, due to the video stream-like nature of our application, both 90% and 95% accuracy over the course of 15fps means it's virtually impossible over the course of a docking procedure to continuously mistake the tag.

Furthermore, the sensing system must be able to do relative distance sensing within 1 meter away with an accuracy of 90% with accuracy in this case defined by how far off the camera position is to the labeled tag with the following equation:

$$1 - |d_r - d_s| / d_r \quad (2)$$

with d_r being the real distance and d_s being the sensed distance. We apply a 90% accuracy to be sufficient due to the relative nature of docking to a bin, which would be the main use case of distance sensing. Due to how close bins are and low movement speeds, software bounds can be used to heavily compensate to be safely within distance bounds to properly pick up bins with the given location accuracy, as long as sensed distances are consistent.

In order to lift a 10 lb bin vertically, a great enough force needs to be generated, and the bin needs to be lifted high enough off the ground to prevent any contact with the ground.

Any floor imperfections are assumed to be relatively small (<1in). The bin weighs approximately 44 N, and should be lifted up to 1.5in (~4cm) total, allowing for 0.5in of vertical misalignment during the docking process.

To properly traverse the office space and reach bin locations, our mapping of the area should be qualitatively accurate. Then during pickup, we need to have a localization accuracy of at most .5m in order to not bump into mapped obstacles and end up in the desired destination.

V. Design Trade Studies

A. Trash Bin Modifications

In order to support moving a 10 pound trash can in the office environment, we considered many bin modifications, including the addition of wheels to the bin, a fixed pedestal, forklift-style slots, circular base plate, or side-mounted flange.

Because our Roomba may not have perfect positioning and drive accuracy, any bin that requires fine positioning would make it difficult to meet the 90% docking success rate. This meant a forklift-style slot would not be feasible since it requires alignment with the bin at a specific angle. Similarly, we decided against using wheels on the trash bin itself since any physical contact by the Roomba could shift the bin's position, using up valuable time by having to re-adjust, and perhaps even causing a docking failure if the guidance algorithm was slightly off.

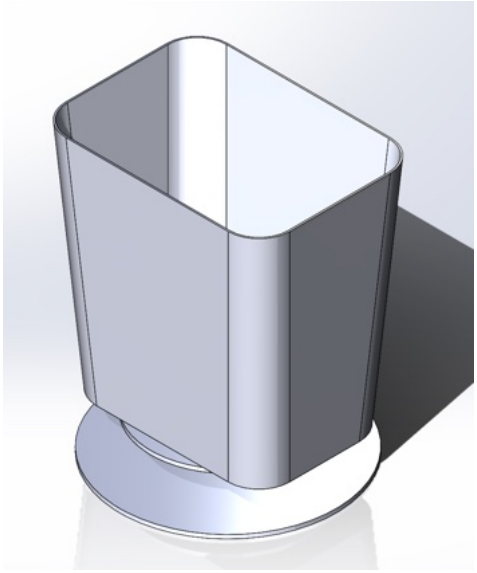


Fig. 5.1. CAD model showing the circular base selected for the bin.

Another idea was to use a pedestal that we could place the trash bin on top of, where the Roomba could drive underneath and lift the bin for pickup. The issues with this solution are twofold. One is that a trash bin elevated off the floor restricts the position it can be in for a pickup, so employees would not be able to move a trash bin around unless they also moved the pedestal. It also complicates the design unnecessarily since the cameras and LiDAR would have to be mounted low to the ground where they have less visibility.

This leaves the side-mounted flange and circular base as the last 2 bin attachments to consider. Both these designs would offer greater variability in pickup positioning, but the side-mounted flange would require a more complicated lift system as explored in part B below. This resulted in our selection of the circular base plate as the best option that would allow us to meet use case requirements as shown in Fig. 5.1.

B. Docking Arms

Since the Roomba is a circular object with no existing attachment to mount a lift system on, we had to design our own. In part A above, we explored the different bin modifications and arrived at two potential solutions. Because 7-gallon trash cans are rectangular, the apparent width varies based on the position around it, so our arms would have to have to similarly change width if using a side-mounted flange. This would require actuation in two axes, which reduces the overall strength and could jeopardize our 10 pound load requirement, in addition to increasing complexity that could reduce the docking success rate. For this reason, we chose to use fixed arms in conjunction with an omnidirectional base.

C. Bin Lift System

Since we decided against the use of wheels, the bin must be physically lifted off the ground for transport. There was really only one practical option here, which was to use stepper motors, since they offer relatively high torque and fine control. To generate lift motion, we considered using either a cam that would spin a partial rotation or a worm gear to produce linear motion. We decided to use a custom cam driven by the stepper motor through gears (see Fig. 8) since it allows for more customization. Additionally, worm gears were deemed unreasonably large since they must be mounted sideways relative to the motor, which increases the width of the lift arms and leaves less alignment room for the bin.

D. Onboard Processor

We explored performing offboard and onboard data processing, with a combination of microcontrollers and PC's, including the RPi, Jetson Xavier NX, and the Jetson Xavier AGX. The vision and navigation subsystems for our robot require a significant amount of data to be processed, which may not happen at the 15 fps laid out in our design requirements on the RPi, leaving the Jetsons or a PC as the only options to perform computation. Furthermore, the onboard processor must support multiple processes running concurrently in a real time setting, as multiple ROS nodes, written in C++ and Python, must all be running and communicating with each other through the publish-subscribe system. Hardware control processes must also be scheduled properly, as the processor must control motors for the bin lift system, communicating with the Roomba, streaming video frames from two cameras and processing LiDAR scans.

A PC would have to be wirelessly networked into a microcontroller onboard, which increases the chance of

failure, latency, and reduces autonomy since the Roomba would be reliant on data processing and communication from multiple hardware nodes. Because the Jetsons already have GPIO and plenty of computational power, it was the logical choice to meet our design requirements. We then needed to further decide between the AGX and the NX. The AGX provided faster computation, but drained more power. In addition, SSH development on the AGX was finicky. After testing, we found the NX's computation to be fast enough for the ROS nodes to meet their required publish rates, and the smoother development experience made the NX the clear choice for onboard processing.

E. Onboard Vision System

To actually sense our surroundings, we had to use some form of camera, LiDAR, or both. LiDAR has demonstrated success in room mapping, especially with acquiring depth information, but cannot provide the fine positioning data required for navigation to a bin for successful docking. As such, we decided to use both a LiDAR and camera. These will be mounted on top of the Roomba to have the best vantage point possible.

We could have gone with a single camera that faces the docking mechanism, but this means its view is obstructed (see Fig. 4) half the time (anytime we tow a bin), and cannot be used for positioning. Because the bin drop-off point needs to be at a certain location, we decided to use two cameras facing in opposite directions that allows for visual navigation all the time.As for the LiDAR, there were two models in the ECE inventory that we could have used, but to avoid a similar issue as the camera where a carried bin causes a significant obstruction, we decided to use the 360-degree version due to better handle obstructions.

F. Environment Mapping

The Environment mapping function went through several iterations. Initial research showed built-in MatLab libraries for LiDAR mapping to be sufficient for our application. We planned to export the MatLab library into C and port it onto the onboard processor. However, after further research and testing, the library seemed to be for stationary applications instead of live exploration. While this application could work due to our robot's low speed, we transitioned to ROS open source navigation libraries. There were two libraries that we considered, *hector_slam* [4], and *gmapping_slam* [5]. Both were viable, but we decided on using *gmapping_slam* because the *hector_navigation* library is not supported on the version of ROS (Noetic) we are using [6].

G. Robot Path Planning

Like the environment mapping system, MatLab was the first choice for path planning as it supported built-in libraries that can navigate based on the given output of the mapping library. However, due to swapping the mapping system to ROS, we decided that transitioning the map input to one that would work with the MatLab library would be inefficient and impractical. Thus we have decided to stick to a unified system for path planning and mapping. Keeping libraries within the same system decreases development time and has more supported features due to increased synergy and compatibility. We decided to use the ROS *navstack* [6] as it is fully supported by our ROS release unlike *hector_navigation* [7].

