Terrestrial Roving Autonomous Scrap Harvester
T.R.A.S.H.

John Chiaramonte
Jack Fenton
Lee Milburn
Catherine Ellington
Jared Raines
Divya Venkatraman

Advised by Waleed Meleis, Ph.D.

April 2022

Contents

Introduction and Related Work

Roadside trash is a massive issue, and one currently solved only by manual labor - a solution that is woefully inadequate. As a duty under the jurisdiction of municipalities and states, it gathers little to no national attention. Very few resources are being put into efforts to automate the process, and all such efforts have been unsuccessful.

To estimate the amount of litter along roadways, a research team selected a random sample of 240 roadway segments, stratified by type and by rural/urban areas.[3] In each segment, a sample area of 300 x 15 feet was identified along the side of the roadway. Observations were then made of littered objects of 4+ inches within the sample site. Separate observations were made within a 15 x 15 foot subarea for littered objects less than 4 inches. The results indicate that there are 51.2 billion pieces of litter on roadways nationwide; and of this, the majority (91%, or 46.6 billion pieces) is less than four inches. Cigarette butts comprise 38 percent of all items littered on the highways, streets, parks and playgrounds. Existing litter also encourages more littering in the area [3].

1.2 million pounds of is trash collected along 10,000 miles of Minnesota state highways each year. Paid municipal crews and low-risk offenders perform community service pick up trash [4]. In the Twin Cities metropolitan area alone, the program costs $2 million a year, even though 75% of the work is done by Adopt-A-Highway volunteer crews.

In 2018, the Ohio Department of Transportation picked up over than 396,000 bags worth of trash from interstates and U.S. routes outside of municipalities. It took state employees, inmates and Adopt A Highway volunteers more than 157,000 hours to collect all the garbage in 2018. The total came to 19,723 full work days, costing Ohio taxpayers $4.1 million [18].

The inefficiency of the current solution - manual labor - is probably best demonstrated by the effect of the recent pandemic on roadside trash.

In Monterey City, California, complaints about trash have increased. Officials say this is not due to increased littering, but rather due to the inability to clean it up. Monterey County Public Works Maintenance Manager Shawn Atkins stated that before COVID, the county relied on people working off community service hours for much of their roadside litter cleanup. Once the pandemic hit, it was not feasible to keep the crews socially distanced the environment properly sanitized, so the program had to be suspended. However, keeping up with litter on Monterey's roadways was a challenge even before COVID. Atkins stated that his cleanup crew was so busy just cleaning up from illegal dumpsites, they did not have time to walk the shoulders of their roads picking up loose trash [9].

Caltrans, which has stewardship for keeping state highways clean, has been faced with the same problem, said Kevin Drabinski, public information officer for Caltrans District 5. Drabinski said it's important to Caltrans that they manage litter, not just because it looks bad, but also because of safety and environmental concerns. Caltrans spends $50 million annually on litter cleanup [9].

Robotic solutions have been attempted, but so far have been unsuccessful. For example, MnDOT (Minnesota Department of Transportation) tried to implement a trash harvester using a design similar to a snowplow machine [5]. It was bulky, required a manual operator, and its brute-force method of collecting trash did not work well. In trials, it often missed pieces altogether, and it was never put into use. Our project uses a vastly different approach to provide a more cost effective and reliable alternative to this problem.

Our compact robot uses a novel three stage approach autonomously map, identify, and pick up trash. One of the advantages of this is extensibility. For example, a drone would likely be the ideal mapping solution if the T.R.A.S.H. System was put into large-scale use. The substitution would require little to no adjustment to the other two stages. Our approach also allows for accurate pickup without the need for high processing power by reducing reliance on heavy software such as YOLO. This makes the final product much more inexpensive, making it more implementable while reducing the downside of real-life use risks such as theft and damage. Similarly, the fact that it is fully autonomous virtually eliminates labor costs from this solution. Far fewer people are required to

maintain a fleet of T.R.A.S.H. Systems than would be required to drive trash-collecting machines, or pick up trash manually. We believe we have successfully established a proof of concept for a novel, scalable, and truly viable solution to the growing worldwide litter problem.

Design Specifications

The T.R.A.S.H. System can clear trash as large and heavy as an average 600mL Spring Valley Water bottle weighing approximately 0.64 kg. The system's robotic base, the TurtleBot 2, has a load limitation of approximately 5 kilograms [15], which presents an upper weight limit on the total load. Since the robot's additional components (external frame, storage container, etc.) are estimated to weigh approximately 3 kg, the trash load must weigh at most 2 kg.

The T.R.A.S.H. System operates in narrow environmental parameters due to budgetary constraints and the limitations of easily accessible technology. The weather must be clear with no rain due to the non-weatherproofed nature of the electronic systems onboard the T.R.A.S.H. System . In addition, due to the wheels that come default with the robotic base (Kobuki Mobile Base), our prototype can only operate on relatively flat, smooth, evenly colored surfaces, with no extreme movement in the background.

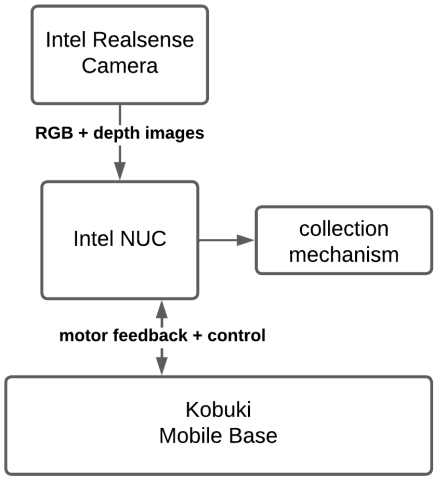

Figure 1: Hardware Overview Diagram

Design Overview

Hardware Overview

The hardware design (Figure 1) has four main components, the depth camera (Intel Realsense D435), the computer (Intel NUC), the mobile robotic base (Kobuki Mobile Base), and a custom designed trash collection mechanism. The camera relays RGB and depth images which are processed in order to identify and target trash. The motor and wheels relay odometric feedback that helps confirm the T.R.A.S.H. System 's current location. The collection mechanism attaches to the front of the Kobuki Mobile base, plugs into power and data ports on the robot, and uses a rotary brush to pick up the trash.

Software Overview

The T.R.A.S.H. System 's software has three distinct stages (Figure 2). In the first stage, the T.R.A.S.H. System maps its environment using a customized version of ORBSlam, an open source Monocular SLAM solution which can accurately create point clouds given RGB and depth camera input. We trained YOLO, an image identification CNN, with our own dataset, then using it to identify and mark the locations of trash clusters on the map. We then render the 3D point cloud down to a 2D occupancy grid using a custom implementation of Octomap. It is from this 2D occupancy grid that the robot can navigate around an environment.

Once the mapping stage is over, the T.R.A.S.H. System enters its second stage: general navigation.The Turtlebot uses adaptive Monte Carlo localization (AMCL) to locate itself in the built map. That way it can identify obstacles and path plan around them to the trash cluster points labeled in that map.

Upon reaching a cluster, the third stage begins. YOLO identifies trash in the RGB image from the Realsense Camera, and from the coordinates of the trash detection and the distance measurement received from the depth camera, calculates the position of the trash relative to the robot. Once it identifies the relative position of the trash, the robot turns towards its target, starts the collection mechanism motor, and moves towards it, picking up the trash piece. Upon successful collection, it returns to the general navigation stage, repeating on until all trash clusters have been visited.

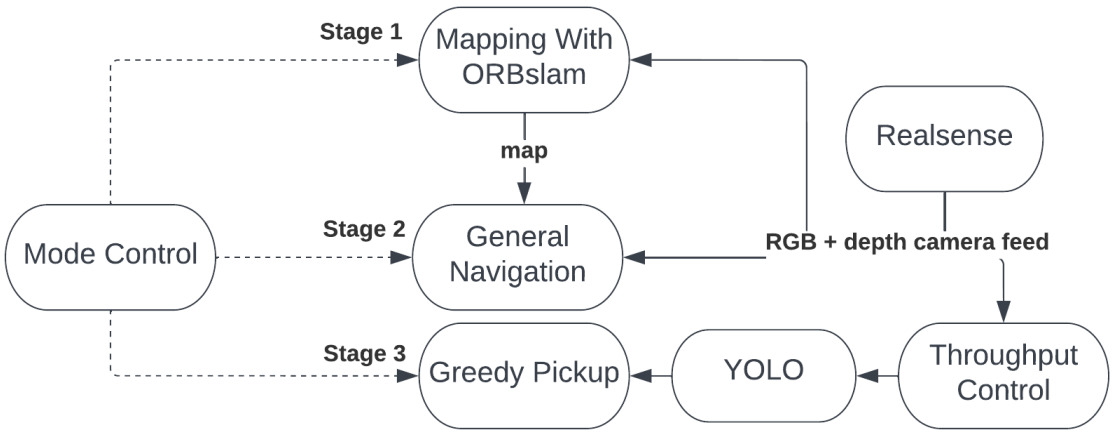

Figure 2: Software Overview Diagram

Figure 3: TurtleBot 2 Base System

Design Details

Hardware

Robot Base

A TurtleBot2 serves as the robotic base for our system. The TurtleBot is a low-cost personal robot kit with compatible open source software. The kit includes a Kobuki Mobile base for movement, a ROS compatible Intel NUC to run the operating system, and either a Kinect camera or Asus Xtion Pro Live 3D sensor, which we have replaced with our own sensor for this project. The robot software includes an SDK, IDE, demo applications, and libraries for visualization, control, and planning. The robot includes wheels with precise odometry, cliff (5cm drop) detection, motor overload detection (current over 3A), gyroscopic sensors, bumpers, wheel drop sensors, three power connectors, five expansion pins, speakers, two LEDs, one state LED, three touch buttons, a 2200 mAh lithium-ion battery, and a dock charging station [8].

