Error Tolerant Multi-Robot System for Roadside Trash Collection

1st Lee Milburn
College of Engineering
Northeastern University
Boston, Massachusetts
milburn.l@northeastern.edu

2nd John Chiaramonte
College of Engineering
Northeastern University
Boston, Massachusetts
chiaramonte.j@northeastern.edu

3rd Jack Fenton
College of Engineering
Northeastern University
Boston, Massachusetts
fenton.j@northeastern.edu

***Abstract*—In this paper, we present the first iteration of an error-tolerant, autonomous, multi-robot system that monitors highway road verges and identifies and collects roadside litter. It is designed to use an aerial vehicle that can rapidly cover a vast area and collect data on the road verge. This data is then passed to a ground vehicle that constructs a map of the road verge and uses a trained Convolutional Neural Network (CNN) to identify pieces of litter. After the pieces of litter are identified on the map of the road verge, the ground robot navigates to each piece of trash, re-evaluates the area, and performs a "greedy pickup" procedure. This final stage accounts for any error in the map's construction or the identified trash's location. We found that ending the robotic system's control flow with a greedy pickup procedure can retroactively account for processing errors of the system as it runs. This increases the system's fault tolerance and allows for the use of cheaper equipment since pinpoint accuracy is not always necessary. In this paper, we present the feasibility of this system by testing in simulation and later using real robotic hardware. We show that the system is effective enough to iterate on its design principles to create a more sophisticated system.**

***Keywords* —Autonomous trash collection, Environmental monitoring, Error tolerance, Multi-robot system**

I. Introduction

Roadside trash is a massive issue currently managed by manual labor - a woefully inadequate solution [7]. Despite being a nationwide issue, the task of waste management is mostly under the jurisdiction of municipalities and it garners little to no attention or investment.

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 [2]. The results indicate that there are 51.2 billion pieces of litter on roadways nationwide. Of this, the majority (91%, or 46.6 billion pieces) is less than four inches.

In Monterey City, California, complaints about trash have increased since the start of the COVID-19 pandemic. 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 his cleanup crew was so busy cleaning up from illegal dumpsites that they did not have time to walk the shoulders of their roads to pick up loose trash [14]. Caltrans, California's public transportation department, has been faced with the same problem. Kevin Drabinski, public information officer for Caltrans District 5, said it's important to Caltrans that they manage litter because of safety and environmental concerns. Caltrans spends $50 million annually on litter cleanup [14].

To address this issue, our multi-robot system uses a three-stage approach to autonomously map, identify, and pick up trash. The three modes are Mapping, Navigation, and Greedy Pickup. We would first have a lightweight drone fly over a specified area on the road and stream its visuals to a ground robot. That robot would generate a map using the drone's input and identify trash pieces on that map. The ground robot would then navigate near each piece of identified trash and then switch to the greedy pickup mode where it scans the area for the suspected piece of trash. After a piece of trash is re-identified locally, the robot moves and collects it. Once either collected or not found, the ground robot then moves to the next piece on its map.

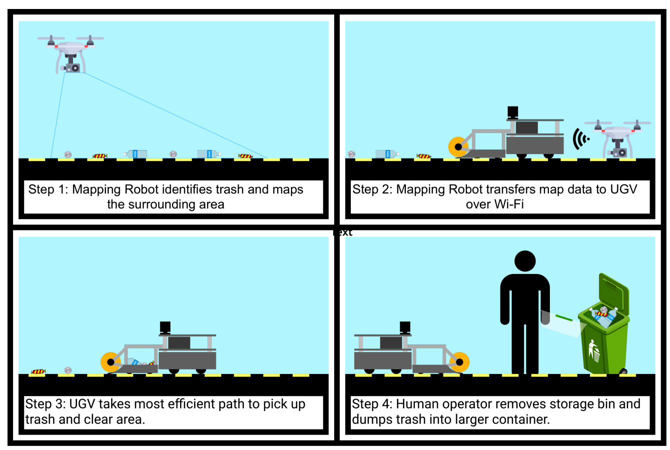

Fig. 1. System Design Overview

Our approach allows for accurate pickup without the need for massive processing power or overly expensive sensors. The system's configuration used the open-source convolutional neural network You Only Look Once (YoLOv4) for image identification [24], the open-source visual SLAM solution ORBSLAM-2 for map-building [19], and open-source ROS navigation software for path planning and navigation. We simulated this system using Gazebo [13] and after receiving consistent results, tested it in a real-world environment. Our real-world results, with a relatively low-powered system, indicate that our approach is a proof-of-concept for a scalable and viable solution to the growing worldwide litter problem.

II. Related Work

There has been research into multi-robot systems used for environmental monitoring. The research for these systems finds that multi-robot systems pose a more effective solution to surveying an environment than static monitoring [6]. There is also research into multi-robot systems that do autonomous trash collection [16]. This research concludes that for maximum efficiency, robots should be aware of their environment when trying to collect trash as opposed to making decisions based solely on their field of view (FOV). Therefore, these two systems, monitoring an environment, and using a collection algorithm to pick up trash in a dynamically changing environment could be combined to create the most effective version of an autonomous collection system. A version of this system has been created to autonomously collect and monitor plastics in rivers [9]. The system includes a central processor that takes in the necessary tasks of the environment and assigns those tasks to underwater autonomous vehicles that then pick up the plastics. This system concluded that a Multi-robot task allocation architecture [11] with a controlling center increased the efficiency of the system, but the hardware for working effectively in that environment would have to be improved for more effective use. Our research team structured our multi-robot system design to have a robot monitor the environment and wirelessly transmit its environmental depiction to a UGV that would collect the trash.