H. Bin Tracking

To track the location and status of each bin, we will be using a database in conjunction with ArUco tags. There were many options for tagging the specific bins. We first considered using April Tags due to their inherent uses in LiDAR applications, however due to the ease of use and compatibility of ArUco tags with the OpenCV library (which is the main onboard camera vision computation system), we decided to use them instead. We also considered different options of the stateful tracking of bins themselves. We needed some system to keep state in between runs, as such, we considered some system that writes to an external hard drive through some sort of file I/O interface, or using a database. File I/O would've involved writing our own file system interface and developing our own encoding / decoding scheme or using a higher level abstraction combined with some Python library such as pickling. This would've provided much stronger flexibility in handling bin storage for future applications but we decided that such flexibility was a low priority for the current project. Thus we decided on a more inflexible state tracking system involving a database, and building an abstraction interface library on top of that. The database streamlines fetching and writing concurrently and has sufficient features for the current goals of the project. Furthermore, there are many built in interfaces in Python to interact with a database of choice.

VI. System Implementation

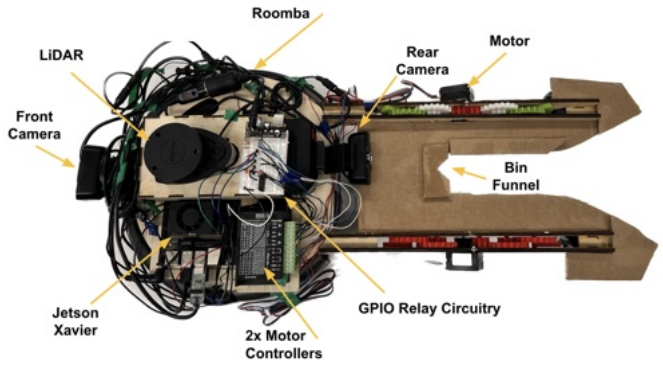


Fig. 6.1. Top-down photo of the final robot

A. Trash Bin Modifications

The trash bin itself needed to support omnidirectional docking as discussed previously, and necessitates a circular interface. To allow for variations in robot positioning while maintaining stability, we will be using a large disk with a slightly smaller diameter than the length of the trash bin, affixed to a larger circular base plate via a central column that

also provides room for the lift arms to slide underneath as shown in Fig. 6.2. All of these components will be made with laser-cut 1/4" and 1/8" plywood, joined together with wood glue. The upper and lower disks will both be 11 inches in

diameter. Because the lower disk has a larger footprint than an original bin, we actually increase stationary stability further. For our testing we will build two of these bins..

Fig. 6.2. Modified bin with ArUco tags and the elevated base

B. *Camera System*

The Roomba will be outfitted with two cameras, both oriented to be facing opposite directions, 180 degrees away from each other. The front camera will be responsible for identifying bins during pickup. As our robot's pickup arm is mounted onto the rear of its body, our second camera is mounted on the rear. The purpose of the rear camera is to gather information for proper alignment of the robot to the bins before the arm is activated for pickup.

C. *Docking Arms*

Our docking arms look like forklift arms, but will actually be static. They will be a box-like structure to provide rigidity, while allowing for components to be mounted inside, namely the stepper motors, gears, and cams for the lift mechanism. The arms will also have small caster wheels at the end, since a bin extended out on the arms will not be able to stay upright independently. These wheels will roll across the base plate on every docking sequence as shown in Fig. 6.3, which is accounted for with additional clearance between the arms and upper bin plate. The arms will be fixed to the top of the Roomba via existing screw holes.

Fig. 6.3. Side view of the lift arms midway through the docking sequence

D. *Bin Lift System*

The lift system uses stepper motors, gears, and cams to achieve our 44 N*m lift capacity and 4 cm lift height, requiring a 4 cm diameter cam. We are using 2 NEMA 17HS4023 stepper motors, which have a reported torque output of 13 N*cm, while being in a small form factor that

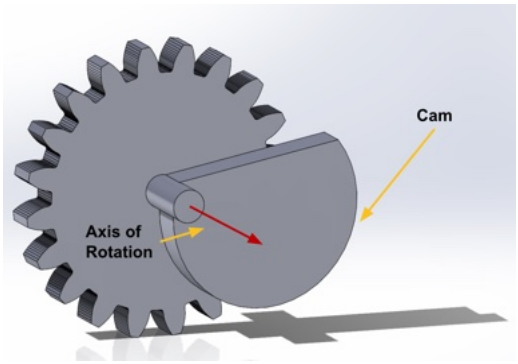


Fig. 6.4. Zoomed-in view of the gear with attached cam

will fit in our docking arms. The required torque to lift the bins is given by the following equation:

$$\tau = F \times r \tag{3}$$

where F is the force perpendicular to a radial from the axis of rotation, and r is the radius at which F is applied. Note that the cam will contact at a distance of $r/2$ away from the axis of rotation, cutting the lift height in half. We can then compute the required torque output of the lift system per motor, which is:

$$\tau_{out} = \frac{1}{2}(F_{bin} \times r_{cam}) \tag{4}$$

$$\tau_{out} = \frac{1}{2}(44\,N \times 2cm) \tag{5}$$

$$\tau_{out} = 44\,Ncm \tag{6}$$

The gear equation given below will determine what input-output tooth ratio is required to meet the torque requirement. This is because a high gear ratio (>>1) will provide a significant increase in output torque.

$$Ratio = \frac{N_{in}}{N_{out}} = \frac{\tau_{in}}{\tau_{out}} \tag{7}$$

Using this equation, we can calculate the required ratio:

$$Ratio = \frac{N_{in}}{N_{out}} = \frac{13\,Ncm}{44\,Ncm} = \frac{0.3}{1} = 3.38 \tag{8}$$

To add a buffer that allows for component friction and other non-idealities, we selected a gear ratio of 6.25:1. This is achieved with two sets of gears having a 2.5:1 ratio, to maintain a low profile and fit within the arms.

The cam is fixed off-center with its edge on the center of rotation of the output gear as shown in Fig. 6.4. For each motor, we have 2 output gears, each with a cam, allowing for a total of 4 contact points to lift from. This ensures stability of the bin and will prevent tipovers from occurring during the pickup process. The gears and cams are 3D printed since it allows for a high degree of customization, and the infill can be adjusted to balance cost and strength. This also allows us to manufacture the cam and output gear as a single piece which makes integration far easier.

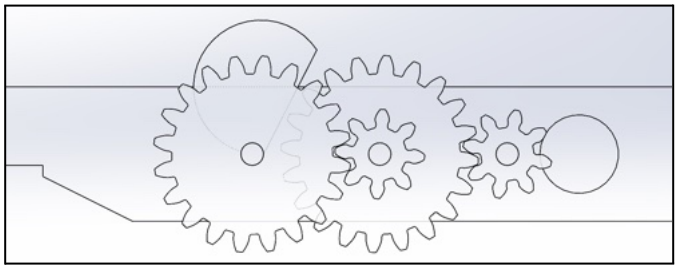


Fig. 6.5. Side-view of one half of a gear set in the static-raised position

The final element of the lift system is a static resting position for the cam in the raised state. This is achieved by allowing the cam to rotate slightly past vertical, with a cut that rests on the other large gear as shown in Fig. 6.5 above. This allows us to turn off the motors when the cams are in the raised position, conserving power and extending the runtime to help meet our 5-hour runtime target.

E. Drive System

Our drive system is built on top of the iRobot Create2. This robot is a Roomba modified with a serial connection for manual control. Using an included USB to serial cable, we will control the Roomba's movements from our NVIDIA Xavier NX using the ROS library *create_robot* [8] which supports tethered driving. This library creates a ROS node that provides a listener on the *cmd_vel* topic, and a publisher on the *odom* topic, both vital to interfacing with the ROS *navastack*. Additionally, these topics allow the coordinator to publish velocity commands to the Roomba over the cmd_vel topic during the bin alignment procedure.