The TurtleBot motor is a standard brushed 12 V DC motor in the Kobuki mobile base and allows the robot to move at a max speed of 65 cm/s and a rotational speed of 180 degrees per second. The wheels allow the robot to maneuver over obstacles up to 12 mm high. The wheels have built-in encoders which allows for extremely accurate odometry, up to 11.7 ticks/mm or 25718.16 ticks/revolution [12]

The base system is pictured in (Figure 3).

Computer

The T.R.A.S.H. System 's computer is an Intel NUC NUC5i7RYH, a single-board x86 computer with an Intel i7 5557U and 8GB of DDR3 memory. It has built-in memory, USB ports, and Wi-Fi capabilities, all of which are used to receive data and control peripheral devices. This computer was chosen since it was available from the same laboratory from which the Turtlebot was obtained as part of the robotic system. This computer was wiped and reformatted with Ubuntu 16.04 since this version of the operating system supports both ROS and associated software libraries necessary to operate the Kobuki within the Turtlebot Ecosystem. The computer itself was released in 2015 and does not possess a GPU, so it ended up being largely underpowered for the intensive Image Identification we performed. A future iteration of this project would almost certainly use a similarly-priced GPU-enabled development board from Nvidia such as the Jetson Nano. Research and first-hand observation of this development board running the same image identification algorithm demonstrated a greater than 20x performance increase for what would be a negligible cost increase in an actual production design.

Collection Mechanism

The collection mechanism is a custom-designed addition to the Turtlebot Robot. Its mechanical construction consists of 20-20 aluminum bars connected by 90-degree brackets on corners using nuts and bolts. In order to allow free range of motion, caster wheels were affixed to the bottom of the collection mechanism frame. The frame was designed such that it could slide into the front of the robot and attach via flexible mounts, which ended up being plastic strips which are looped around the metal poles of the Turtlebot and allow for complete removal of the collection mechanism from the Kobuki or even adaptation to another robotic base entirely. (Figure 4) is an image of the initial, hand-drawn design of the collection mechanism frame, which remains true to the final construction (Figure 5) in form and dimensions, minus the placement of some additional support beams and attachment of the motor, bin, and brush.

When the collection mechanism motor is activated, it sweeps trash up a ramp into a plastic storage container, which was designed as a similar size to the Turtlebot (14 x 14 x 16.5 inches) so as not to significantly impact the Turtlebot's center of gravity. A funnel was added to the front of the collection mechanism, seen on the right side of the picture below, in order to rein in trash that may have been missed slightly. This funnel made it easier for the system to pick up larger items like water bottles as well as smaller items like paper balls when either the robot turns to a slightly incorrect angle or the caster wheels lock or turn erratically, introducing error in the overall turning of the robot.

The design of the electronics system for the collection mechanism largely revolved around the question of how to control a large high-power motor from a computer, and due to the previous experience our team possessed, a microcontroller-based solution was introduced. An Arduino Mega is connected to the NUC using a USB cable, and that Arduino is then connected to an L982N Motor driver breakout board over PWM-enabled GPIO pins (5V). This motor driver breakout board is supplied with 12V by the Kobuki base itself on a 12V, 1.5A Max port. The output of the L982N is the DC motor itself which drives the chain and eventually the brush on the front of the robot, pictured in the figure above. The circuitry is depicted in (Figure 6), where the Kobuki's power source is represented as a DC power source and the DC motor is represented as a labeled square.

The weight threshold for the Turtlebot is 5 kg. The peripheral components and electronics take up 3 kg of weight, so the storage bin could theoretically hold up to 2kg of additional weight; however, the specifications of the Kobuki were likely erring on the safe side, so more weight is likely possible, but was not tested thoroughly.

The brush itself was hand-designed and fabricated due to the fact that multiple industrial brush manufacturers refused to make a custom model and no available solutions fit our design specifications. We used a 1-inch diameter aluminum pipe as the core of the brush, and used garage door brush

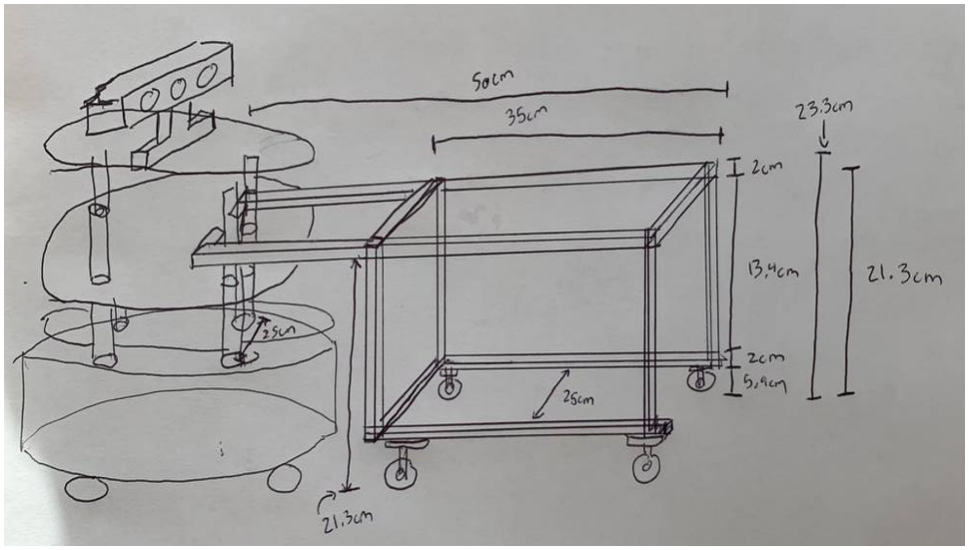

Figure 4: Collection Mechanism Design

sweeps as the bristles. The garage door brush sweeps were cut into segments and, In order to affix the bristle segments to the pipe, a metal-infused epoxy, JB-Weld, was used. Four bristle segments were attached tangentially to the metal pipe in order to create a rotary brush. This brush rotates on pillow block bearings and is powered by a chain-driven sprocket. The brush is pictured in (Figure 7).

Intel Realsense Camera

The Intel Realsense D435 is an RGB-D camera that transmits both RGB and depth information. Each pixel in the depth image is a 16 bit depth value in millimeters. Its maximum range is 10 meters, with an ideal range of 0.3 meters to 3 meters. Its depth field of view (FOV) is 87° horizontally and 58° vertically, while its RGB field of view is 69° horizontally and 42° vertically. It has a depth frame rate of up to 90 fps, and a RGB frame rate of up to 30 fps. This camera was mounted to the front of the collection mechanism's frame to allow for maximum visibility of trash as well as the environment for mapping purposes. A 3D printed mount for the Realsense D435 was sourced from Thingiverse, an open-source platform for the sharing of 3D models[19]. The camera was mounted using M3 bolts and was connected to the NUC using a high-speed USB 3.1 Gen 2 Type C cable.