For our system to be cheap and lightweight, existing software was needed that works in real-time on standard CPUs in a wide variety of environments. The Robotic Operating System (ROS) [17] is an open-source robotics framework that allowed each of our hardware and software components to communicate freely in real-time, and each software component used was compatible with this framework. ORB-SLAM2 was the ideal solution for mapping [18][15]. It uses differing angles of static environmental features to create a map and a keyframe-based SLAM approach that reduces the overall data size of the SLAM map considerably [1]. Since the system is designed with visual sensors, a software to visually identify trash was necessary. YoLOv4 is a CNN model trained from annotated images to place bounding boxes around specified objects in RBG images. Adaptive Monte Carlo Localization (AMCL) is the method of navigation used as well as the name of a compatible software stack used for navigation provided by ROS [8]. AMCL takes in odometry feedback from the robot's wheels and scan data derived from the RGB-D Camera to navigate. After some static conversions from ORB-SLAM2's native map format to a 2D occupancy grid, AMCL can autonomously navigate around an environment. These existing software stacks served as the framework for the multi-robot system to be built.

III. System Design

A. System Overview

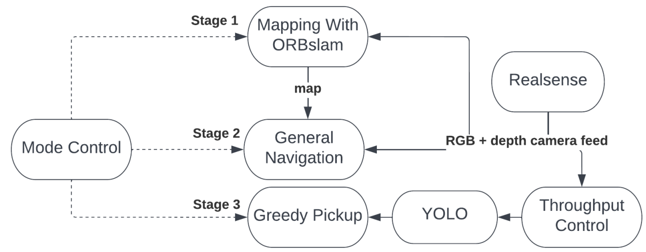

Fig. 2. Systems' Communication Flow

After initialization by a human operator, the mapping robot will scan an area with a visual sensor. This sensor data will be compiled using Simultaneous Location and Mapping (SLAM) technology to create a continuous digital map of the target area which will then be wirelessly transmitted to the unmanned ground vehicle (UGV). The UGV will identify pieces of trash in the environment using computer vision algorithms and construct a two-dimensional map populated with target coordinates of identified trash. The UGV will then create an efficient path between the target coordinates in the map. Once the UGV sets off on the calculated path, it will confirm the trash location using an onboard visual sensor and proceed to pick it up. Once the UGV has completed its rounds or the bin is detected as full, it will return home, and a human operator will empty the bin.

B. Mode Controller

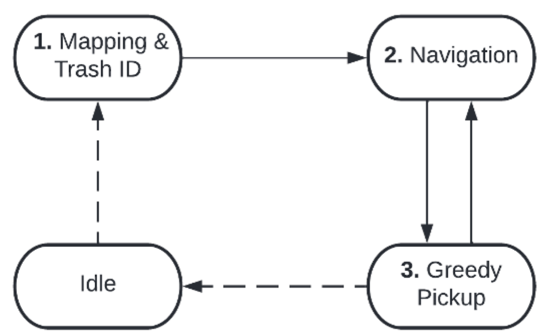

Fig. 3. Mode Controller Flow

The "Mode Controller" was created to switch between the three separate software components of the system: Mapping, Navigation, and Greedy Pickup. The Mode Controller is a ROS node that communicates with the Mapping, Navigation, and Greedy Pickup nodes, turning them on or off as needed. The Mode Controller starts in idle before putting the system into the Mapping mode. Once mapping finishes, the mode controller turns off Mapping mode. The map is then passed on to the Navigation mode alongside the coordinates of identified trash. The Mode Controller then turns on Navigation mode. Once a trash coordinate has 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. Navigation and Greedy Pickup modes will alternate until all trash is removed from the environment. Once all the trash is picked up and no marked coordinates remain on the map, the Mode Controller turns back to idle and awaits further instruction.

C. Mapping

The system navigates the surrounding area and maps its environment using ORB-SLAM2. 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 [21]. In experimentation, the maps were initially filled with noise that led to an inability to navigate the space, figure 4. Morphological operations are commonly used tools in image processing to clean up an image. By “eroding” and “opening” the space, errant data points that were being misidentified as occupied were removed. By “closing” the space gaps caused by the sparse data, holes in our map were closed, and smooth, continuous maps were generated, figure 5. The product was an occupancy grid very close to real-world surroundings with a real-time, lightweight mapping solution.