F. Onboard Vision System

The vision system uses Python's OpenCV library to handle bin identification, location as well as rudimentary distance sensing. The vision system greatly depends on the camera used, and thus should be calibrated on the software side to handle distortion effects [9]. The camera is used to identify ArUco tags in the environment and communicate ArUco coordinates in relation to the robot to the central logic hub. ArUco tags also provide unique identification numbers which can be read with OpenCV. This system also uses the ArUco tags to differentiate between the different possible orientations of a trash-bin in relation with the robot itself (by assigning a different ArUco id modulo the number of orientations).

Our LiDAR system is implemented using the Slamtec RPLIDAR A1M8. The LiDAR will be used for mapping, localization, and obstacle detection, and be connected directly to the Jetson Xavier NX using an included USB adapter. Due to the height restriction of our robot, which is necessary to drive under desks, the mounting position of the LiDAR will bring the trash bin in alignment with the LiDAR sensor. This creates an obstruction that the LiDAR will interpret as a wall, messing up localization. To avoid this issue, a mask will be applied to the LiDAR's output. This decreases its FOV, but this is a reasonable tradeoff as it makes localization and implementation much more straightforward. We further discuss this in section J.

G. Environmental Mapping

We will be using SLAM to create a 2D map of the robot's office environment. Specifically, we will be using the ROS module *gmapping_slam* which constructs a map from 2D *laser_scans* provided by the Slamtec RPLIDAR A1M8 and odometry data provided by the Roomba. In order to create the initial map, a one time setup is completed, which requires a manual exploration of the office space by remote controlling the Roomba until a satisfactory map is created.

H. Navigation System

We use a ROS open library called *navstack* for our navigation system. The *navstack* is a conglomeration of ROS modules each that pertain to a specific task. Critically, it includes the *amcl* package, which is responsible for localizing the robot, the *global_planner* package which creates the path from robot to bin locations, and the *basic_local_planner* package which handles following the overall path while taking into account the environment and obstacles. The high-level components that the *amcl* subscribes to are the scans from the LiDAR, the pregenerated map, as well as the Roomba's odometry data.

It utilizes these readings, alongside an adaptive particle filter to estimate the robot's position. With it, it generates a global shortest path to goal destinations through the use of Dijkstra's. Local cost planning is used for obstacle avoidance. The navigation system's output will then be routed into the state coordinator system for control of the robot.

Due to the nature of the *amcl* package, if significant slippage occurs, the orientation of the Robot in relation to the room can become misaligned. To combat this issue, we implemented tuning of the *navstack*, which required us to adjust the max and min linear and theta velocity of the robot, the robot's footprint, the inflation radius of obstacles, weightage given to the noise of odometry and LiDAR scan data, map update frequency, exponential cost functions for obstacles, and goal tolerances. All of these parameters are dependent on the robot implementation as well as the environment in which the robot is intended to navigate within.

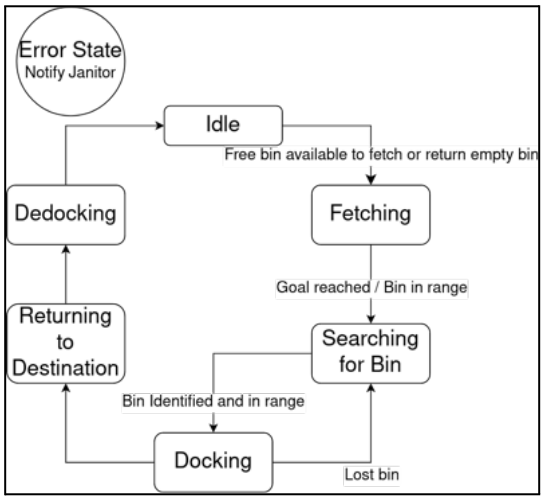
I. State Coordinator

To coordinate all of the tasks that the robot needs to accomplish, a central decision maker is necessary. To keep our software stack unified, this system will be implemented as a ROS node in Python. The control flow logic must accomplish the task set out by the use-case requirement, and thus different states are needed to track necessary logic and metadata. The state machine is responsible for tracking additional meta information such as current goal bin, the current goal location to reach, as well as the robot's current position. Initial information about bins are fetched from the database. Due to stateful behavior and the necessary tracking of metadata, the coordinator is implemented as a Python class. Depending on the state, the coordinator will subscribe and/or pull information from the LiDAR, and two web cameras to decide on state logic. To receive information from the LiDAR, the coordinator subscribes to the *scan* topic, and

to get information from the webcams it uses a Python video stream from OpenCV. To communicate with other components, it publishes movement directives to rostopics that the Roomba's movement system subscribes to, in order to control the bin alignment system. The coordinator also uses the custom *Arms* class to control the motors for the bin pickup system. Furthermore, it is responsible for publishing and canceling goals to their respective rostopics that the *navstack* subscribes to. Similarly, the coordinator subscribes to a transform topic published by the *navstack* to track where the robot's current coordinates are. State changes are done after one task is accomplished and can be seen in the state diagram below (Fig. 6.6).

The state coordinator uses the *rospy* framework to interface with ROS nodes. We use *rospy* to register callbacks with different topics. For example, we listen in on the robot's current orientation and LiDAR scans. These sensor fields are both used during bin docking, but our coordinator uses a single process with no mutex locking, which introduces a challenge for obtaining updated sensor information in a thread safe manner. We rely on the Python GIL (Global Interpreter Lock), which only allows a single Python thread to be running at a given time. This means that to get an updated LiDAR scan, we need to sleep the coordinator thread, which drops the GIL, allowing the LiDAR scan callback to acquire the GIL and update the distances array object. Then, it sleeps again allowing the coordinator to take back GIL, and access the LiDAR scan distances array object without data races.

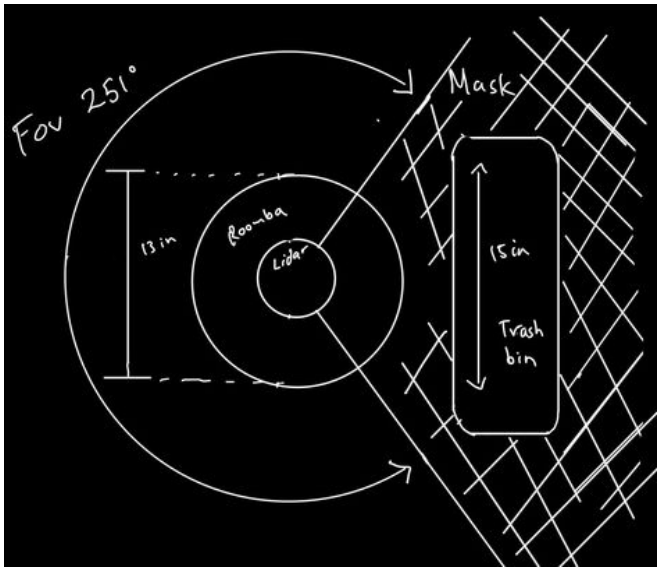
Fig. 6.6. Software state coordinator functional diagram



J. LiDAR Masking

In order for the navigation to work properly while the robot is carrying a bin that can block the back field of view of the LiDAR, masking is necessary, which may be perceived as a local obstacle by the *navstack*. Furthermore, due to the proximity of the bin while the robot is carrying it, it intersects with the robot's footprint, creating a negative costmap value, which can stop navigation entirely. In order to avoid this problem, we mask out the back of the LiDAR scan such that the *navstack* has no view of the potential area a bin will be in. This is accomplished through the use of a ROS node written in C++, which subscribes to the LiDAR and transforms the information into a 251° view in front of the robot as shown in Fig. 6.8, then publishes to a different topic, which the *navstack* subscribes to. Due to frequent publishing and the crucial nature of this information for the entire *navstack*, low latency was a must. The lower-level control and speed of C++ was chosen over Python for the success of this proxy node.