Software

Mode Control

In order for the T.R.A.S.H. System to understand what to do, it needs a central software controller to switch between its different modes. A "Mode Controller" was created to switch between the three separate software components of the system: Mapping, Navigation, and Greedy Pickup. This Mode Controller is a ROS node which communicates with the Mapping, Navigation, and Greedy Pickup nodes, shutting them on or off as needed. The Mode Controller starts off as idle before putting the T.R.A.S.H. System into Mapping mode. Once mapping is deemed completed (a map has been generated and trash has been identified within that map), the mode controller then turns off Mapping mode. The map is then passed on to the Navigation mode along with the target coordinates with trash. The Mode Controller then turns on Navigation mode. Once a target coordinate has

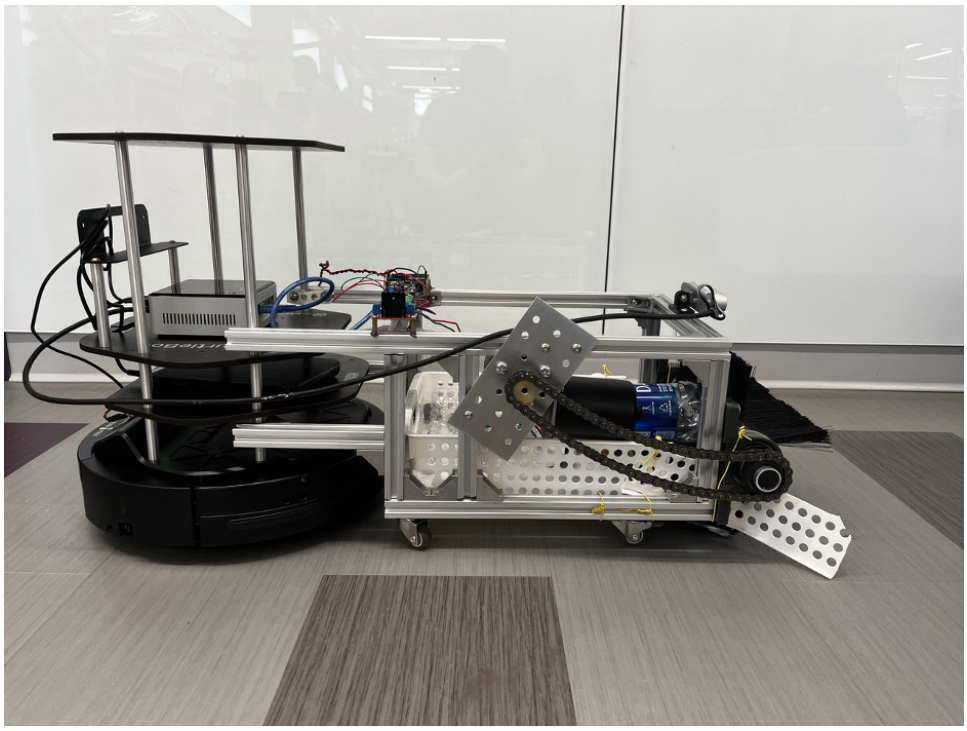

Figure 5: Final Construction

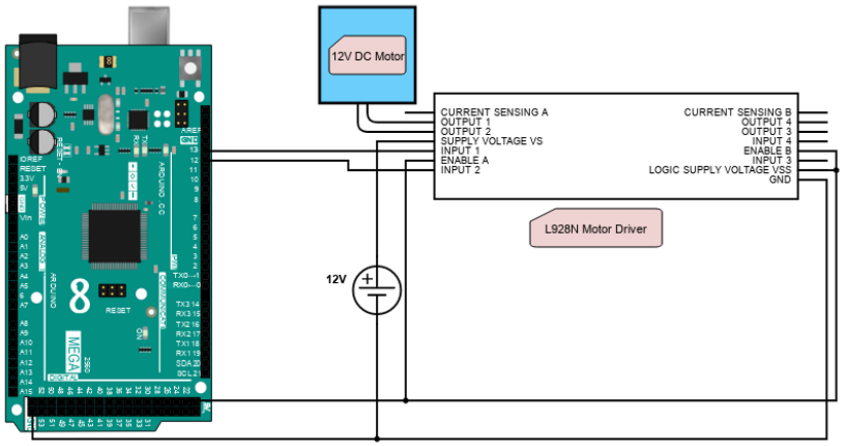

Figure 6: Motor Circuitry

Figure 7: Brush

been reached by the Navigation mode, the Mode Controller next turns Navigation off and Greedy Pickup on, picking up the trash. Navigation mode is once again activated and the robot navigates to the next marked trash location before the Mode Controller turns on greedy pickup once more. Navigation and Greedy Pickup will alternate until all trash is removed from the environment. Once all of the trash is picked up and no marked coordinates remain on the map, the Mode Controller turns back to idle and awaits further instruction. This relationship is pictured in (Figure 8).

Mapping

The T.R.A.S.H. System hinges on the ability to navigate the surrounding area and map its environment as it moves. There are many SLAM (Simultaneous Localization and Mapping) solutions that have been developed in the last 20 years that have been made open source, but many of these require heavy resource demands. Most use either expensive sensors such as LIDAR, high computational demand, or data rich maps with extensive memory allocation, so finding a light-weight real-time SLAM solution is rare.

One project that has proved to fit this description and gained in popularity over recent years is the visual (camera based) SLAM solution ORB-SLAM2 [13] [11]. "ORB-SLAM2 [is] a complete SLAM system for monocular, stereo and RGB-D cameras, including map reuse, loop closing and relocalization capabilities. The system works in real-time on standard CPUs in a wide variety of environments from small hand-held indoors sequences, to drones flying in industrial environments and cars driving around a city". The original ORB-SLAM, ORB-SLAM2, and ORB-SLAM3 all build upon the same structure: environmental features represented as a bag of words of ORB binary features invariant to rotation and scale; the same features for tracking, mapping, relocalization and loop closing; real time loop closing based on the optimization of a pose graph; and a keyframe-based

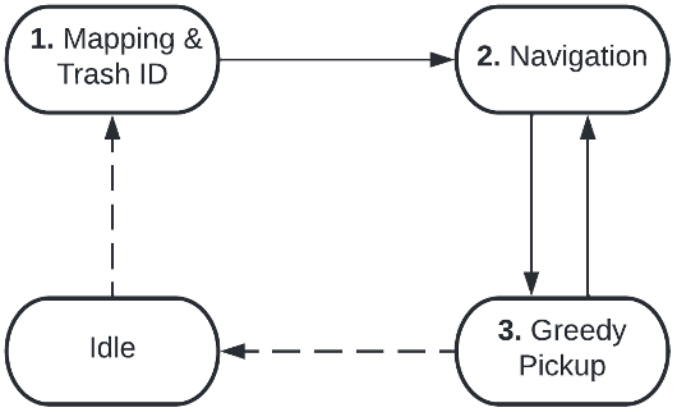

Figure 8: Mode Controller Diagram

SLAM approach that reduces the overall size of the SLAM map [10]" (Figure 9).

Many different iterations have been made to the original ORB-SLAM2, and its successor ORB-SLAM3, but our system constraints proved a challenge for deploying most of them [14] [2] [17] [13] [7] [1]. Both our operating system (Ubuntu 16.04) and ROS distribution (Kinetic) were already end-of-life at the time of this project so all dependencies have dropped support and are very difficult to revert to previous versions.

Picking which iteration of ORB-SLAM to build upon came down to what functionalities were already provided in each. Our approach to path-planning required the full map to be represented in the form of a 2D occupancy grid, so we selected an iteration [14] that iteratively builds a 3D occupancy grid by transforming each map update into a point cloud and adding it to the octomap (and then projecting it down into 2D space (Figure 10). This repository is modified to allow for much easier customization with two .yaml files containing all the configuration properties. It should be noted that the sparse nature of an ORB-SLAM map is defined explicitly here, with only 1000 orb-features defining each frame and an estimated republish rate of once per second.

Like the rest of our project, this repository is designed to be used within ROS as a ROS node. In our default RGB-D configuration, the node subscribes to 2 topics (RGB and depth image topics) and in turn publishes all necessary data built by the ORB-SLAM2 system. This includes a point cloud of all map key-points, the current camera pose, the full camera path trajectory, and a morphologically transformed version of the projected occupancy grid (Figure 11). Morphological operations are commonly used tools in image processing to clean up an image [16]. We found in experimentation that our maps were initially filled with noise that led to an inability to navigate the space. By "eroding" and "opening" the space we were able to remove errant data points that were being misidentified as occupied and then by "closing" we were able to close gaps caused by the sparse data points in our map. The end product was an occupancy grid very close to real-world surroundings with a very light-weight mapping solution.

Trash Identification

Simultaneously as an area is being mapped, the T.R.A.S.H. system also detects trash in that map. In order to recognize where in the map a piece of trash is, it first needs to find the location of a piece of trash relative to the robot. A system to locate trash was devised using an advanced open-source Image Identification software called YOLOv4, the position and rotation data of the Kobuki Base,

