A System for Autonomous Retrieval of Valued Objects from Hazardous Environments

Mingi Jeong, Le Chang, Yijia Wu, and Julien Blanchet
Department of Computer Science
Dartmouth College
Hanover NH, USA
{mingi.jeong.gr, le.chang.gr, yijia.wu.gr, julien.b.blanchet.gr}@dartmouth.edu

***Abstract*—There are many scenarios in which objects must be retrieved from hazardous environments, such as space mining and disaster relief. Using an open source robotics framework and a skid steer robot, we present a robotic system capable of entering an unknown environment, identifying valuable assets, identifying a safe zone, retrieving the valuable asset, and transporting it to the safe zone. We discuss simplifying assumptions that were made (such as the use of cubes laden with fiduciary markers), as well as trials that we performed and the subsequent improvements that were made. Lastly, we discuss the limitations of our systems and propose future work for this research area.**

I. Introduction

A major advantage of using robots instead of humans for a given task is safety, given that robots are replaceable and human lives are not. This is especially true for tasks that involve substantial amounts of risk, such as find-and-retrieve operations in dangerous environments [1]–[4]. Ideally, one would imagine deploying a robot from a safe location from which the robot could enter the dangerous environment (such as a burning building), locate the target object (such as a stranded pet), and transport the target object back to the safe location without ever putting the human operator at risk. Such a robot needs to be able to perform autonomous mapping, localization, and path-planning in an unknown environment, as neither a stable communication link with the operator nor prior knowledge of the environment are guaranteed. Such a robot would also need to be able to recognize the target object and transport it back to the specified safe zone. Lastly, such a robot would need to detect and avoid potentially dangerous obstacles in order to maximize chances of mission success and to avoid putting the target object (which could potentially be a person) in harm's way.

This paper is structured as follows: Section II analyzes the related works aligned with the goal of this paper, and Section III describes the problem our system aims to address in detail. Our technical approach and implementation are discussed in Section IV. Section V presents the results of several experiments we performed while developing this system. Section VI concludes our paper and presents suggestions for future work.

II. Related Work

Our problem can be broken down into several subproblems - namely, 1) methods for finding and retrieving items, 2) methods for localization, mapping, and navigation in hazardous environments, and 3) methods for transporting - or pushing - objects. We review related work in each of these three subproblems in the sections that follow.

A. Foraging Tasks

Our find-and-retrieve task is essentially a single-robot version of the problem of foraging within the field of cooperative multirobot systems (see [5] for a description). Within this field, Rybski et al [6] examined the effects of localization and communication on foraging time using a team of simple robots that employ a basic "find-grab-return" finite state machine controller. Deneubourg et al [7] examine a method for reducing interference in this task. Much work on foraging relies on utilizing many simple, cheap robots rather than fewer more capable and more expensive machines. Cheap robots often have less robust sensing mechanisms and reduced mobility which could hamper their ability to successfully detect and avoid threats posed by dangerous environments. Furthermore, robots used in foraging research are often physically smaller and unable to retrieve larger target objects (such as a pet).

B. Environment Mapping and Path Planning

The canonical task of sensing, mapping, localization, and planning (SLAM) within an unknown environment is a well studied problem (see the textbook from Thurn, Burgard, and Dieter [8] for an overview of SLAM algorithms). Therefore, we constrained our examination of previous work on this topic to studies that specifically addressed the issue of harsh or potentially dangerous static environments.

As part of a robot designed to assist human rescue teams following natural disasters, Birk et al [9] implemented a navigational system fusing occupancy grid data from a range finding sensor with readings from dead reckoning encoders and the orientation provided by an onboard compass. This combination proved sufficient for the short-duration trials they ran. Baker et al. [10] developed a robot capable of exploring the hazardous environment of subterranean mines, using a 3-D laser sensor to obtain a discretized terrain map and projected A* algorithm-based routes according to the traversability of the map determined by obstacle distributions. Ulivi et al [11] describe an obstacle-avoiding motion planning system that uses fuzzy logic-based perception and stochastic grid mapping

(different from binary; empty or full) in conjunction with an A* algorithm for the navigation.

C. Kinematics of Pushing

The subproblem of pushing a target object introduces additional constraints on robot mobility beyond hardware limitations - in particular, the robot must remain behind the target object along that entire path back to the safe zone. To accomplish this our pushing task can be divided into several subtasks as follows. 1) Find the target and move towards it. 2) Position robot behind the target and facing the goal. 3) Push the target object to the goal area. During this process, the robot may lose control or even lose sight of its target. If this happens, it will then go back to the first step and repeat this procedure.