Fig. 6.7. Top-down diagram demonstrating the masked-out region of the LiDAR



K. Bin Tracking

Tracking bins is a stateful protocol, and must be maintained between sessions. It must also be scalable and have concurrency control for future scaling. As such, a lightweight database with standard Python library support is perfect for our use case. Thus, we chose to go with SQLite, which is part of the standard Python library. The database tracks bin locations in the mapping environment, whether the trash needs to be taken out, each bin's identification number, whether the bin is missing, the bin's location for the janitor to pick up and the bin's home location. The database interfaces with the backend logic system on the onboard processor through a custom Python interface that handles bin tracking logic, including tracking bin locations on drop-off, pick-up, and spillage of bins, fetching current free bins, fetching bins to take back as well as successfully emptied bins.

L. Electronics Power System

As for our power system, the primary concern is delivering enough current at the appropriate voltages to keep all components active, while avoiding brownouts or current-limiting conditions. See the table below for the power requirements of our various components. Note that the currents are maximum ratings, and are likely less under normal operating conditions.

TABLE I. SYSTEM POWER REQUIREMENTS

Component	*Voltage (V)*	*Peak Current (mA)*
Nvidia Jetson AGX (10W mode)	12	~850
USB Camera (x2)	5.0 (USB)	70
360-degree LiDAR	5.0 (USB)	100
TB6600 Drivers (x2)	3.3/5, 9-42	1500

With this analysis, we can determine that the peak current needed is 4250mA, which is the sum of the Jetson and stepper drivers, but not including the USB devices. This is because the 10W Jetson power mode includes the running of USB peripherals, which itself has a rated max output current of 500mA. With these requirements, we believe a battery >20Ah of capacity and a 12V DC output will be more than enough to meet our needs for a 5-hour runtime.

The Jetson's GPIO pins drive output at 3.3V, which is sufficient for the TB6600 drivers, but the current output from the Jetson is insufficient to maintain the control signal voltage. As a result, we designed a relay circuit that makes use of a 12V-to-5V voltage converter to provide ample current at sufficient voltages to control the stepper (Fig. 6.8). The relay circuit has independent controls for the enable pins of the drivers, but the direction and pulse controls are shared for ease of use.

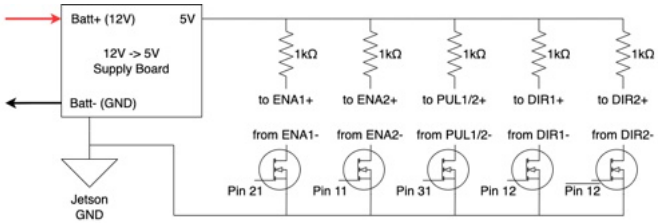


Fig. 6.8. Schematic of the stepper driver relay circuit

M. Bin Lift Software System

The bin lift arms are controlled by using the *Arms* Python class. The Python code controls the lift arms by using the Jetson's GPIO pins to send signals to the motor control relay circuit. The class itself exposes method calls to raise and lower the bin, and keeps track of the current state of the arms (whether the arms are in a raised or lowered position) to protect the gear system from over-rotation. Originally, we hoped to use a dedicated PWM pin made available by the Jetson, but due to non-functional PWM pins on the Jetson (and many other broken pins), we had to implement software PWM control, where we turned a regular GPIO pin on and off in a tight loop with manual process sleep commands.

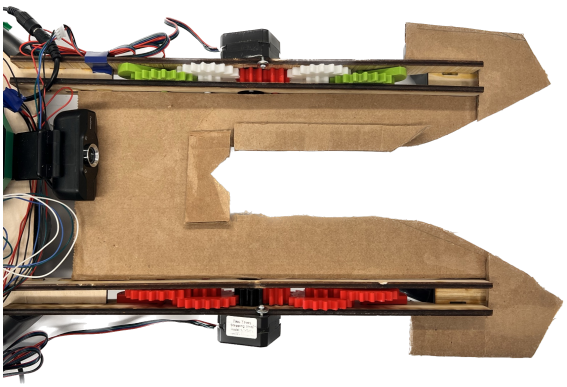
Due to shifts in the gears when docking, the left and right gear lift systems may become out of sync when initiating a lift operation. This mismatch may cause a bin lift failure by not raising the cams enough. To correct this drift, we over-rotate the gear system on lift, causing the cams to come into contact with the primary reduction gear, ensuring all gears are in the same position as shown in Fig. 6.9.

Fig. 6.9. Side-view of the bin platform on the cams in the raised position

N. Bin Alignment Hardware System

We originally planned for a guideless docking system, but found the alignment system to be unreliable. To mediate this, we added an alignment funnel that pushes the bin into the correct position for lifting. Specifically, we wanted to avoid issues when a cam wouldn't be under the upper plate of the bin, which would cause a lift failure. The funnel also accounts for different bin dock orientations by having an extra divet at the back of the funnel, and increases lift success by increasing the angle from which the robot can approach.

Fig. 6.10. Close-up view of the alignment funnel located between lift arms



O. Bin Alignment Software System

As denoted in the state diagram (Fig. 6.7), when the robot is in the process of traveling to a bin position, it periodically checks both the front and rear cameras for the presence of an ArUco tag. Each bin has some associated ArUco tag ids, if the current bin to pickup's associated ArUco tag is recognized by the cameras, and the tag is less than 110 cm away, then we transition to *docking*. If no ArUco tag is detected, then the coordinator waits until the goal position is reached, and then transitions to the *searching for bin state*. In this state we rotate in place checking the cameras for detected ArUco tags. If we are unable to find the bin, then we set the bin's state to missing in the database and return to the *idle* state and begin searching for other bins to pick up.

Once the docking state is entered, we first check if the front camera has detected the tag, if so we align the camera to be in line with the center of the tag, then initiate a 180º turn to position the back camera. At this point, we enter a loop where we first check if the back camera is lined up with the tag, and then if so move forward a set distance. We have a vector that denotes the max left and max right alignment of the tag at different distances from the tag (and use LERP for in-between distances). This is necessary because as the robot approaches

the bin, the max left and right positions of the bin necessarily shift as the arc length that the tag can cover decreases. Additionally we found that waiting 0.25 seconds between orientation adjustments, greatly increased docking success.

Once we get to about 56 cm away from the bin, we transition to using the LiDAR scanner to detect the distance from the robot to the bin, which is denoted by the minimum distance reported by the LiDAR inside a 47º FOV to the rear of the scanner. We slowly move backward towards the bin, making sure to sleep in between movements to run LiDAR scanner callbacks and stop momentum between distance calculations. We decide the final stopping distance based on the orientation of the bin. The bins have different ArUco tags on different sides enabling us to detect the orientation. If the bin is oriented with the wider side to the robot, we move back until the minimum distance is 27 cm, if the thinner side is oriented to the robot, we move back until 22 cm. Finally, we send one last move back command to ensure the bin stand is fully contained by the bin funnel, at which point we lift the bin and prepare for returning the bin.

VII. Test, Verification and Validation

A. Physical Constraints

We imposed several physical constraints on the dimensions of the robot, to ensure smooth performance within an office environment. We specified a height of 30 inches, less than three feet wide and 3 feet long to fit between and under desks. After measurement with a meter stick, we found that our robot had a height of 8 inches, a length of 30 inches (2.5 ft) and a width of 15 inches (1.25 ft). These fall within the constraints.