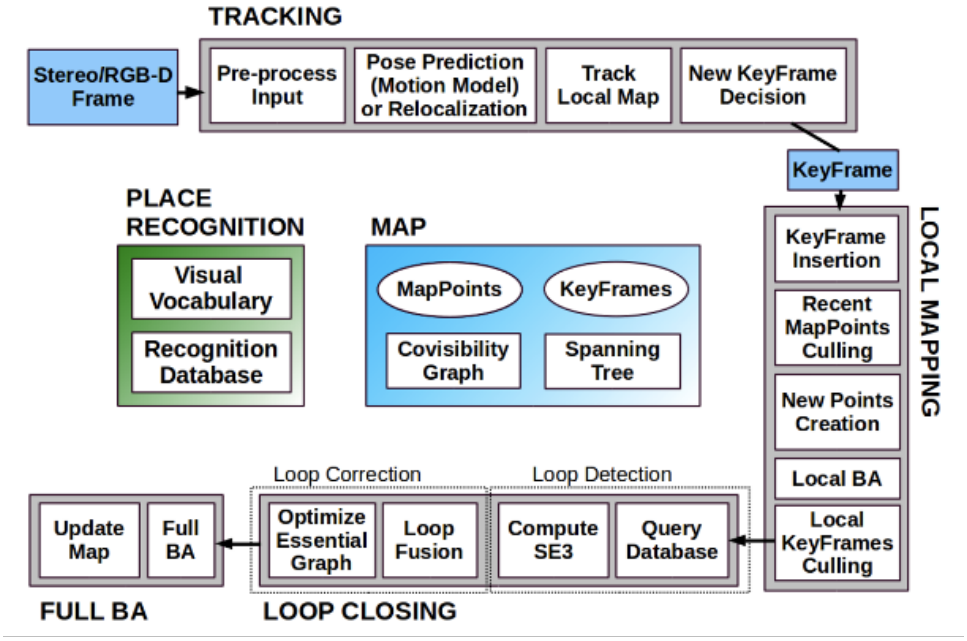

Figure 9: ORBSLAM System Threads and Modules

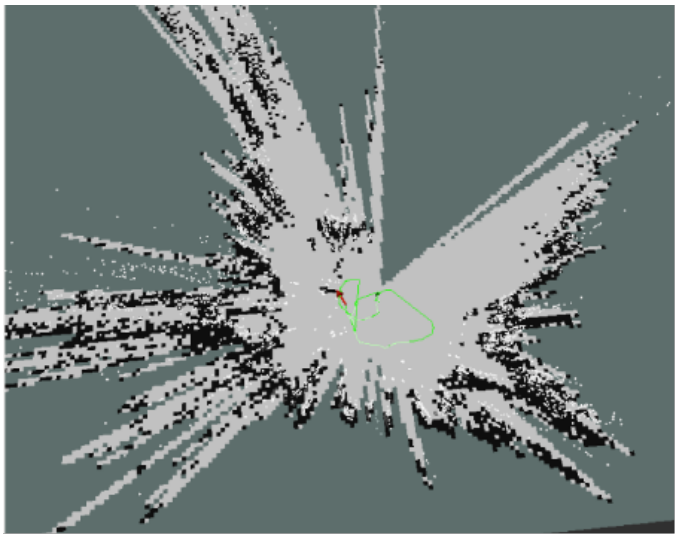

Figure 10: Raw Occupancy Grid

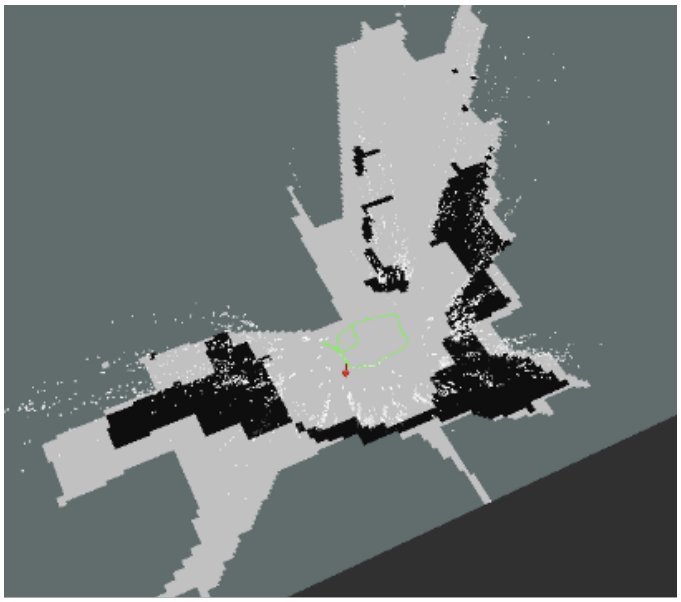

Figure 11: Morphologically Transformed Occupancy Grid

and the depth camera feed provided by the Intel Realsense D435 RGB-D camera.

The first step in the trash identification pipeline is image identification using YOLOv4. YOLOv4 is a convolutional neural network which we trained with a custom dataset of over 1000 images, each taken of varying pieces of trash from the perspective of the robot. Trash pieces used in the training images included water bottles, plastic cups, paper balls, aluminum cans, and others. Each of these over 1000 images was hand-labeled by our team and fed into our machine learning model using a 80-15-5 split between training, validation, and testing images, respectively. Once the model was trained and running in our software stack using a customized open-source ROS wrapper for YOLOv4 [20], the model can detect trash extremely confidently such that we set the threshold for a successful detection to over 90% in the rest of our software stack. The image identification model runs constantly while the environment is being mapped using the RGB camera feed and returns "bounding boxes" around identified trash pieces in the image (Figure 12).

These bounding boxes provide x and y pixel locations for the trash in the image, but do not contain any information about where the trash lies in the environment, so more information is needed before the trash identification can be placed in the map.

The next step in this pipeline is to identify the angle of the closest piece of trash relative to the camera. This is accomplished by using the center pixel X coordinate of an identified piece of trash. Using the field of view of the camera, an imaginary triangle can be created to discover the angle of the trash relative to the camera in the real world by using pixels as the coordinate system (Figure 13).

Since the FOV angle is known as 69.4 degrees, its opposite side is known as 640 pixels, and it is known to be an isosceles triangles, the remaining side lengths and angles can be extrapolated as this is considered a trigonometrically "solved" triangle. Using this triangle we can calculate the angle of the identified trash piece using the inverse tangent function, as shown in the following equation and in (Figure 14).

$$\theta = \tan^{-1} \frac{trash_x - 320}{462.139}$$

The next step in the localization process is to figure out the distance between the camera and