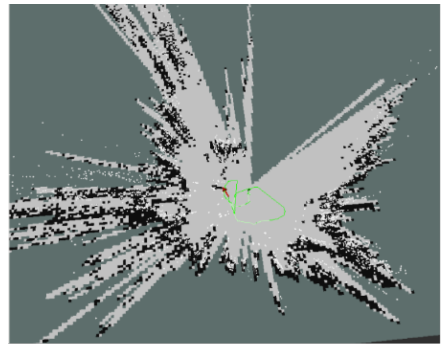

Fig. 4. Raw Occupancy Grid

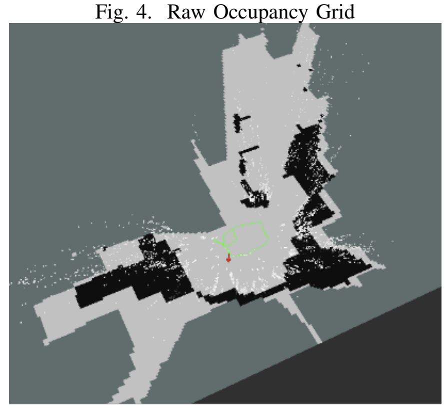

Fig. 5. Morphologically Transformed Occupancy Grid

D. Trash Identification

Simultaneously, as an area is being mapped, the system also detects trash. To recognize where on the map a piece of trash is, the mapping robot first finds the location of a piece of trash relative to itself. The system to locate trash was devised using multiple components: YOLOv4, the odometry data of the robot, and the depth camera feed provided by the Realsense RGB-D camera, figure 7.

The first step in the trash identification pipeline is image identification using YOLOv4. YOLOv4 is a convolutional neural network that we trained with a custom dataset of over 1000 images, each taken of varying pieces of trash from the perspective of the robot. Each image was hand-labeled and fed into the machine learning model using an 80-15-5 split between training, validation, and testing sets. The model was trained and runs in our software stack using a customized open-source ROS wrapper for YOLOv4 [24]. The image identification model runs simultaneously while the environment is being mapped using the RGB camera feed and returns “bounding boxes” around identified trash pieces in the image, providing coordinates relative to the camera’s image frame, figure 6.

These bounding boxes provide 2D pixel locations for the trash in the image but do not contain any information about where the trash lies in the environment. Therefore, the next step 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.

The FOV angle is 69.4 degrees, its opposite side is 640 pixels, and it is known to be an isosceles triangle, the remaining side lengths and angles can be extrapolated as this is considered a trigonometrically “solved” triangle. Using this triangle the angle of the identified trash piece is calculated using the inverse tangent function, as shown in the following equation and in figure 9.

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

The next step in the localization process is to determine the distance between the camera and the piece of trash. This is accomplished using the depth camera feed provided by the 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.

The depth picture can be indexed as a matrix using the 2D 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

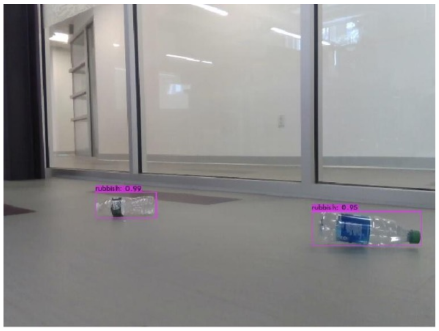

Fig. 6. Trash Bounding Boxes

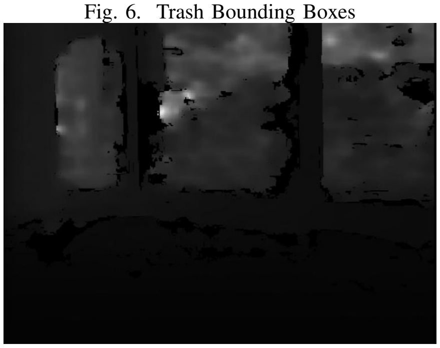

Fig. 7. Depth Feed

trash relative to the robot has been acquired. Using a second triangle with coordinates in meters, both the angle of the trash relative to the robot as well as the distance between the trash and the robot can be extrapolated.

The first unknown variable encountered is the distance between the trash and the center of the robot base, d. Using the distance between the camera and the center of the robot base s, as well as the distance between the trash and the camera taken from the depth camera feed ($depth$), d can be solved using the Law of Cosines as shown below.

$$c^2 = a^2 + b^2 + 2ab\cos(c) \tag{2}$$

$$c = \sqrt{a^2 + b^2 + 2ab\cos(c)} \tag{3}$$

$$d = \sqrt{(depth^2 + r^2 + 2(depth)(s)(\cos(180^\circ - \theta))} \tag{4}$$

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

$$\frac{\sin X}{x} = \frac{\sin Y}{y} \tag{5}$$

$$\frac{\sin(\beta)}{depth} = \frac{\sin(180^\circ - \theta)}{d} \tag{6}$$

$$\beta = \sin^{-1}\left(\frac{depth\sin(180^\circ - \theta)}{d}\right) \tag{7}$$

Once the angle to the piece of trash relative to the robot and the distance between these two points became known, these values were added to the robot's current position to realize the piece of trash on the map. However, some difficulties arose when the computer did not process the images fast enough.