According to Balch and Emery [12], this procedure can be implemented by the finite state machine controlling of robot with a sequence of behaviors. The behavior assemblages are formed by several motor schemas such as scan, go-to-target, dock, push...The controller calculates an instantaneous heading and speed based on sensor readings, so it is quick and precise. Complementing this work, Mason and Lynch **??** examine the stability of pushed objects (in particular, the problem of keeping pushed objects in fixed contact against a manipulator) and describe a planner enabling robots to find obstacle-avoiding stable paths. Both these works can be seen as building blocks to the target-pushing subtask present in our study.

III. Problem Statement

Our project's goal is to develop and integrate the technological building blocks required to achieve this objective in a simplified scenario. Specifically, we aim to build a control system for a skid steer robot that will enable it to, starting from an unknown location in an unknown environment, locate the target object and safe area and push the target object into the safe area while avoiding obstacles. We make the following assumptions in our scenario: (1) We have a static environment (all obstacles are in a fixed position for the duration of the experiment). (2) Only a single robot is available for the task (we do not address a multirobot find-and-retrieve). (3) The target object and safe zones are identifiable with AR tags. (4) The ground surface is flat, and any terrain not occupied by an obstacle is navigable by our robot. (5) All obstacles are at least 10cm tall and are detectable by the robot's lidar sensor.

IV. Approach and Implementation

A. Hardware

Our system uses the a skid steer robot, namely the Husarion ROSbot 2.0 [13]. This robot has four independently controlled wheels with encoders for odometry, a lidar sensor, a stereo RGBD camera, four distance sensors, and an IMU. This robot includes wifi connectivity, but we elected to use a wired ethernet connection for increased reliability and bandwidth. The robot's operating system was ROS Kinetic [14] running on Ubuntu 16.04 [15]. We visualized the robots behavior using RViz [16], and controlled the robot using a command-line interface from Visual Studio Code with the Remote-SSH extension [17], [18].

(a) Husarion ROSbot 2.0 with Bumper Attached

(b) AR Cubes, used as the treasure object and goal zone

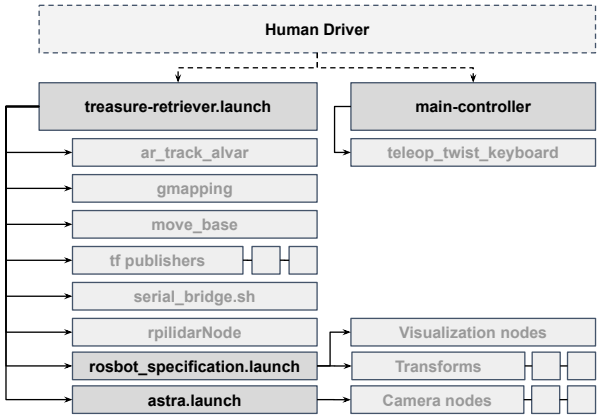
Fig. 1: Hardware Components

To represent the target object and the goal zone, we generated AR tags with specific size and ID and attached them on two cubes as treasure object and goal zone separately as shown in Fig. 1b. Each face of cube has an AR tag with the same size and a unique ID. This configuration allows us to identify the orientation and position of each cube if any of the sides are in view of the robot's camera.

It is unstable and difficult to push the target object directly by the robot, so we designed a bumper to help pushing. We first made a bumper that is box like and has two layers so it could be strong enough to hold and push the object. However, this bumper is too thick that the distance sensor can detect it as an obstacle. Then we improved our design by changing it from two layers to one layer and lengthening the arm on both sides as shown in Fig. 1a. The new bumper isn't as strong as the first one but with a larger opening angle, it is easier to catch the target object which can tolerates more orientation error of the robot. The distance sensors won't be influenced as well. With this bumper, we can catch object within 60 degree range and always keep it inside the bumper unless robot continues rotating for a long time.

B. System Architecture

Our system is implemented using a main controller in conjunction with third party nodes. The most important third party nodes are gmapping [19], which performs simultaneous localization and mapping; `ar_track_alvar` [20], which recognizes AR Tags (fiduciary markers) affixed to retrieval object and goal zone targets; and `move_base` [21], which performs path planning and navigational control. The system is launched in two parts (Fig. 2a): first, the `treasure-retriever.launch` file is used, which launches the transformation publishers, hardware drivers, mapping node, navigation node, and AR tag recognition node. This activates all sensors on the robot, sets up the `tf` hierarchy, and launches support processes. Second, the `main-controller` node is launched individually. This node acts as a central



(a) Launch System.

(b) Finite State Machine of the main controller. Light gray boxes indicate life cycles of import third-party ROS nodes.