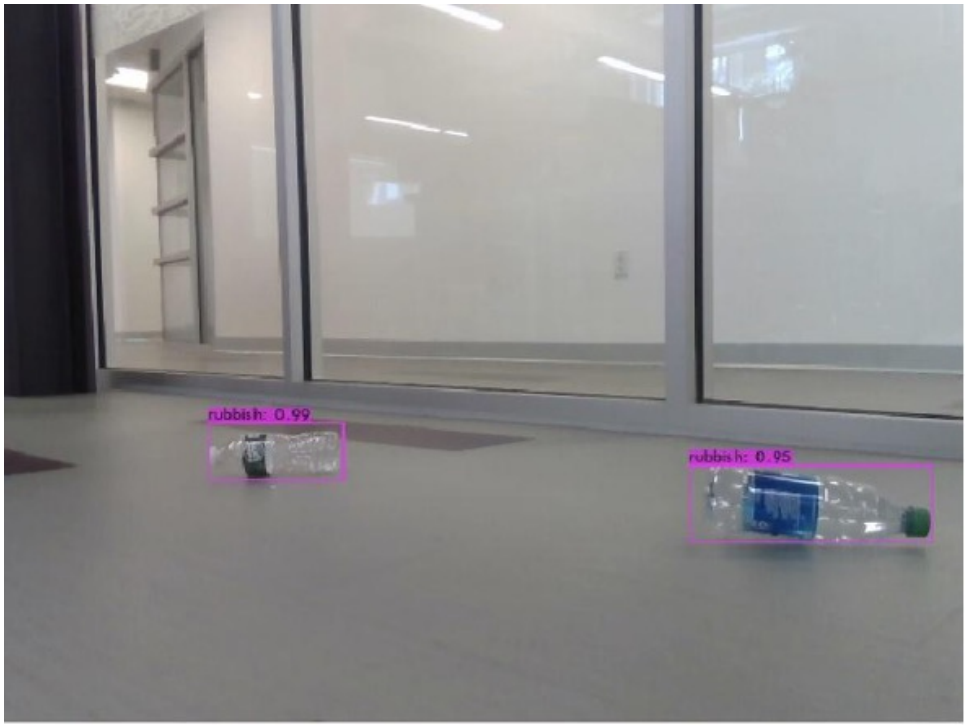

Figure 12: Bounding Boxes

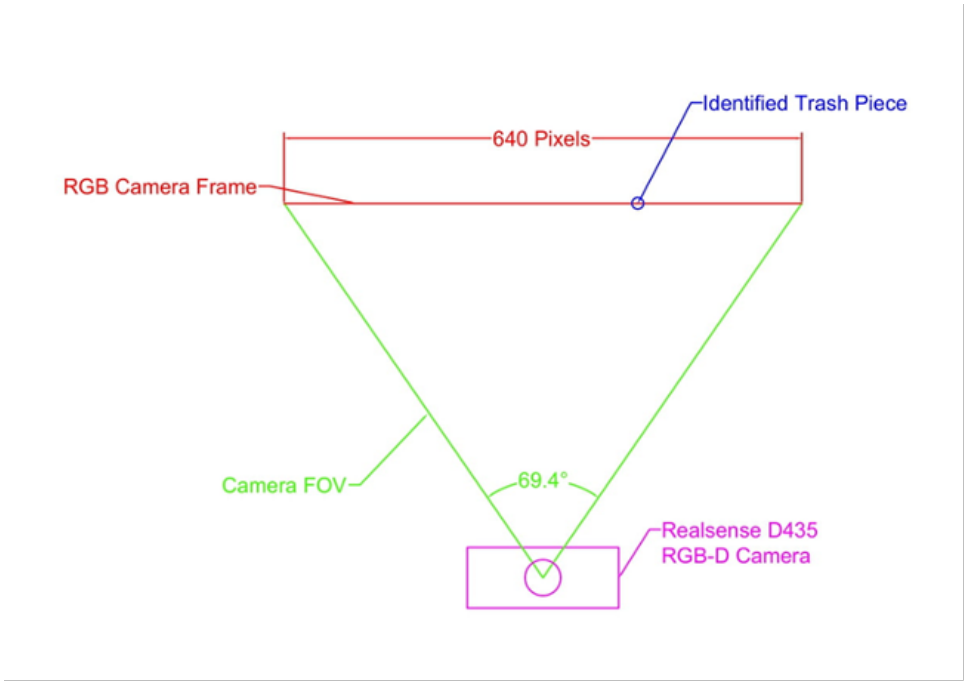

Figure 13: Field of View Diagram

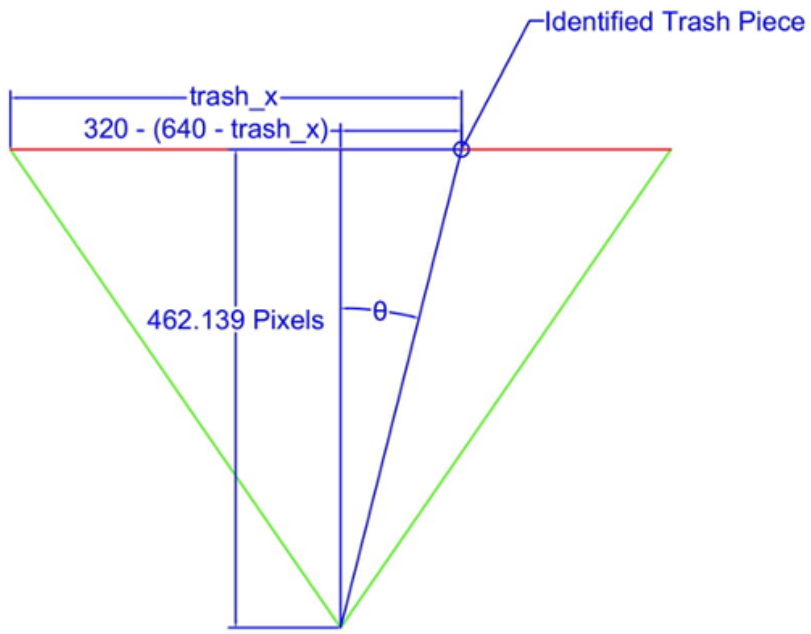

Figure 14: FOV Trignometric Calculations

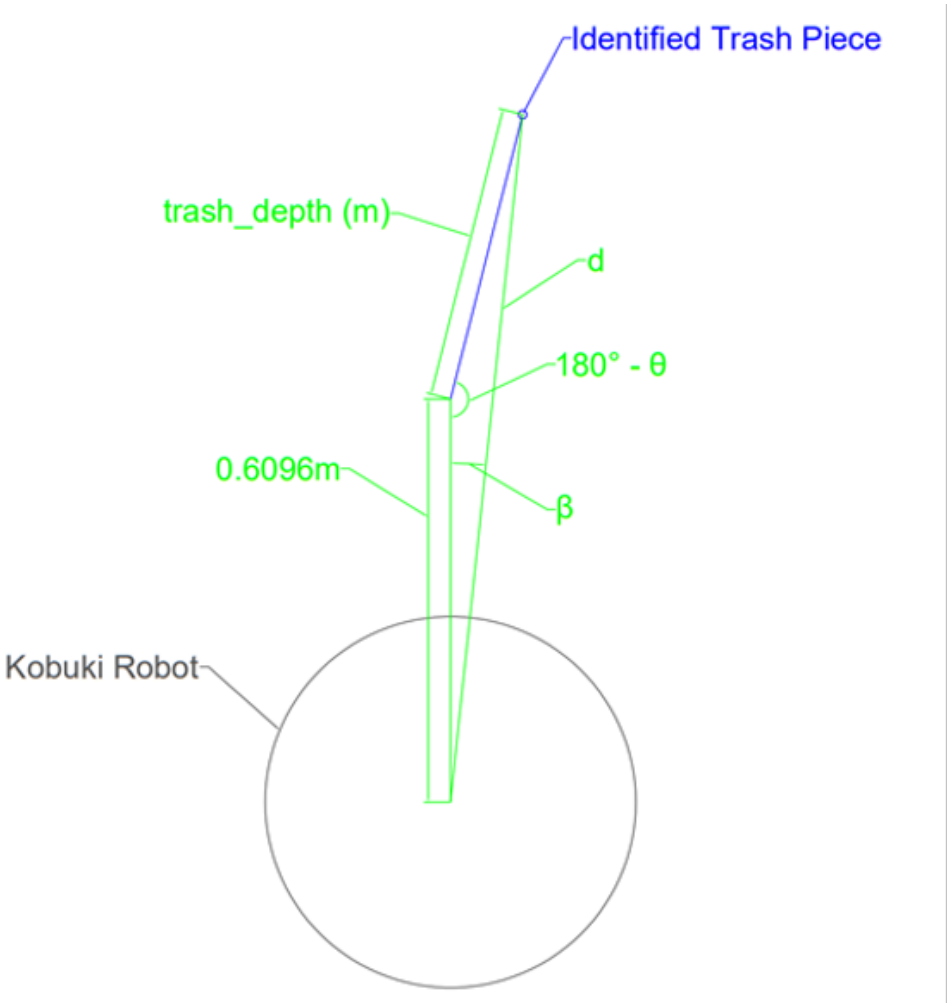

Figure 15: Robot-Trash Trignometric Calculations

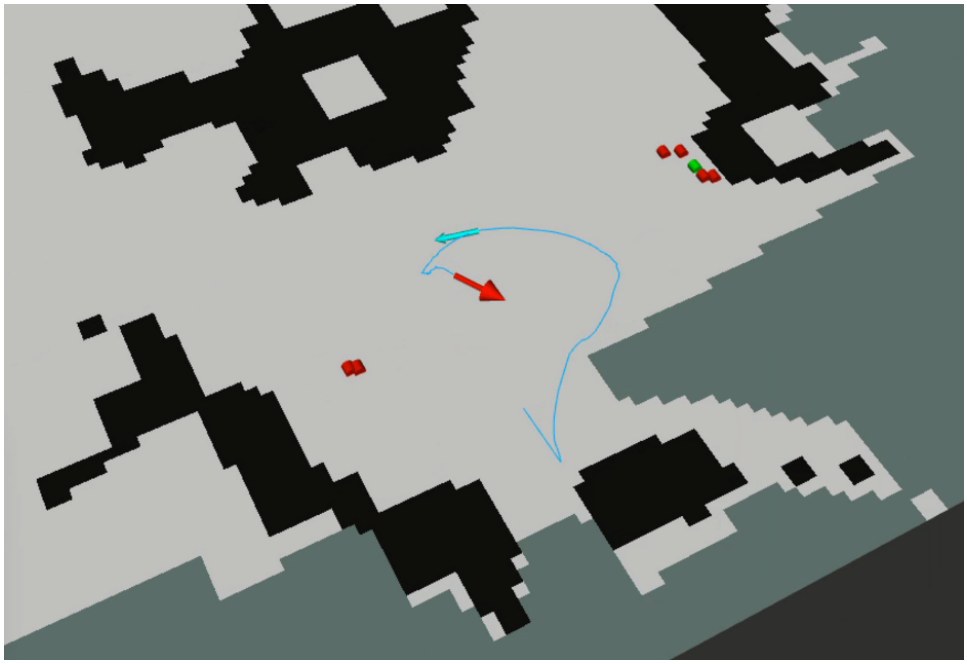

Figure 16: Robot Trajectory in Map

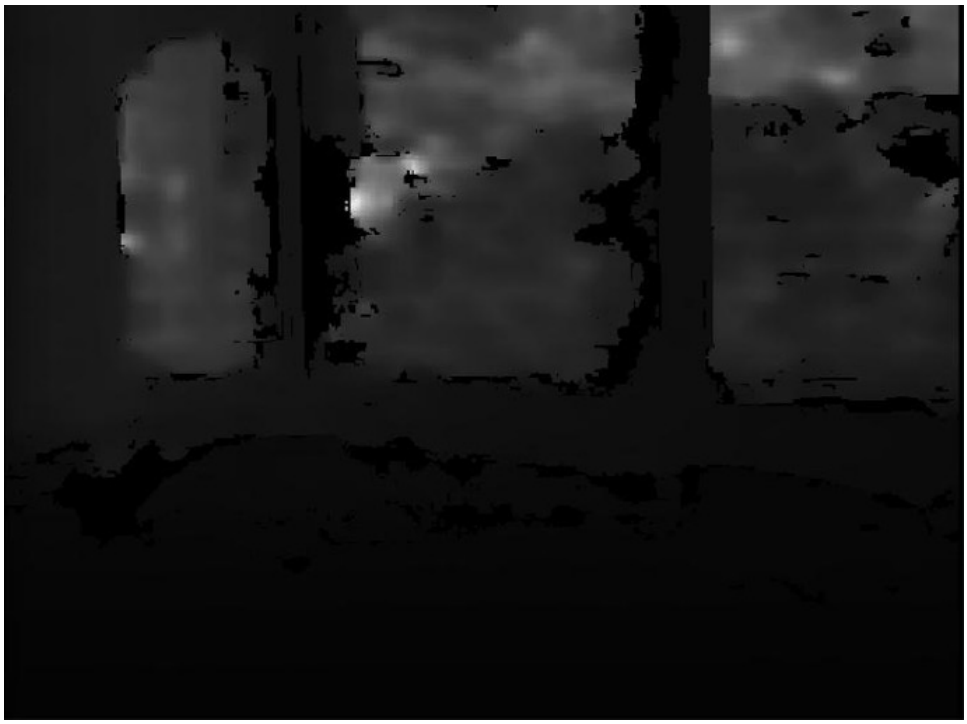

Figure 17: RealSense Depth Feed

the piece of trash. This is accomplished using the depth camera feed provided by the Intel Realsense D435 RGB-D camera. This camera outputs a grayscale image in which each pixel is a 16-bit value representing the distance to that pixel in millimeters directly from the center of the camera (Figure 17). The depth picture is difficult to visually understand from a human perspective, but it can be easily indexed as a matrix using the x and y coordinates given by YOLO's bounding box to determine the exact distance between the camera and any piece of trash.

Once the distance between the camera and the trash has been calculated, all information necessary to localize the piece of trash relative to the robot have been acquired. Using a second triangle with coordinates in meters, we can extrapolate both the angle of the trash relative to the robot as well as the distance between the trash and the robot, both of which are necessary to localize the piece of trash (Figure 15).

The first unknown which must be solved is the distance between the trash and the center of the Kobuki robot, d. Since the distance between the camera and the center of the robot is known as 0.6096m and the distance between the trash and the camera was taken from the depth camera feed ($depth$), d can be solved using the Law of Cosines as shown below.

$$
\begin{aligned}
c^2 &= a^2 + b^2 + 2ab\cos(c) \\
c &= \sqrt{a^2 + b^2 + 2ab\cos(c)} \\
d &= \sqrt{(depth^2 + 0.6096^2 + 2(depth)(0.6096)(\cos(180^\circ - \theta))}
\end{aligned}
$$

Once d is known, the final variable which needs solving is β. This can simply be solved using the Law of Sines.

$$
\begin{aligned}
\frac{\sin X}{x} &= \frac{\sin Y}{y} \\
\frac{\sin(\beta)}{depth} &= \frac{\sin(180^\circ - \theta)}{d} \\
\beta &= \sin^{-1}\left(\frac{depth\sin(180^\circ - \theta)}{d}\right)
\end{aligned}
$$

Once the angle to the piece of trash relative to the robot and the distance between these two points is known, theoretically all that needs to happen is to add these values to the robot's current position in order to realize the piece of trash on the map; however, some difficulties arose in timing. YOLOv4, when run on our old, low-power computer, processes images at a throughput of about 0.5-0.8 FPS with about 4-5 seconds of latency from when the image was originally taken. This creates a large gap between the time when the image was taken and the current position of the robot. In order to combat this, we had to log the entire path of the robot as it was mapping with exact timestamps for every single position in this path from ORB-SLAM2. Once we receive a successful trash detection, the timestamp given from YOLOv4 from when that image was taken is passed to the path and a position and angle, or `Pose` in ROS terminology, is output. It is from this `Pose` that we add the distance d and angle β in order to localize the piece of trash relative to the map itself. This process is depicted in (Figure 16) as the robot simultaneously maps an area and identifies trash in the map.

The thin blue line is the path of the robot as it maps the area. The red arrow is the current position of the robot in the map. The cyan arrow is the `Pose` where the robot was when the YOLOv4 image was taken. From this cyan `Pose`, a red trash detection is then finally placed on the

map. Every trash detection is plotted, and a separate anti-clustering node averages these together, which is discussed further in the following section.

Anti-Clustering

There was an issue with the original trash detections. Images of the same piece of trash would be processed more than once. This lead to a lot of noise for the robot to sort through, causing up to twenty or thirty detections for two pieces of trash. There would also be erroneous detections where our YOLO v.4 model would classify something as trash that we didn't want to be classified as trash. We had to sort through the duplicate detections and the noisy random detections. To do this we ran each trash detection through a filter. Everytime we would publish a new trash detection we had a subscriber that would listen to the trash detections and determine if it was a new trash detection or a detection of a piece of trash already detected.

To do this we store all the detected pieces of trash and if the new piece of trash is within a given radius we take that as that piece is part of the trash cluster we're filtering to and we take a rolling average of the detections. That means we count how many times a piece has been detected and use that to weight the amount of influence the new detection will have on moving the trash cluster's position. This dramatically decreased the amount of detections which we would autonomously pass to the navigation. The other issue was the erroneous trash detections. To avoid this we filtered out the clusters which did not have enough detections found, we figured that all these noisy detections were either or two detections so if we only published detections that have 3 or more points averaged together we could ensure that we're looking at real trash points.