B. Battery Life

We specified a working period of 5 hours. During testing, for a non-stressed system, we were able to drain 80% of the battery bank within 4 hours, extrapolating to a success in that case. However, during larger stress tests, with constant movement, we lost 23% battery over the course of an hour, which extrapolates to a failure, and another stress test with non-consistent use of motors and *navstack* caused the system to run out of battery after 4 hours and 11 minutes. Another full stress test that ran continuous movement, with *navstack* and bin lift netted a decrease of 67% over the course of 2.65 hours. This is extrapolated to failure as well. We are surprisingly bottlenecked by the onboard battery bank that provides power to the processor, and the motors rather than the Roomba battery itself. One note we need to make is that our theoretical assumption was that the processor drew 10W of power, however we adjusted the configuration to be a 20W setting for faster computation. That may explain the offset in theory and application, though it is also possible that our theoretical guess does not account for all real world variables. This is another case of a trade-off we made, favoring computation over power draw.

C. Camera FPS

We chose hardware components that hit the 15 fps we laid out, and verified with a fps tracker that both front and back webcamera's support an average fps of 30. This matches the hardware specifications, and matches our theoretical prediction.

D. Sensing Accuracy

We require a distance measurement accuracy to be within 90%. We found that the exact distances measured by the webcams to be relatively useless, as logic surrounding them is based more on the consistency of camera readings rather than how accurate they are. When we require more accurate distance sensing such as when to decide if a bin is sufficiently within the funnel during docking, we swap to the LiDAR's readings. Nonetheless, we found both sensing systems to be relatively accurate. We did five trials, with each trial placing a trash bin with an ArUco marker at different distances, and read the sensor data and applied equation (2) to our readings to find a LiDAR accuracy on average of 99.6% and web camera average accuracy of 92.3%. This sufficiently falls within the 90% we set out for.

E. Bin Identification

We specify that the bin identification must be correct at 90% for up to 2 meters away, followed by a higher accuracy of 95% for distances from 0.15 m to 1 m. Through testing, we found that we actually achieve a 100% accuracy through the entire range from 0.15 m - 2 m. We achieved this result by putting various ArUco tags at 0.15 m from the sensing device, then 0.1 m - 2 m in 0.1 m increments for 10 trials each. This took some tuning however, as we needed to calibrate the camera as well as change the background on which the ArUco tags were posted on. We added a white sheet of paper as the background of the ArUco tag, as a raw tag on the black bin had an extremely low success rate for bin identification because there is little differentiation between the black of the tag and the bin.

F. Bin Pickup Weight

We originally specified that our system must be able to lift a load of 10 lbs, as we estimated that the full trashcan of an average office worker to be around that amount. This amount is estimated to be from left-over food, but mostly wrappers which are light in nature. The heavy items are unfinished drinks which can be 1-2 lbs each, but we suspect that most office workers are not tossing copious amounts of unfinished drinks. Through iterative testing, we found that our bin lift system cannot lift anything beyond 6.1 lbs. We did iterative testing from 3 - 6.1 lbs, with 5 - 8 trials each. We stopped at 5 trials for weights that consistently failed, and went to 8 for consistent success.

We had a 100% success rate for weights up to 4.1 lbs, and found a drop off at 5 lbs to 75%. We considered the test a success if the bin was lifted far enough off the ground to be carried around by the robot. Unfortunately this does not hit the 10 lb goal we set out with. Our theoretical conditions

showed that in ideal conditions with maximum tolerable voltage (42V) we could lift 18lbs. Our 4-pound consistent lift capacity is a far cry off that goal, which we believe is due to both friction in the system and our lower motor operating voltage of 12V.

G. *Bin Docking*

We also chose to test the docking system as a subcomponent itself. We wanted to achieve a 90% success rate to hit the 85% success rate for full integration testing. To do so, we did 5 trials of placing bins at different angles with respect to the robot, for multiple different configurations, tuning the linear and rotational speed of the robot during the docking process.

The final configuration we settled on had a 100% success rate within the 5 trials, and we noticed that generally as we increased the speeds, we had worse results. We added pauses to allow for the robot movement to settle down, and the lower speeds meant the effect of the arms' angular momentum on the rotation accuracy was reduced.

Also, we were only able to achieve our success rate after adding a funnel to the arms, as before adding the funnel, the tolerance of the top base of the trash bin and the cams were too small at 1 cm for any consistent ability to dock. The funnel was able to get us from virtually no success to 100%, when combined with the lower movement speeds. The trade-off introduced here is the more mass we add to the arm, the slower our rotation could be, increasing the amount of time it took for a successful docking to happen. However, we chose to opt for consistency over speed.

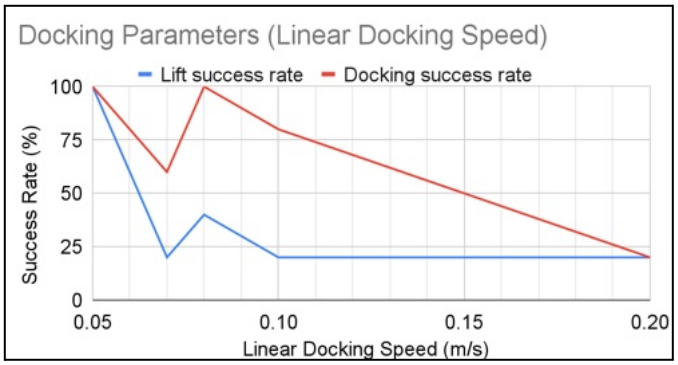


Fig. 7.1. Test results of linear docking speed versus docking success rate

H. *Mapping Accuracy*

Qualitatively, our map resembled the two environments we ran it in. We, however, did notice that the LiDAR scans did not work well with windows, as the scan will usually project through them. As such, maps generated in the HH1307 room were quite poor due to the low windows of the room. However, a more traditional work area such as A101 provided a much more accurate map.

We did not make a serious attempt to quantitatively measure the accuracy of the map as our use-case solution involved working flexibly without needing to remap the office

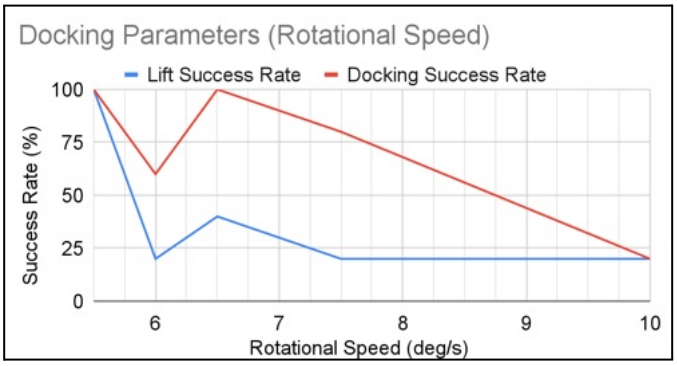


Fig. 7.2. Test results of rotational docking speed versus success rate

every day, where furniture and equipment can be moved, and thus a general map with the overall structure must suffice.

I. *Path Planning with Goal Tolerance*

To ensure our navigation stack works properly, we want the robot to reach its goal within a 0.5 m radius. The *navstack* has a parameter for this, which we tuned to be lower than the 0.5 m specification by setting a goal tolerance of 0.2 m.

We ran navigation testing on two goal locations with 5 trials each, both with one turn, although one was further and one was closer. For the closer one we achieved an average offset of the center of the robot from the goal location of 0.2 m and 0.21 m from the further one. Notice that we always remained within the 0.5 m specification we set out with, but stray just a bit from the set goal tolerance of 0.2 m. This is expected from the accumulation of slippage over the course of a trip. This is another reason why we set our goal tolerance to be much lower than 0.5 m to account for such a possibility.

This value was tuned up from the original 0.15 m, as lower goal tolerances mean longer overall trip times. With a low tolerance, the robot may get stuck trying to get within the goal even if it is off by just a couple of centimeters. Due to the nature of the application however, we decided that increasing it would improve the consistency and lower the time taken for the robot to complete its goal. This brings up a trade-off between accuracy and speed, and given our use case we decided in this case accuracy was a worthy loss as it still fell within our design-case requirements.