Algorithm 1 Mapping Trash to a Map

Input: Robot's path r in the map m, YOLO Bounding Box b
Output: Pose of piece of trash in the map p, orientation o of robot relative to p

```
confident pieces cp ← empty
for every item in b do
    if items i's trash confidence is greater than ct then
        cp ← i
    end if
end for
yolotimestamp yt ← b[0].timestamp
for pose pr in r do
    time difference td ← abs(yt − pr)
    if you don't have a closest pose timestamp cpr to the yolo timestamp yt then
        cpr ← pr
        smallest time difference std ← td
    else
        if td < std then
            std ← td
        end if
    end if
end for
for each timestamp, image in depth camera history do
    Find the closest depthimage di taken to b
end for
Set robots orientation o from when the picture was taken
for trashpiece tp in cp do
    get distance d of tp from di
    get tp's angle theta from robot base
    trash x distance tdx ← o + (d ∗ cos θ)
    trash y distance tdy ← o + (d ∗ sin θ)
    p ← tdx, tdy
end for
```

YOLOv4, when run on the Intel NUC, processed images at a throughput of 0.5-0.8 FPS with about 4-5 seconds of latency from when the image was originally taken. This created a large gap between the time when the image was taken and the current position of the robot. To account for the processing latency, the path of the robot as it was mapping is logged with timestamps for every position in its path from ORB-SLAM2. Once the mapping robot received a successful trash detection, the ROS timestamp given from YOLOv4 from when that image was taken was passed to the path, and a `Pose` is output. It is from this `Pose` that distance d and angle β are added to localize the piece of trash relative to the map itself.

In the figure 11, 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, getting an approximate location

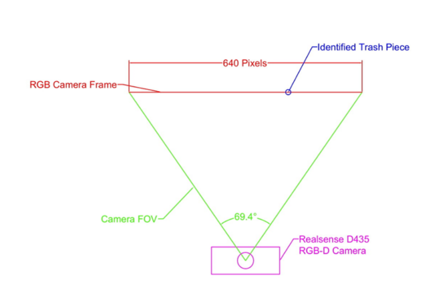

Fig. 8. FOV Diagram

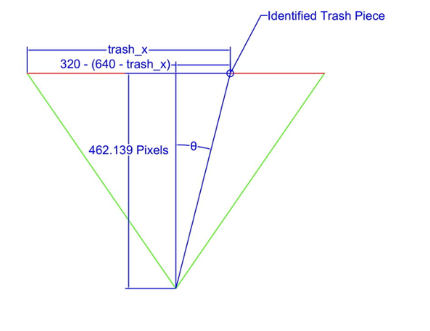

Fig. 9. FOV Trignometric Calculations

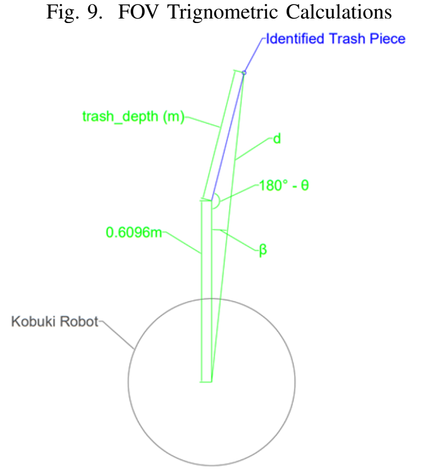

Fig. 10. Robot-Trash Trignometric Calculations

of the piece of trash.

E. Anti-clustering

Initially, we found consistency issues with the trash identification. Either images of the same piece of trash were processed more than once, or the trash's estimated position became inaccurate as the SLAM map updated. This problem led to a large amount of noise, causing up to and exceeding thirty detections for two pieces of trash in one single trial. In some limited cases, our YoLOv4 model would also erroneously classify a random background object as trash. To sort through the noisy detections, each new trash detection was run through a filter. Every time a piece of trash was detected, a ROS subscriber would listen to the detection and determine if it was

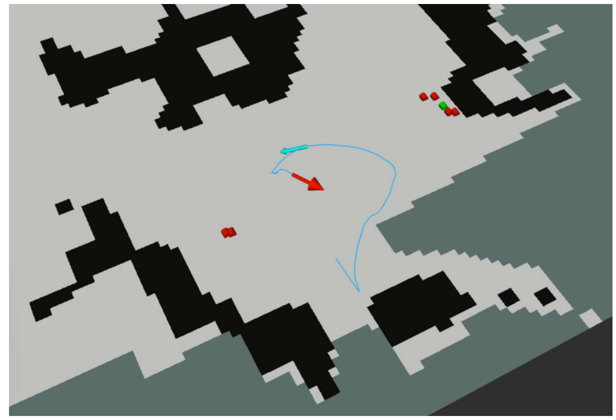

Fig. 11. Robot Trajectory in the map

a new piece or detection of a piece of trash already found. A clustered piece of trash is denoted by the green mark in figure 11.