General Navigation + Path Planning

General navigation consists of two parts: localization and path planning.

We use Adaptive Monte Carlo Localization (AMCL), which uses odometry feedback from the motored wheels and scan data from the Realsense Camera. It returns a probability distribution of its current location in the map, using sensor data to confirm or deny the objects around it. AMCL loads a 2d occupancy grid that has the trash points identified on it. The points that the robot then navigates to points that are within two meters away from the detected trash point because greedy pickup is routinely effective within that distance.

After the destination points are set in the map, the robot path plans using the Dynamic Window Approach (DWA) which sends velocity vectors to circular bases keeping it within distance of a generated path through the 2D Octomap. Parameters can be passed to the DWA planner which specifies how tightly or loosely the robot must follow the generated path.

The robot will try to reach locations and if its probability gets too low it can enter recovery behavior. Recovery behavior is done by the robot spinning around to confirm its location against the scan data it receives from the Realsense. After it localizes itself it can continue trying to generate a path to the next point that's passed to it.

Greedy Pickup

Once the Navigation portion of the software stack places the robot near a marked trash location, we needed another system to actually pick up the trash. Since the trash mapping and navigation are both not 100% inaccurate, a third stage in the trash pipeline was necessary to correct for this error. A system called Greedy Pickup was created to ignore all navigation and map factors and solely focus on seeking out the trash directly nearby.

Broadly speaking, greedy pickup follows five main steps:

- Using YOLO, we calculate the angle of the trash relative to the camera
- Using the depth camera's data, we calculate the distance from the robot to the trash

- The Kobuki turns towards the piece of trash, facing directly towards its target.
- The motor is activated, turning on the collection mechanism.
- The T.R.A.S.H. System sets off at optimal speed to pick up trash.

Firstly, when greedy pickup is activated, it is fed a direction to start scanning, and from this direction begins to turn and look for trash by using YOLOv4. Once it receives a trash detection from YOLOv4, it calculates the position of the trash using the same algorithm explained in the Trash Identification section, but uses the local Kobuki odometry data rather than any navigation or pathing position as the base position to localize the piece of trash. After it knows where the trash is, it turns back towards it at the precise angle, turns on the collection mechanism's motor, and moves exactly 0.2m past the trash's location. The robot overshoots by 0.2m to ensure that the collection mechanism has enough time to rotate fully and push the trash into the collection mechanism's plastic bin. Once this occurs, the motor is once again turned off, and, if the entire system is running, navigation resumes.

In order for the collection mechanism to turn on or off, the NUC must send a serial packet to the connected arduino with only three bytes in sequence, either [0x59, 0x59, 0x59] to start the motor or [0x4E, 0x4E, 0x4E] to stop the motor. Once the Arduino receives a start packet, it then outputs a PWM signal to two of its GPIO pins which control the L928N motor controller. The PWM signal gradually increases from a low duty cycle to a higher duty cycle to control the current spikes on the 12V line from the Kobuki to the Motor. In the initial design on bench power, starting the motor from 0 to full power produced an initial current spike of approximately 1.9A before settling around 0.9-1.1A when in normal motion or picking up an object. The initial spike was over the 1.5A limit provided by the 12V port accessible on the Kobuki robot. In order to eliminate the current spike, a slow ramp-up of the duty cycle of the PWM signal was introduced from 20% duty cycle to a maximum of 80% linearly over the course of 5 seconds. This removes the initial current spike and ensures that the motor can both properly power the collection mechanism and does not exceed the 1.5A current limit.

Using greedy pickup allows for accurate pickup without the need for high processing power by reducing reliance on heavy software such as YOLO. This makes the final product much more inexpensive, reducing the downsides of real-life use risks such as theft and damage. It also makes our project adaptable to uncontrolled testing environments, where trash is likely to shift at least slightly after initial mapping.