Next we look towards verifying our use-case requirements are met through a series of integration tests between our subsystems.

J. *Results for Collision Testing*

Apart from full integration testing, we did separate trials for navigation with human obstacles. We did 20 trials, moving the robot between 2 different locations with 2 human interventions to see if the robot is able to stop itself from colliding with the humans. We had 19 successful trips and 1 failure, where the robot collided with the first human, resulting in a 95% trip success rate. This hits our use-case requirement of 95% that we set out for. Further testing in integration trials were run with humans and we found that no human collisions happened within the 20 trials done there.

K. *Results for Movement Speed*

Extrapolated from the use-case study of the Meta NYC

office, we found that a movespeed of 0.21m/s would be required for the worst case of all 90 bins needing to be taken out in one day in addition to having some buffer. Through testing of 5 standard trials we found that the robot was able to move at a speed of 0.235 m/s by timing how long it took for it to cover a distance of 3 m. This matched our theoretical prediction of how fast the Roomba could move. We do note that this speed is not always hit during trips. During integration testing, we found that navigation with 0.22 m/s did not lead to any decrease in success rate in comparison to slower speeds of 0.1 m/s. This was done with navigation over 10 trials at each configuration. We do acknowledge that while navigation speed is able to meet the use-case requirement, the speed of the robot during docking processes is kept to a minimum to ensure smoother and more accurate control.

L. Results for Full Trip Integration Testing

We set out with the goal of hitting a round trip success rate of 85%, which is made up of various sub component success rates. We require 95% of trips to have no collisions, 99% of trips to have no spillage, and a 90% bin docking success rate. To get our results we ran 20 trials, with 5 trials each of different bin locations and robot movement speeds. From these 20 trials, we achieved a 80% success rate, with 4 failures total. We had a 100% success with bin spillage, as no trash bin fell off the robot during any of the trips, two docking failures, one collision failure, and one navigation failure. This means we hit the docking success rate of 90%, and a 95% collision prevention success rate. However, due to the one navigation failure, we still ended up just below our goal.

We decided that this continuous run of 20 trips was a good representation of a full integration test, as our tests were run with the system continuously on, just with the same bin taken to 2 different locations after each trip. We found that this test had an average round trip time of around 3 minutes, which if extrapolated means that we are under the time limit for 90 bins over 5 hours. For 90 bins over 5 hours means a round trip time of 3 minutes and 20 seconds. However, we also acknowledge that the office space that we tested in is much smaller than the Meta use case, and thus our system could take much longer.

We also did a full software loop of the state diagram test with a single bin, where a bin is fetched, brought back to take out position (the bin drop off location), then picked up and brought back to the original position (the desk location). This mimics the full work flow of the robot, and we found that the current state of the robot is able to complete one full state transition loop. The point of this test was more to see how the control logic flows rather than testing the other navigation components as we felt that those had been sufficiently tested at this point.

VIII. Project Management

A. Schedule

The schedule is color coded based on the team member that is primarily responsible for the task, with color combination involving shared tasks. Tasks were usually done in one week periods with a steady progression of subsystem bring up prior to March 3rd 2023. While that was our initial plan, and we followed our schedule strongly for the first half of the project, the second half of our project encountered significant setbacks, which forced us to use a lot of our slack time, and even fall behind at points. Full integration fell to the end of finals week. Hardware freeze was a setback, and navigation tuning took much longer than expected. Further changes and iteration on hardware forced repeated software tuning which further delayed development. The full schedule can be found by the Gantt chart showcased by Table III.

B. Team Member Responsibilities

Team members' responsibilities are split by team member backgrounds.

Mason has a strong hardware background, giving him default responsibilities on hardware-related tasks. This includes 3-D modeling, designing, and building the docking mechanism. His embedded background also means he has partial responsibility in embedded communication between subsystems of the robot, specifically power distribution and stepper motor controls with GPIO.

George and Jack both come from software-oriented backgrounds, relegating most software tasks split amongst them. While Jack has the primary responsibility of LiDAR bring up, and George is responsible for the bin tracking backend with a database; more complicated software systems such as path planning, CV, and scheduling logic have shared responsibility between them.

Integration is the partial responsibility of all team members due the embedded communication and software and hardware nature of the project.

While individual team members have primary responsibilities, each is responsible for the success of the project as a whole. This means that we each are responsible for helping in the case that challenges arise for tasks.

C. Bill of Materials and Budget

Please refer to Table II at the end of this document for the bill of materials and their associated costs. We ended up not using some of our materials because they were only backups for other systems (like the AGX) or prototypes for 3d printing. The one item we had to purchase post design report were the caster wheels which were necessary to add stability to the lift arms.

D. Risk Management

Our robot will have numerous systems, any of which could fail. It is vital that we have backup systems and implementation plans in place in the event that this occurs.

During the initial stages of our project, we ran into connectivity issues with the NVIDIA Xavier AGX. The AGX would sometimes disconnect from our secure shell session making it difficult to develop. These issues were resolved by reflashing the AGX and switching to an ethernet cable instead of a USB cable for SSH. We have many software components

to tackle, so the developer workflow is vital to the success of our project. If we were to run into an unrecoverable situation with the AGX, our contingency plan was to utilize the Xavier NX which prior teams have found less problematic. Halfway through the semester we ended up utilizing this fallback, and decided to swap to the Xavier NX, which greatly improved development efficiency and wireless SSH performance. Luckily, we reserved the NX early, making the swap painless as we didn't need to wait for the hardware component.

We also developed a backup plan for the LiDAR system. We are currently using the Slamtec LiDAR A1M8, which accurately identifies solid objects such as desks, frames and walls, but may not correctly identify other objects like desk chairs. Therefore, we have also acquired the Intel RealSense LiDAR Depth Camera L515 as a backup. In case our original LiDAR does not provide accurate obstacle detection, our implementation would have pivoted to the L515. However, in the end, the LiDAR A1M8 was sufficient for our application and we did not have to pivot.

Another risk that we have addressed is the Roomba's power supply. The Roomba itself is powered by an internal battery pack. This battery pack is old, and may have lost performance. To meet the use case requirement of running the Roomba for 5 hours, we have located aftermarket battery replacements that provide additional range, and ordered it as well. Doing so, we replaced the old Roomba battery to have a much longer battery life. However, we did not foresee that the battery bank for the processor and motors was the actual bottleneck. Although we had plans to address this, by utilizing 2 batteries instead of 1, or getting a larger battery, we did not have time to reimplement design changes to incorporate the additional battery or buy another one.

Finally, to address the risks in the SLAM and navigation stacks, we analyzed past projects that have implemented robot localization and path planning using overhead camera systems. We believe that this technology could be adapted to our problem space, and identified this route very early. We had plans to pivot to an overhead camera configuration if we could not have gotten the *navstack* up and running. Additionally, our team has previous experience with this type of system. Fortunately we were able to get the open source *navstack* up and running, though it might have been helpful to incorporate stop losses in our schedule for pivoting to other solutions, as it took us nearly 3 weeks to tune the *navstack* to a working state.

Unfortunately we did not have a risk mitigation plan for bin docking, and had to engineer solutions on the fly near the end of our schedule during iteration testing. This constrained us into the types of solutions we could attempt especially given the constrained time-frame and resources near the end of the semester.

We found our budgeting strategy was relatively successful albeit a bit inefficient. We ordered the main components we needed early, leaving a lot of buffer for unforeseen circumstances. This, however, left us with some budget left over that we could have used to potentially buy additional components to implement such as a larger battery bank.

IX. Ethical Issues

Our primary ethical concern is that we want our robot to assist janitorial staff, rather than replacing them. We understand that as robotics become more and more advanced, some jobs will eventually be replaced, but we hope that our design lives symbiotically with janitors, as they will be responsible for managing the autonomous system, as well as maintaining its operations.