To accomplish this, all the detected pieces of trash were stored at their initial positions. If any new trash detection was within a set radius of a previously detected piece of trash, the new trash detection became combined with the established piece by taking a rolling average of the detections. The calculations are seen in the equation 8, where $p_{1x/y}$ is the existing trash detection's respective x and y coordinate, $p_{2x/2y}$ is the new trash detection's x and y coordinates, and a is the amount of times p_1 has been averaged to that point.

$$p_{1x} = \frac{p_{1x}a + p_{2x}}{a+1} p_{1y} = \frac{p_{1y}a + p_{2y}}{a+1} \tag{8}$$

Algorithm 2 Anti-Clustering Algorithm

Input: Trash Pose tp
Output: Poses of clustered trash pieces ctp
averaged Trash poses atp tupled with times averaged ta
$(atp, ta) \leftarrow empty$
if atp = 0 **then**
  $atp \leftarrow tp$
  $ta \leftarrow 1$
else
  for every pose tuple pt in $atpt$ **do**
    get x and y bottom and top around the trash's location xb, xt, yb, yt
    if $xb \leq tpx \leq xt$ & $yx \leq tpy \leq yt$ **then**
      pose tuple x ptx = (ptx * ta + tpx) / (ta + 1.0)
      pose tuple y pty = (pty * ta + tpy) / (ta + 1.0)
      ta + 1
    end if
  end for
  if tp is not in atp **then**
    $atp \leftarrow tp$
  end if
  $ctp \leftarrow$ all trash poses tp in atp where tp_i's $ta_i > 2$ times
end if

This anti-clustering algorithm decreased the total amount of detections to accurately reflect the number of trash pieces seen in the environment. To avoid erroneous trash detections, average trash locations without enough detections were determined as "noisy" and filtered out. Trash points were only published to the navigation stack if it had three or more detections averaged to one point.

F. General Navigation and Path Planning

General navigation consists of two parts: localization and path planning. The Robot first receives the 2D occupancy grid from our mapping software, alongside the trash detection coordinates. Once these data are received, the robot then navigates to within two meters of the nearest detected trash point. Navigation only needs to navigate near a trash location since greedy pickup is routinely effective within a two meter distance, and the anti-clustering algorithm accounts for noise in our trash detections.

To get the goal pose g you need it's orientation, and x/y coordinates. The equation 9 describes how to get the angle for the pose where p_1 is the starting robot pose and p_2 the trash pose. In equation 10 the distance the goal pose is from the robot is calculated to have it within Greedy Pickup range, where d_3 is the goal distance, d_1 is the distance between the trash and the robot, and d_3 is Greedy Pickup's range. In equation 11 the coordinates of the goal pose is found where $g_{x/y}$ is represents the coordinates respectively.

$$\theta = \arctan(\frac{p_{1y} - p_{2y}}{p_{1x} - p_{2x}}) \tag{9}$$

$$d_3 = d_1 - d_2 \tag{10}$$

$$g_x = d_3 + p_{1x}\cos(\theta) g_y = d_3 + p_{2x}\sin(\theta) \tag{11}$$

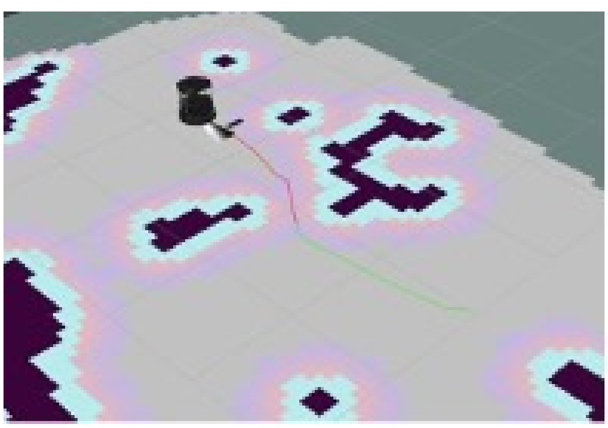

Fig. 12. Robot Path Planning in Rviz

The Navigation and path planning stack was based on the ROS-provided open-source AMCL software stack. This software loads and localizes the robot in a mapped environment and its DWA planner creates a path between the identified points of trash. We created another software module that feeds our target coordinates from our trash detection phase into AMCL's path planner to follow the most efficient path between the robot's current location and the nearest possible trash point, figure 12.

G. Greedy Pickup

Once the Navigation portion of the software stack places the robot within 2 meters of the piece of trash, Greedy Pickup is activated. Greedy Pickup is an asynchronous algorithm that ignores all navigation and map factors and solely focuses on seeking out the nearby trash directly.

When the greedy pickup is activated, it rotates in a given direction to look for trash using YoLOv4. Once it receives a trash detection, it calculates the position of the trash using the same algorithm explained in the Trash Identification section. After the robot localizes the trash, it turns back towards the trash at its precise angle, turns on the collection mechanism's motor, and moves exactly 0.2m past the trash's location to ensure proper collection. Once this occurs, the motor turns off and navigation resumes.

Algorithm 3 Greedy Pickup

Input: Detected Trash pose list tpl
set timeout time to
set confidence threshold ct
for Trash pose tp in tpl **do**
  Determine whether tp is to the left or right of the robot in the Map
  Spin in direction of tp, scanning for confirmation
  if Robot gets $tp \geq ct$ **then**
    Robot Stop spinning
    Use III-D algorithm to find relevant Robot orientation and trash pose
    Get destination angle
    Robot turns to destination angle
    Robot Turn on collection mechanism and drive over detected piece of trash
  else
    Robot exceeded to looking for trash
  end if
end for

For the collection mechanism to turn on or off, the NUC sends 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 Robot base 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 robot base. To eliminate this 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.

H. ROS Middleware

ROS was used to connect all the functions of this system. Every design block in figure 2 functions as a node, or multiple nodes, which subscribe and publish information to the other nodes. ROS would also be used to network between the different robots in the multi-robot system over WiFi or other wireless protocols.