Transforms

In order for the robot to interpret the sensor data correctly, all of its physical components have to know where they are in respect to each other. We wrote transforms that reconcile the distance between each component in the robot. For example, our transforms coordinate the perspective of the Realsense Camera (set on the edge of the collection mechanism) with the odometry data (centered at Kobuki Base.) This results in a transform tree amongst all the robot's components that allow for it to accurately complete each step in the trash mapping and picking up process.

Code

Our entire software stack is publicly visible on our GitHub Organization's page, spread over many repositories [6].

Design Evaluation

Testing was conducted both in a simulated environment and in a controlled, real-life testing environment, consisting of an enclosed space with randomly scattered pieces of trash. The robot was evaluated on its ability to accurately map the enclosure, identify and mark the pieces of trash, choose an efficient path, and pick the trash up.

Simulated testing (Figure 18) was done in the Gazebo Robotics simulator. This simulator was included in the base Turtlebot SDK and includes near true-to-life recreation of the entire turtlebot system. The use of ROS allows for all code in the navigation stack and greedy pickup to be run against the simulator and behave exactly identically to reality. This simulator was instrumental in the initial testing of movement and navigation as it allowed our team to test many different speed parameters and movement algorithms without risking any physical damage to the robot. This simulator is depicted in the figure below. Due to the recording capabilities of ROS, we were also able to record all data associated with movement and mapping and play them back in a simulated environment to recreate and reevaluate our physical testing. This allowed for useful visualizations and assessments of what the robot was thinking at any given time.

In order to evaluate the performance of the entire system once it was completed, trash was scattered around the capstone lab and the system was engaged. The Mode Controller was activated and the robot proceeded to map the environment, navigate to trash points, and attempt to pick up the trash. In these tests there were several degrees of success and failure. In nearly every attempt, mapping was done extremely successfully and trash points were in the relative positions in the map as in the real world, minus some error introduced by the clustering problem. The two main sources of error were navigation and greedy pickup. Even though we used an advanced navigation algorithm, it did not properly account for the physical changes we made to our robot, so the robot experienced some inconsistent difficulties in turning and moving to the target location. The inconsistent performance of the caster wheels introduced some additional error that the navigation could not deal with. Fixing these would have required us to rewrite large portions of the open-source navigation stack to account for these edge-cases, which was outside of the scope of our project. In addition, greedy pickup also experienced similar inconsistencies, only successfully picking up trash in 70% of cases over 30 trials of varying trash within 2 meters. The compounding of these error factors meant that the entire system only functioned perfectly roughly 50% of the time over several tests, with different error sources inhibiting the pickup of the trash each time. Since these errors were largely due to the physical design of the collection mechanism, and the software was largely functional, a future iteration of this design would most definitely include a much more robust collection mechanism and attention to detail regarding the mechanics of the robot. However, since the Turtlebot was the only robot available and we had limited time and resources to complete complex mechanical, electrical, and software designs, we allocated more time to the software side and made that portion of the project as robust as possible. Given these factors and the relative success of our robot, we believe that the T.R.A.S.H. system demonstrated a successful proof of concept for an automated trash collecting robotic system.

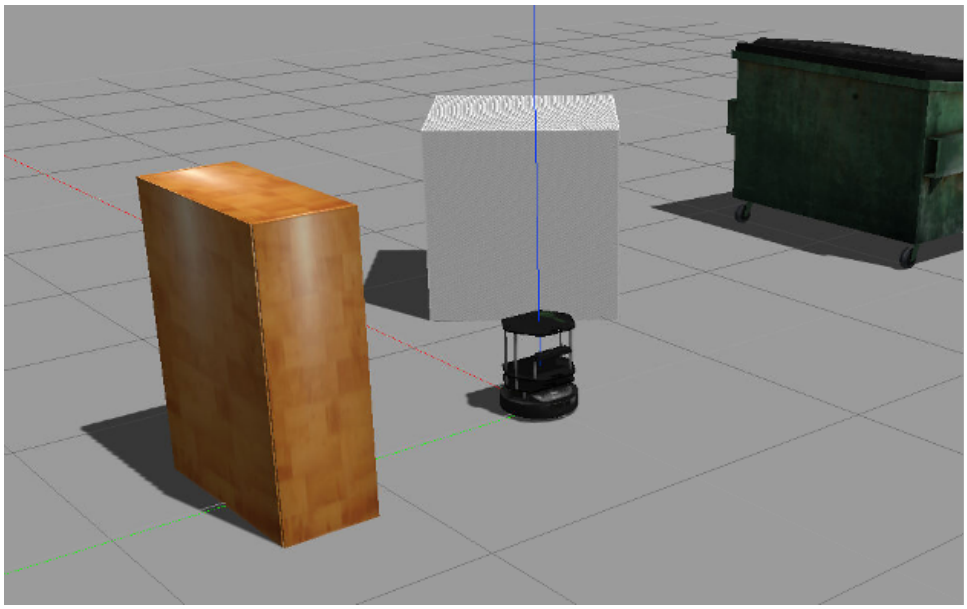

Figure 18: Simulation Testing

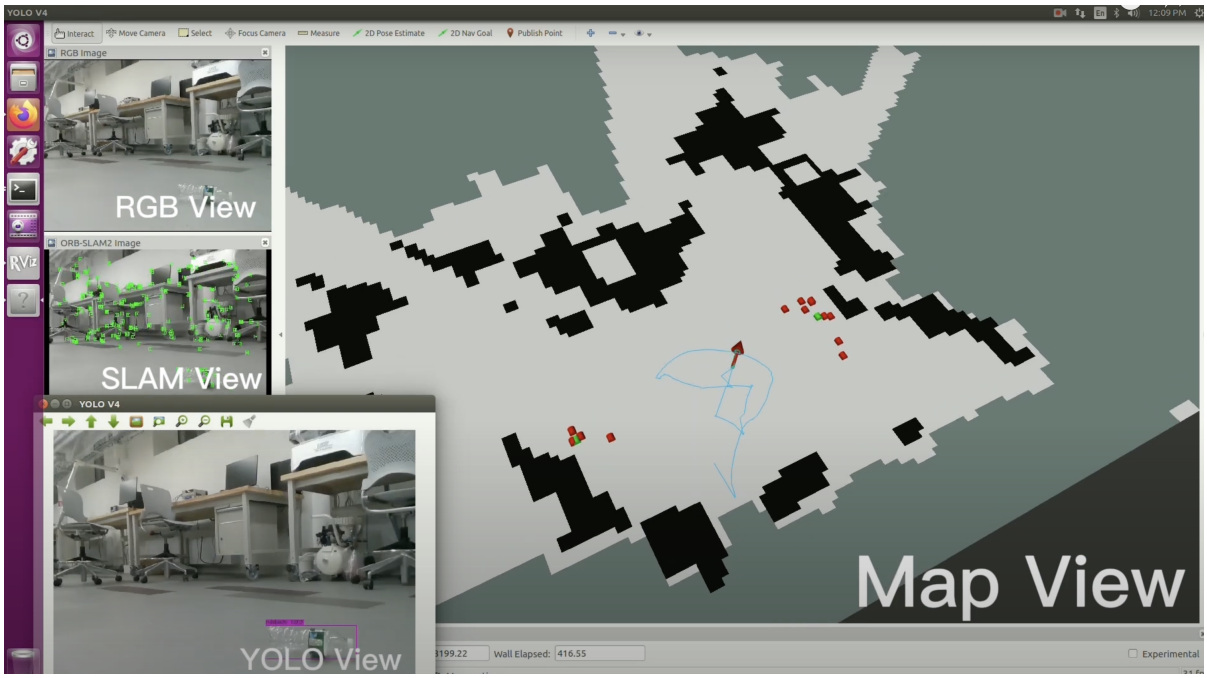

Figure 19: Real Time Data Processing and Map Creation

Safety

Since our project is fully automated, the only human interaction is the emptying of the trash receptacle when the robot is idle. There are no sharp edges or moving parts that could pose a risk. No data is collected or used from the user, so privacy is not an issue. The use of this trash removal process makes collecting trash in dangerous environments like side of roads safer due to not having manual cleaners and possible highway collision accidents. Our robot is not equipped to recognize or process anything from its RGB-D feed except trash, and the feed is not stored beyond use for other purposes.