Fig. 2: System Architecture.

manager of the other nodes, and is implemented using a finite state machine (FSM) with five states as shown in Fig. 2b.

In order to give the human driver control over the robot's movements during the `SEARCH_OBJECTIVES` phase, the controller launches the `teleop_twist_keyboard` node, which interprets key presses and publishes `Twist` commands to the serial bridge over the `cmd_vel` topic. During this phase, the human driver can see whether the robot has located the treasure and goal zone and can press `ctrl-c` to advance the controller to the next state, `FETCH_TREASURE`.

The controller stops execution of the `teleop_twist_keyboard` node upon entering `FETCH_TREASURE` and takes over control of the robot. Using actionlib [22], the controller issues the location of the treasure cube as a navigation goal to `move_base`, which both plans the route to the treasure and issues the commands to get there. Once the action is complete, the controller advances to `DELIVER_TREASURE` and issues a second navigation goal to `move_base`, this time with the goal zone as the navigation goal. Once this second action is complete, the controller advances to `DONE` and issues no further commands to the robot.

C. SLAM

One of the most important factors for achieving the goal of this paper is making a robot properly get to the position of either a treasure or a goal zone. This entire navigation process requires localization of the robot, i.e., where the robot is positioned in the environment, as well as building a map based on the sensing of the environment. However, in this paper, given that the Husarion ROSbot 2.0 is deployed in

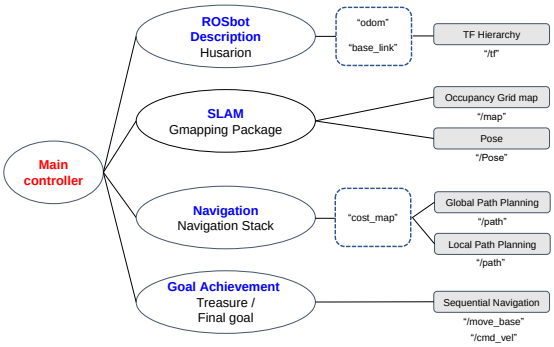


Fig. 3: Node map of SLAM and navigation.

TABLE I: gmapping parameters

Parameter	Value
xmin $\sim$ xmax range [m]	-5 $\sim$ 5
ymin $\sim$ ymax range [m]	-5 $\sim$ 5
particles [ea]	120
resolution [m]	0.1
map update interval [sec]	1

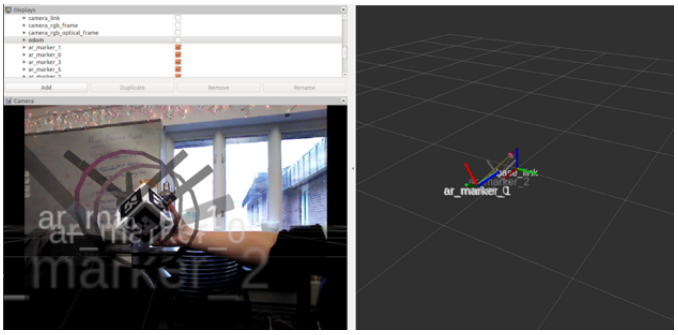
an entirely unknown environment, we chose the methodology based on SLAM (Simultaneous Localization and Mapping). In other words, using the robot's sensor data and localization, the robot gradually builds a map without saving or loading the map.

Considering available slam packages [23]–[25], we chose to use the built-in and OpenSLAM *gmapping* package [19]. The gmapping package utilizes Rao-Blackwellized particle filter [26] to localize the robot and eventually establishes the mapping as follows: sampling, importance weighting, resampling, and map estimating. As the Husarion ROSbot is made as a differential drive robot with four wheels, we are able to harness the power of the ROS built-in gmapping package by meeting the hardware requirement including the odometry and LiDAR sensor.

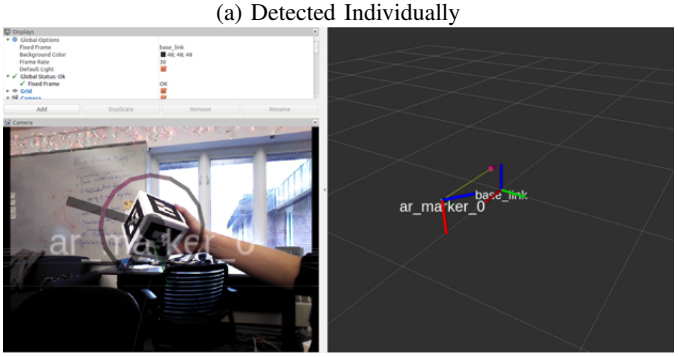
As shown in Fig. 3, the gmapping package publishes `/map` and `/pose` topic which contains the occupancy grid map data and the robot's pose data, respectively. Each sub part illustrates the relevant published topic by that node. When the operator maneuvers the robot by teleop key, this `gmapping` requires the connection between the odometry, `base_link`, and the sensor by tf hierarchy. Thus, we used ROSbot specification consisting of URDF and Xacro format files in order to conduct the transformation based on the odom [27]. The technical details of gmapping we principally used here are as shown in Table I.