IV. Experiment

A. Goal

The purpose of this experiment is to assert the feasibility of this system design before iterating on the hardware used to make it scalable and adaptable. 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.

The robotic system is meant to 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 [20], 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 UGV operates in narrow environmental parameters. The weather must be clear with no rain since the electronic systems onboard the UGV are non-weatherproofed. 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.

B. Testing In Simulation

Simulated testing was done in the Gazebo Robotics simulator, 13. This simulator was included in the base Turtlebot SDK and includes a near true-to-life recreation of the entire turtlebot system. The use of ROS allows for 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. All data associated with movement and mapping were recorded and played back in a simulated environment to recreate and reevaluate our physical testing. This allowed for useful visualizations and assessments of what the robot was processing at any given time.

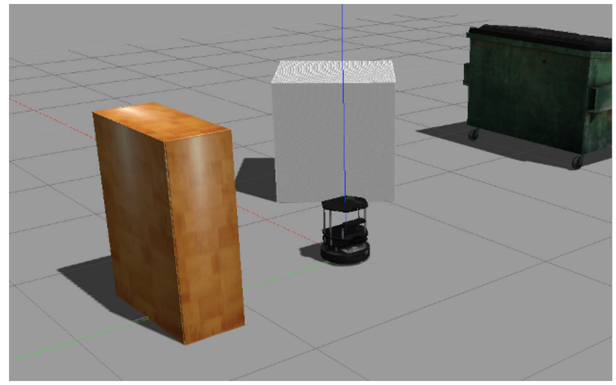

Fig. 13. Simulation Testing Still

C. Testing the Hardware

1) Experiment System: The system's design is meant to function with a mapping robot and a ground vehicle. To simulate the mapping vehicle in this experiment the UGV plays both roles. The UGV first passed through the area to get a map and identify trash. Then taking that information to navigate and pick up the trash.

2) The Robot: The hardware design uses a modified Turtlebot-2 as a base design, figure 15, that has four main components: the depth camera (480p RGB-D Intel Realsense D435) [10], the computer (7-year-old Intel NUC with 8GB of RAM) which runs a GNU-Linux OS along with ROS to manage sensor data collection and real-time processing, the mobile robotic base (Kobuki Mobile Base) [12], and a custom-designed collection mechanism. The camera relays RGB and depth images which are processed to identify and target trash. The Kobuki's motor and wheels relay odometric feedback that helps confirm the UGV'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.

3) The 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. To allow free range of motion, caster wheels were affixed to the bottom of the frame. When the collection mechanism motor is activated, it sweeps trash up a ramp into a plastic storage container. A camera mount was printed so the Realsense could be attached to the front of mechanism [23]. A funnel was added to the front of the collection mechanism to rein in the trash that may have been missed slightly, figure 18.

The design of the electronics system for the collection mechanism is an Arduino Mega that is connected to the NUC using a USB cable, figure 17. The 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 from a 12V, 1.5A Max port. The output of the L982N is the DC motor which drives the chain and the brush.

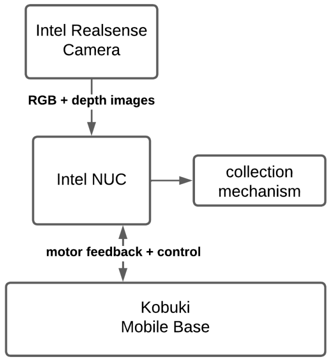

Fig. 14. Hardware Overview Diagram

Fig. 15. Turtlebot2 Base System

Fig. 16. Custom Designed Brush

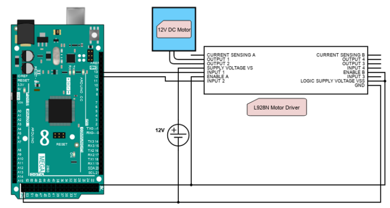

Fig. 17. Motor Circuitry

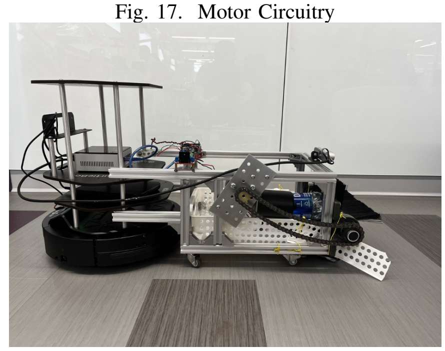

Fig. 18. Final Construction

The brush itself was hand-designed and fabricated since there is no commercial brush model that fit the design specifications for the mobile robot, figure 16.

D. The Environment

The environment where the robot was tested was a room with a random configuration of chairs and obstacles put around an open space. Therefore, the map would be created each time in a dynamic environment and the system would have to account for a new configuration. In that space, trash was put in different locations for all tests. We tested up to four pieces of trash in the environment at a time.

E. Tests

1) Testing Mapping/ Image Identification accuracy/ Accounting for Latency: To test the trash detection and how well the processing latency was accounted for, expected maps with the approximate trash locations were checked against the created ones.

2) Testing the Greedy Pickup Algorithm: To test Greedy Pickup, the mode was enabled with varying pieces of trash within two meters of the robot. The piece of trash would start out of the UGV's perception range. The UGV would have to scan for the trash, identify it and then collect it. This trial was run with trash at 2 meters, 1 meter, and half a meter distance from the UGV.