Parts

Prototype Cost

Part	Quantity	Cost/unit	Source	Total
Intel Realsense D435	1	0	River Lab	0
Intel i7 5557U NUC 8GB RAM	1	0	River Lab	0
Kobuki Mobile Base	1	0	River Lab	0
500 mm T Slot 2020 Aluminmum Extrusion	9	2.49	Amazon	22.41
Cast Iron Pillow Block Mounted Bearings	2	6.22	Amazon	12.44
#35 ANSI No., 3/8" Pitch, 15 Teeth, 1" Bore Sprocket	2	9.44	Amazon	18.88
PZRT 2020 Aluminum Profile Connector Set	2	26.99	Amazon	53.98
L298N Motor Driver Controller Board	1	8.99	Amazon	8.99
35 Roller Chain 5 Feet	1	14.98	Amazon	14.98
1" Low Profile Casters Wheels	8	3.00	Amazon	24.00
Electric Gear Motor 12v DC 100 RPM	1	106.38	Amazon	106.38
M2 Brass Spacer Standoff Screw	10	0.05	Amazon	0.50
Plastic Storage Tray Basket	1	6.61	Amazon	6.61
USB-C to USB-A 3.3 foot cable	1	12.95	Amazon	12.95
M5 x 15mm x 1.2mm 316 Stainless Steel Washers	4	0.25	Amazon	1.00
J-B Weld 8281 Professional Size Steel Reinforced Epoxy	1	15.29	Amazon	15.29
T6 Aluminum Round Straight Tubing	1	29.34	Amazon	29.34
TMH Door Brush Sweep - 3 inch brush	1	44.99	Amazon	44.99
High Speed HDMI Cable	1	4.95	Amazon	4.95
HDMI to Mini HDMI Adapter	1	8.49	Amazon	8.49
Total				**386.14**

Production Cost per Unit

Part	Quantity	Cost/unit	Source	Total
Intel Realsense D435	1	279.94	Bottom Line Telecom	279.94
Intel i7 5557U NUC 8GB RAM	1	300	Ebay	300.00
Kobuki Mobile Base	1	996.82	Generation Robots	996.82
500 mm T Slot 2020 Aluminmum Extrusion	9	2.49	Amazon	22.41
Cast Iron Pillow Block Mounted Bearings	2	6.22	Amazon	12.44
35 ANSI No., 3/8" Pitch, 15 Teeth, 1" Bore Sprocket	2	7.57	USA Roller Chains	15.14
Alumimum Corner Bracket	24	0.82	Grainger Industrial	19.68
Aluminum T-Slot Nut	80	0.1975	Global Industrial	15.80
M5x8 mm Hex Socket Cap Screw Bolt	80	0.051	Fastener Superstore	4.08
L298N Motor Driver Controller Board	1	2.30	Makerfabs	2.30
35 Roller Chain 5 Feet	1	7.55	PGN Bearings	7.55
1" Low Profile Casters Wheels	8	1.44	MapCaster	11.52
Electric Gear Motor 12v DC 100 RPM	1	56.25	Grainger Industrial	56.25
M2 Brass Spacer Standoff Screw	10	0.0022	Fastener Superstore	0.03
Plastic Storage Tray Basket	1	2.47	Grainger Industrial	2.47
USB-C to USB-A 3.3 foot cable	1	0.60	ByteCable	0.60
M5 x 15mm x 1.2mm 316 Stainless Steel Washers	4	0.036	Global Industrial	0.15
1.6" Tube Brush	1	14.00	Global Industrial	14.00
High Speed HDMI to Mini HDMI Cable	1	3.35	Cable Whole Sale	3.35
Total				**1764.8**

Note: Many of the items purchased for the prototype were sold as sets. We have broken down

and prorated the sets into the costs of their individual components (which is how we would order them in bulk) to more realistically reflect the production cost

Standards and Constraints

The software components communicate using ROS, which is an open-source robotics middleware suite. ROS allows us to create nodes that can send and receive information or requests from each other. This allows processes to publish real-time information like images to ROS topics, or subscribe to topics in order to receive this information. Each of the software tasks, mapping, navigation, greedy pickup, and mode control, represent a ROS node. These nodes publish and subscribe to each other to pass or save information, and communicate in general. Although ROS was not a requirement, it is a standard in the robotics field. This middleware suite allows software engineers to utilize low level controls and pass messages between different robotics systems or components. ROS was a dependency or easily integrated into the other packages used to map and navigate the environment, including ORBslam. The software constraints we had to deal with included the versions of software that were used, as well as the turtlebot hardware. The turtlebot was borrowed from a lab along with the NUC computer and the external battery bank to power the robot. Due to the $700 budget, these components were not able to be replaced or updated. At the beginning of the project, the group was under the impression that the version of Ubuntu could not be changed on the NUC, as well as the version of ROS that was running on the drone we were testing with. This caused many issues with utilizing versions of Ubuntu, ROS, ORBslam, and other software packages that were outdated and were not compatible anymore. Although a drone was ultimately not used and the Ubuntu version on the NUC was able to be changed, the project was too far in to update all versions of the dependencies used.

On the hardware side, both the caster wheels used to build the collection mechanism and the wheels on the Turtlebot slipped on surfaces that were too smooth and were not capable of rough outdoor navigation. The wheels of the turtlebot also helped provide odometry which was not as accurate on super smooth surfaces. The rotary brush on the front of the collection mechanism that was used to pull in trash found by the robot was supposed to originally be purchased off the shelf. After reaching out to multiple companies, the team was not able to purchase the correct sized brush due to supply chain issues as well as quantity constraints, since we only wanted to purchase one to two. The team built the rotary brush from scratch instead by cutting an aluminum pipe to size and using JB weld as an adhesive to connect pieces of brush around the tube. Lastly, the bearings used to connect the rotary brush to the collection mechanism and allow the piece to freely spin had very large outer casings. Larger pieces of trash such as water bottles and cans would occasionally get stuck under the bearings and not be able to be collected properly. Due to the constraint of the current bearings, brush, and collection mechanism not being able to be replaced, the problem was solved by adding bent plastic pieces to the sides of the collection bin that funneled trash into the rotary brush. The plastic pieces sat under the bearings and funneled trash in front of the bearings, instead of under them where the trash could get stuck or pushed out of the way.

References

[1] abhineet123. *ORB SLAM2*. URL: https://github.com/abhineet123/ORB_SLAM2. (Accessed: 11-June-2021).

[2] appliedAI-Initiative. *ORB SLAM2 ROS*. URL: https://github.com/appliedAI-Initiative/orb_slam_2_ros. (Accessed: 11-June-2021).

[3] Keep Louisiana Beautiful. *Executive Summary: Litter in America*. URL: https://keeplouisianabeautiful.org/wp-content/uploads/2015/09/Litter-in-America-Executive_Summary_-_FINAL.pdf. (Accessed: 10-June-2021).

[4] Heather Brown. *Good Question: Who Picks Up The Trash Along The Highways?* URL: https://minnesota.cbslocal.com/2017/04/17/gq-highway-trash/. (Accessed: 10-June-2021).

[5] Jonathan Chaplin, Sue Lodahl, and Dewayne Jones. *Designing a Machine for Picking Up Litter Along Minnesota Highways*. URL: https://www.lrrb.org/pdf/200821TS.pdf. (Accessed: 10-June-2021).

[6] John Chiaramonte et al. *Capstone-W3*. URL: https://github.com/Capstone-W3. (Accessed: 11-June-2021).

[7] hellovuong. *ORB SLAM odom*. URL: https://github.com/hellovuong/ORB_SLAM_odom. (Accessed: 11-June-2021).

[8] Iclebo Kobuki. *Kobuki User Guide*. URL: http://kobuki.yujinrobot.com/wiki/online-user-guide/. (Accessed: 11-June-2021).

[9] Stephanie Melchor. *Roadside Trash A Growing Problem*. URL: https://www.montereyherald.com/2021/01/23/roadside-trash-a-growing-problem/. (Accessed: 10-June-2021).

[10] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. DOI: 10.1109/TRO.2015.2463671.

[11] Raúl Mur-Artal and Juan D. Tardós. "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262. DOI: 10.1109/TRO.2017.2705103.

[12] Inc. Open Source Robotics Foundation. *TurtleBot2*. URL: https://www.turtlebot.com/turtlebot2/. (Accessed: 11-June-2021).

[13] raulmur. *ORB SLAM2*. URL: https://github.com/raulmur/ORB_SLAM2. (Accessed: 11-June-2021).

[14] rayvburn. *ORBSLAM2 ROS*. URL: https://github.com/rayvburn/ORB-SLAM2_ROS. (Accessed: 11-June-2021).

[15] Clearpath Robotics. *TurtleBot 2 - Open Source Personal Research Robot*. URL: https://clearpathrobotics.com/turtlebot-2-open-source-robot. (Accessed: 10-June-2021).

[16] Adrian Rosebrock. *OpenCV Morphological Operations*. URL: https://pyimagesearch.com/2021/04/28/opencv-morphological-operations/. (Accessed: 11-June-2021).

[17] UZ-SLAMLab. *ORB SLAM3*. URL: https://github.com/UZ-SLAMLab/ORB_SLAM3. (Accessed: 11-June-2021).

[18] H. K. Sparling. *Ohio Roads Are Covered in Trash. And It's a Huge Waste of Money and Time*. URL: https://www.cincinnati.com/story/news/2019/07/08/ohio-roads-trash-litter-pickup-odot/1548598001. (Accessed: 10-June-2021).

[19] Makerbot Thingiverse. *Intel RealSense D435 Camera Mount for OpenBuilds 20x20mm Extrusion*. URL: https://www.thingiverse.com/thing:2965905.

[20] Tossy0423. *darknet ros*. URL: https://github.com/Tossy0423/darknet_ros/tree/master/. (Accessed: 11-June-2021).