Another concern that may be raised is the operation of the robot in corporate offices, where trade secrets may be present and privacy is necessary. Our robot operates in the evenings when there are few people around, so there will be minimal disturbances and the onboard cameras will not be seeing private information. Additionally, the only pieces of data we store are the bin positions and the pre-generated map so there is no way to access any visual information captured, unless our system is infiltrated.

For the issue of infiltration, our system can function on internal networks, so any malicious override attempts would have to first bypass the security measures put in place by the company who uses our system. Ensuring adequate network security is not the responsibility of our robot, but it would be a recommended item to have for any user.

X. Related Work

Autonomous robots that are used to move items around are not unique, especially as robotics rapidly develops. Thus, there are quite a few predecessors to our implementation albeit designed for different use-case scenarios and purposes.

Perhaps the first implementation when we think of autonomous item movement is Amazon warehouse robots. Amazon, the world's leading online retailer, deploys over 520,000 drive units of its proprietary robot named Proteus. They are warehouse robots designed with navigation and perception features to move GoCarts (warehouse storage containers) [1]. These are designed for warehouse usage which includes rugged industrial design choices that are unfit for the office applications that our project is geared towards.

Similar to swarm warehouse robotics, there was a project that tested the feasibility of swarm applications called PARROT. The implementation used several robots to transport pallets in parallel, offering large speedups [10]. Although there are similar aspects in perception and path planning, we note that their implementation focuses on scale of parallelization in a proof-of-concept fashion, and is not applicable nor actually usable in the office environment.

Another application that is similar in use case and technologies to our implementation are the food delivery robots we see on the streets, with the most notable example called Starship. Such robots use different perception technologies than ours, and they do involve heavy path planning. They are also designed for much more rugged outdoor environments and as such their form factor would not fit well in the open office environment.

In a familiar vein, a robot that uses similar technologies and has to do with garbage pick up and cleanliness rests in Recycle Bot, an autonomous robot that uses LiDAR, CV and a Roomba drive system to identify plastic bottles on the ground to pick up and store on its onboard container [11]. While thematically similar and using similar technologies, our use case is quite different from theirs, and as such the mechanical portions of our designs differ greatly.

We notice that there are a great deal of applications that are geared towards moving something around to make human lives easier. We find that as such our implementation is but another iteration, or rather step, in the frontier of robotics.

XI. Summary

In all, our design uses proven systems and combines them to provide a solution that can greatly streamline and improve the logistical challenges of maintaining large open offices. The autonomous nature of the robot means that a one time setup of this robot in a given office greatly improves efficiency and gets rid of a large menial task for custodial workers, and can be easily scalable to cover multitudes of areas.

Nonetheless, while the benefits scale greatly, we faced challenges in implementation both due to the technical nature of the task as well as our inexperience. We faced large challenges with working with ROS as well as integrating the open source resources we are using.

As a result, while we met most design and use-case requirements, we failed others. Some design requirements can easily be fixed with additional time and resources such as buying a bigger battery bank, however others require more experience and larger software changes. For one, perhaps a lift system that raises the bin from directly on top of the robot instead of using arms could have made localization better due to less slippage of the arms, as well as increased docking abilities. Furthermore, stronger motors could've been used to help us reach the bin weight lift goal. If given the time, we could've also opted to add an IMU, to help with slippage, and also a more accurate robotic base instead of the Roomba could have provided better odometry data.

Over the course of the project, we learned a lot of lessons. Tuning ROS parameters as we worked with them took a significant amount of effort, and we believe that setting up a simulated environment could've greatly increased efficiency in tuning. We also learned that hardware design is incredibly important in streamlining software logic. In hindsight there may have been different form factors to change to greatly decrease the software complexity. This was learned from the alignment of the arms, as we found that adding a funnel made a software task possible that was originally near impossible due to the poor motor controls. There is a balancing act that must be struck with balancing hardware implementations with software complexity, as hardware itself also faces immense challenges.

We found it incredibly difficult to bring up the motors for the arms, as it took us an entire day to find an unexplainable bug in the relay circuit added to drive the motors. Increasing hardware complexity introduces more points of failure and increases the number of components to fabricate, greatly slowing down the development process. We also learned that hardware freezing as soon as possible is important. It allows the software stack to be tuned more efficiently because whenever hardware components were added, some retuning needed to be done on the software side to account for the change of design.

While we are all graduating, we like to think that there are still many components that can be improved. The software state machine could become more robust to include better error detection and state transitions. We also think that iterating on the design is important and can make the system more reliable.

Nonetheless, we learned and grew a lot, facing many challenges along the way, allowing us to end up with a working implementation. However, we believe that while we've built a working foundation, with more experience and time we could've made the system much more robust. With further tuning and design iterations we believe that our solution can be a flexible and robust implementation that can be applied to thousands of offices across the world. In addition, the modular nature of software and robotics means that scaling the implementation and adding additional use cases is not only feasible, but easy to do. We believe our solution to be one that can provide great efficiency and betterment for the modern workforce, and is another step in the robotic revolution.

Glossary of Acronyms

CV – Computer Vision
FOV – Field of View
FPS - Frames Per Second
GIL - Global Interpreter Lock
GPIO – General Purpose Input/Output
LERP - Linear Interpolation
LiDAR – Light Detection and Ranging
ROS – Robot Operating System (software package)
RPi – Raspberry Pi
SLAM – Semi-autonomous Localization and Mapping

References

[1] Jed John Ikoba, "Amazon announces the Proteus, a fully autonomous warehouse robot", Accessed on 3/3/2023, [Online]. Available: https://www.gizmochina.com/2022/06/23/amazon-proteus-fully-autonomous-warehouse-robot/

[2] Meta Floor Plans, Accessed on 3/3/2023, [Online]. Available: https://www.vno.com/office/property/770-broadway/3311677/landing

[3] "Camera Basics for Visual SLAM", Accessed on 3/1/2023, [Online]. Available: https://www.kudan.io/blog/camera-basics-visual-slam/#:~:text=The%20ideal%20frame%20rate%20for,fps%20based%20on%20the%20application.&text=There%20are%20ways%20to%20increase,based%20on%20the%20use%20case.