3) Full System Trials: Full system trials were then conducted, starting with the mapping of an environment and labeling trash points in that environment. Then to navigating to the labeled pieces of trash in the environment and collecting the pieces of trash using Greedy Pickup.

V. Results

A. Accounting for Processing Latency

The map creation and the image identification, figure 23, were shown to be an accurate, figure 19, fast and lightweight way of monitoring an area and identifying pieces of trash.

B. Greedy Pickup Results

Greedy pickup is shown to be accurate to an extent with a success rate of 77.78%, figure 20. There were multiple failures in these trials that were caused by the design of the collection mechanism. The collection mechanism was not equipped with odometric wheels that could relay its' location so, at times it would overturn, and the caster wheels introduced an error the system was not aware of and could not account

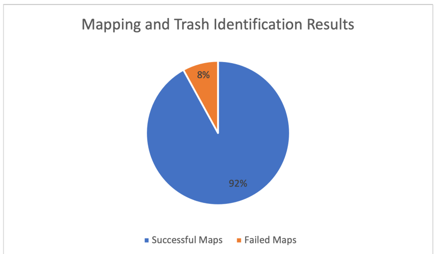

Fig. 19. Accurate Map Creation with Identified Trash Points Result

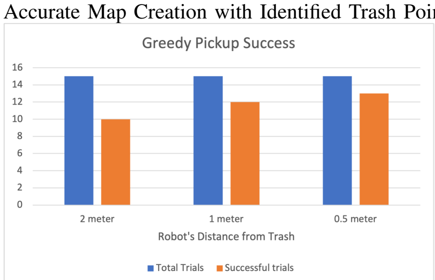

Fig. 20. Greedy Pickup Results

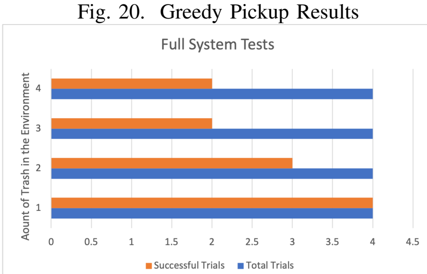

Fig. 21. Full System Run-through Results

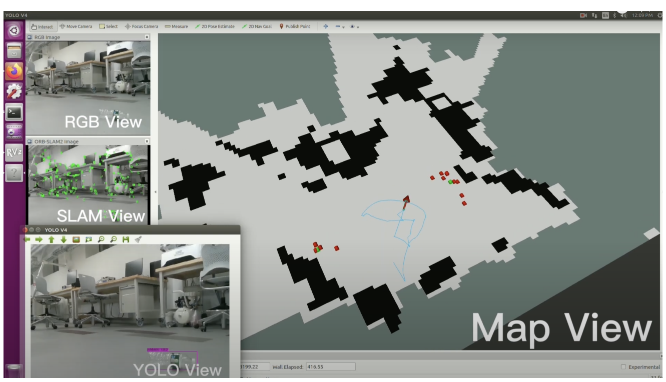

Fig. 22. Real Time Trash Identification and Map Creation

for. This caused the UGV to over or under turn occasionally and not successfully collect the trash. The Greedy Pickup algorithm has shown the ability to increase the error tolerance of the system, however, there are more improvements to the algorithm that could be made to make it more error-tolerant. For example, an added "lock on" mechanism that when a piece of trash is in sight it would keep the piece of trash in the center of its FOV as it moves forward to collect the piece of trash. Ensuring that the failure-causing edge cases are accounted for.

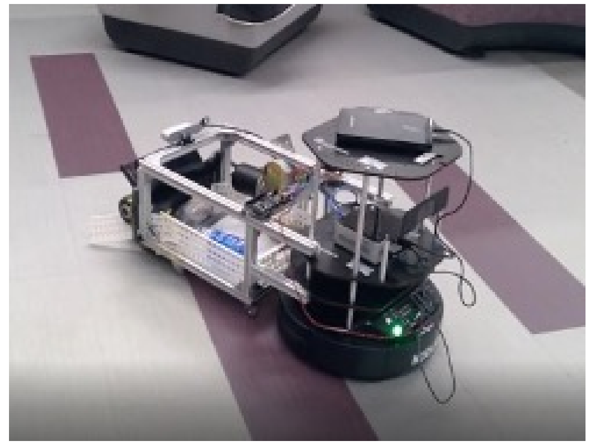

Fig. 23. Still of UGV Picking up Water Bottle in System Test

C. Full System Results

These results, with a 68% success rate, figure 21, show that there are possible errors that can be introduced to this system and that they compound on themselves as the task gets more complex. During these trials, the collection mechanism introduced errors during navigation and greedy pickup. The collection mechanism introduced errors by being slightly out of position. Since the Collection mechanism did not have sensor feedback to tell the UGV it was not in the correct position the little errors compounded into failures to collect trash.