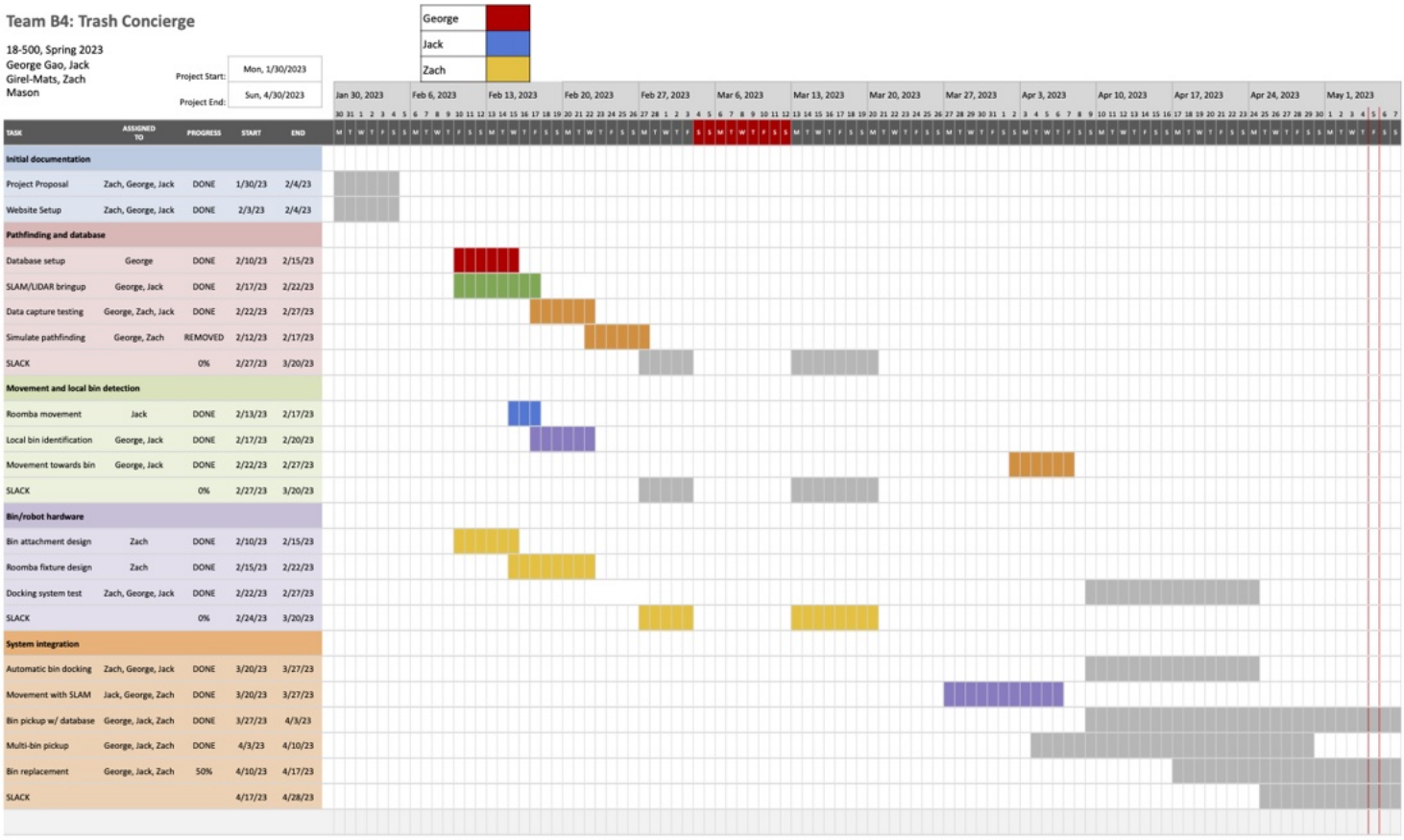
[4] "hector_slam" Accessed on 3/3/2023, [Online]. Available: http://wiki.ros.org/hector_slam

[5] "gmapping" Accessed on 3/3/2023, [Online]. Available: http://wiki.ros.org/gmapping

[6] “navigation” Accessed on 3/3/2023, [Online]. Available: http://wiki.ros.org/navigation
[7] “hector_navigation” Accessed on 3/3/2023, [Online]. Available: http://wiki.ros.org/hector_navigation
[8] Perron, Jacob. “create_robot.” ROS Wiki, 4 October 2022, http://wiki.ros.org/create_robot. Accessed 5 May 2023.
[9] Fernando Souza, “3 Ways To Calibrate Your Camera Using OpenCV and Python”, Accessed on 2/24/2023, [Online]. Available: https://medium.com/vacatronics/3-ways-to-calibrate-your-camera-using-opencv-and-python-395528a51615
[10] Prithu Pareek, Omkar Savkur, Saral Tayal, “P.A.R.R.O.T: Parallel Asynchronous Robots, Robustly Organizing Trucks” Accessed on 3/3/2023, [Online]. Available: http://course.ece.cmu.edu/~ece500/projects/f22-teama2/wp-content/uploads/sites/211/2022/12/Capstone_Design_Report-2-compressed.pdf
[11] Meghana Keeta, Serena Ying, Mae Zhang “RecycleBot” Accessed on 3/3/2023, [Online]. Available: http://course.ece.cmu.edu/~ece500/projects/f22-teama4/wp-content/uploads
https://www.irobotweb.com/-/media/MainSite/Files/About/STEM/Create/2018-07-19_iRobot_Roomba_600_Open_Interface_Spec.pdf

Description	Manufacturer	Model	Quantity	Cost (Dollars)	Not Used
NEMA 17 Stepper Motor	Twotrees	Twotrees-16565	3	$27.59	
TB6600 Stepper Driver	OUYZGIA	200205003	2	$29.47	
7 Gallon Trash Bins	AmazonCommercial	B08PDV3YY7	2	$23.08	
USB C to 4x USB A adapter	Keymox	B0835L59N2	1	$9.55	
24Ah battery, 12VDC & USB out	SinKeu	HP500S	1	$99.99	
Jetson Computing Device	NVIDIA	Xavier AGX	1	~$1900.00	X
2D, 360 degree LiDAR system	SlamTec	A1M8	1	$99.99	
Programmable Roomba	iRobot	Create2	1	$199.99	
1080p Camera	TedGem	CE0140_01	2	$27.80	
Small Gear	Mason/TechSpark	v1.26/27	2	$2.00	
Cam Gear A	Mason/TechSpark	v0	1	$10.00	X
Full Gear Set	Mason/TechSpark	v1	1	$23.50	X
Additional Gear Set	Mason/TechSpark	v1.1	1	$4.50	X
Small Gear	Mason/TechSpark	v1.1	1	$1.00	X
Cam Gear Set	Mason/TechSpark	v1.1	1	$12.50	X
DC Wires	GINTOOYUN	B09573KNGW	1	$10.00	
DC Splitter	GINTOOYUN	FQL20210419	1	$8.00	
Wheels	HOLKIE	PBJL2BBS	1	$17.61	
Cutoff Gear Fronts	Mason/TechSpark	v1	1	$16.50	
Cutoff Gear Rears	Mason/TechSpark	v1	1	$16.50	
Reduce Gear	Mason/TechSpark	v1	1	$14.00	
Battery 3850mAh	Tenergy	11742	1	$39.99	
Redo Gears (Small + Cam)	Mason/TechSpark	v2	1	$9.00	
Reduce Gear	Mason/TechSpark	v2	1	$14.00	
1/8 Plywood 1x2 ft	TechSpark		1	$3.00	
1/4 Plywood 1x2 ft	TechSpark		1	$4.00	
Jetson Computing Device	NVIDIA	Xavier NX	1	~$1200.00	
LiDAR Camera	Intel	Realsense L515	1	$589.00	X
Miscellaneous Plywood	TechSpark		1	$0.00	
Wifi Adapter	TP-Link	Nano AC600	2	$17.99	X

TABLE II. BILL OF MATERIALS



TASK	ASSIGNED TO	PROGRESS	START	END
Initial documentation				
Project Proposal	Zach, George, Jack	DONE	1/30/23	2/4/23
Website Setup	Zach, George, Jack	DONE	2/3/23	2/4/23
Pathfinding and database				
Database setup	George	DONE	2/10/23	2/15/23
SLAM/LIDAR bringup	George, Jack	DONE	2/17/23	2/22/23
Data capture testing	George, Zach, Jack	DONE	2/22/23	2/27/23
Simulate pathfinding	George, Zach	REMOVED	2/12/23	2/17/23
SLACK		0%	2/27/23	3/20/23
Movement and local bin detection				
Roomba movement	Jack	DONE	2/13/23	2/17/23
Local bin identification	George, Jack	DONE	2/17/23	2/20/23
Movement towards bin	George, Jack	DONE	2/22/23	2/27/23
SLACK		0%	2/27/23	3/20/23
Bin/robot hardware				
Bin attachment design	Zach	DONE	2/10/23	2/15/23
Roomba fixture design	Zach	DONE	2/15/23	2/22/23
Docking system test	Zach, George, Jack	DONE	2/22/23	2/27/23
SLACK		0%	2/24/23	3/20/23
System integration				
Automatic bin docking	Zach, George, Jack	DONE	3/20/23	3/27/23
Movement with SLAM	Jack, George, Zach	DONE	3/20/23	3/27/23
Bin pickup w/ database	George, Jack, Zach	DONE	3/27/23	4/3/23
Multi-bin pickup	George, Jack, Zach	DONE	4/3/23	4/10/23
Bin replacement	George, Jack, Zach	50%	4/10/23	4/17/23
SLACK			4/17/23	4/28/23

TABLE III. GANTT CHART