VI. Conclusion

Our contribution is a multi-robot system design applied to monitoring and managing roadside litter. We developed a system that can map, identify, and pick up pieces of trash. The system is designed to be relatively cheap and scalable which could be done because we introduced different algorithms that account for errors in a system that has less precise sensors. Our Greedy Pickup algorithm accounts for errors in trash identification, and trash can be accurately identified in a system with low processing power. Our custom-designed collection mechanism, designed to pick up the main offending types of trash found on the side of the road, was also introduced.

This system builds off previously known algorithms: AMCL and DWA for navigation, Orbslam-2 for map creation, and YOLO.v4 for trash identification. It also build off multi-robot systems to monitor a dynamic environment and combines that with information with a dynamic collection algorithm, Greedy Pickup, to create an efficient collection process.

The next steps for this system would to be refine the mechanical design of the UGV, such as adding odometry sensors onto the collection mechanism's wheels and improving the wheels to be able to drive over more difficult terrains. There are also improvements to be made to the Greedy Pickup algorithm such as adding a "Lock on" capability. Better logic could be applied to the navigation, for example, a future iteration of this system could use a graph search algorithm to find the most efficient path between every piece of trash. An aerial robot can also be introduced as the mapping robot in the next iteration of this system for testing, to more accurately represent the issues that would arise from changes in the perspective of the system. More replicas of the UGV could be added into the system as well, so there would be multiple trash collectors in one environment at the same time. The system could also have mapping and trash collection happening in parallel instead of the modes being sequential.

VII. Acknowledgments

This is a video of a full run-through our system did with two pieces of trash in the environment [5]. This is the central repository in our organization that sets up the system [4]. This is the Greedy Pickup repository which holds all the functions laid out in this paper [3]. We would like to thank the contributions of Divya Venkatraman, Jared Raines and Catherine Ellingham for their work on gathering data, editing the paper and system networking. We'd also like to thank Dr. Taskin Padir and his Robotics and Intelligent Ground Vehicle Research Laboratory (RIVeR) [22] for allowing us to use their hardware to test our design.

References

[1] Yong-bao Ai et al. "Visual SLAM in dynamic environments based on object detection". In: *Defence Technology* (2020). ISSN: 2214-9147. DOI: https://doi.org/10.1016/j.dt.2020.09.012. URL: https://www.sciencedirect.com/science/article/pii/S2214914720304402.

[2] 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).

[3] John Chiaramonte, Jack Fenton, and Lee Milburn. $caveman_mode$. URL: https://github.com/Capstone-W3/caveman_mode. (Accessed: 20-April-2021).

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

[5] John Chiaramonte et al. *T.R.A.S.H. Demonstration*. URL: https://www.youtube.com/watch?v=cIwk0Kh9k7E. (Accessed: 5-July-2021).

[6] Maria Valera Espina et al. "Multi-robot Teams for Environmental Monitoring". In: *Innovations in Defence Support Systems – 3: Intelligent Paradigms in Security*. Ed. by Paolo Remagnino, Dorothy N. Monekosso, and Lakhmi C. Jain. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 183–209. ISBN: 978-3-642-18278-5. DOI: 10.1007/978-3-642-18278-5_8. URL: https://doi.org/10.1007/978-3-642-18278-5_8.

[7] Conserve Energy Future. *The Catastrophic Effects of Littering on Humans*. URL: https://www.conserve-energy-future.com/littering-effects-humans-animals-environment.php. (Accessed: 14-June-2021).

[8] Brian Gerkey. *AMCL Package Summary*. URL: https://wiki.ros.org/amcl?distro=melodic. (Accessed: 10-March-2022).

[9] Le Hong, Weicheng Cui, and Hao Chen. "A Novel Multi-Robot Task Allocation Model in Marine Plastics Cleaning Based on Replicator Dynamics". In: *Journal of Marine Science and Engineering* 9.8 (2021). ISSN: 2077-1312. DOI: 10.3390/jmse9080879. URL: https://www.mdpi.com/2077-1312/9/8/879.

[10] Intel. *Intel® RealSense™ Depth Camera D435*. URL: https://ark.intel.com/content/www/us/en/ark/products/128255/intel-realsense-depth-camera-d435.html. (Accessed: 5-July-2021).

[11] Alaa Khamis, Ahmed Hussein, and Ahmed Elmogy. "Multi-robot Task Allocation: A Review of the State-of-the-Art". In: vol. 604. May 2015, pp. 31–51. ISBN: 978-3-319-18299-5. DOI: 10.1007/978-3-319-18299-5_2.

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

[13] Nathan Koenig and Andrew Howard. "Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator". In: ().

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

[15] 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.

[16] Shinkyu Park, Yaofeng Desmond Zhong, and Naomi Ehrich Leonard. "Multi-Robot Task Allocation Games in Dynamically Changing Environments". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 8678–8684. DOI: 10.1109/ICRA48506.2021.9561809.

[17] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*. Kobe, Japan, May 2009.

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

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

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

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

[22] Padir Taskin. *RIVeR*. URL: https://robot.neu.edu/. (Accessed: 7-July-2021).

[23] Makerbot Thingiverse. *Intel RealSense D435 Camera Mount for OpenBuilds 20x20mm Extrusion*. URL: https://www.thingiverse.com/thing:2965905.
[24] Tossy0423. *darknet ros*. URL: https://github.com/Tossy0423/darknet_ros/tree/master/. (Accessed: 11-June-2021).