D. Object and Goal Zone Recognition

To recognize treasure object and goal zone, we need to identify them with specific patterns that distinguish them from other surrounding objects. Ideally, when we recognize the object, we can get the relative pose between robot and object. We used the `ar_track_alvar` ROS package which is based on Alvar, an open source AR tag tracking library. The main



(a) Detected Individually



(b) Detected as a Bundle

Fig. 4: Defining bundles of AR tags improves treasure cube and goal-zone localization.

function of this package is generating AT tags, identifying and tracking the pose of AR tags. Instead of recognizing multiple AR tags individually, we track the six AR tags in the same cube as a bundle. In this way, after inputting the spatial relationship information of six tags, we can directly get the pose of the cube to the camera from images. Then as shown in Fig. 4b, with the predefined coordinate frame relationship in the robot, the relative pose of cubes to the robot can be calculated once the camera detects the tags.

With this method, the distance to the tags can be calculated because we know the size of the tags and can detect four corners of the tag, so we only need RGB image without depth information. The black and white feature of AR tags can also realize more stable detection than colorful feature in environment with varied illumination.

E. Path Planning and Execution

Based on the SLAM in subsection *C*, the main task to achieve the mission is that the robot should plan a path from the starting position to the destination. During this task, the robot is required to avoid collisions with obstacles and safely navigate using a cost-effective method. Since the built map is based on the occupancy grid map format, we are able to use a `move_base` path planning method that can determine the sequence of poses on the grids. The `move_base` is a node from the `move_base` package and it comprises the core component of the high-level ROS navigation stack.

The `move_base` node enables Husrarion ROSbot's navigation in terms of two ways: global path planning and local path planning. The global path planning stands for the navigation plan as the shortest path to the goal zone typically using the saved map by `map_server`. However, in this study, we replaced the saved map with subscription of the map by SLAM gmapping package once the mapping is finished after the robot finds the two ar tag markers; thus, the ROS operating complexity was significantly reduced as discussed in Section V. As a default, we adhere to `nav_core`'s BasePlanner as of `navfn` using *Dijkstra* algorithm for global planning [28]. Moreover, the path is plotted not by binary occupancy grid, that is, empty or occupied, but by global cost map. This cost map allows the robot to avoid the obstacles on the built map in a safer way.

On the other hand, while the robot is following the global path, the local planner is activated in relation to the global path and unexpected obstacles. In this study, we used *Dynamic Window Approach* [29], [30] as a default local planner [28]. As a result, the robot can determine the sequence of the motion by maximizing the objective function in the corresponding circumstances as

$$G(v,\omega) = \sigma(\alpha \cdot textrm head(v,\omega) + \beta \cdot \text{dist}(v,\omega) + \gamma \cdot vel(v,\omega)) \quad (1)$$

where G is the objective function. v is transitional velocity, ω is angular velocity, σ is smoothing operator, $head$ is approximation to goal angle, $dist$ is the minimum distance to obstacles, vel is the linear velocity value in the velocity set, and α, β, and γ are the weights for the functions.

In addition, the interface between the global and local planning makes it possible for the robot to adaptively navigate to the goal. Hence, even if the global planning was not sufficient right after the detection of targets, the local planner appropriately adjusts the course within a small scale by eventually publishing the `cmd_vel` topic to arrive at the treasure and goal point in a sequence.

The `move_base` consisting of the above concept was configured by ROS action library. We implemented the action client based on MoveBaseAction; thus, the two destinations, i.e., the treasure and the goal zone, were inputted as a MoveBaseGoal. Also, we set up the necessary parameters for elements such as global and local cost maps, distance threshold and so on in a separate config folder. In consequence, instead of a typical *topic* communication, the *action* communication allows the interfacing by ensuring a continuous and interactive way.

V. Experiment Result

A. Static Maps

Our current approach uses gmapping [19] to perform simultaneous localization and mapping upon start up. However, one disadvantage we encountered with gmapping is that the map has a tendency to become corrupt in scenarios when the reckoning from odometry system is inaccurate such as if a wheel is caught in a cable. In these scenarios, odometry may report a rotation that is substantially different from the physical

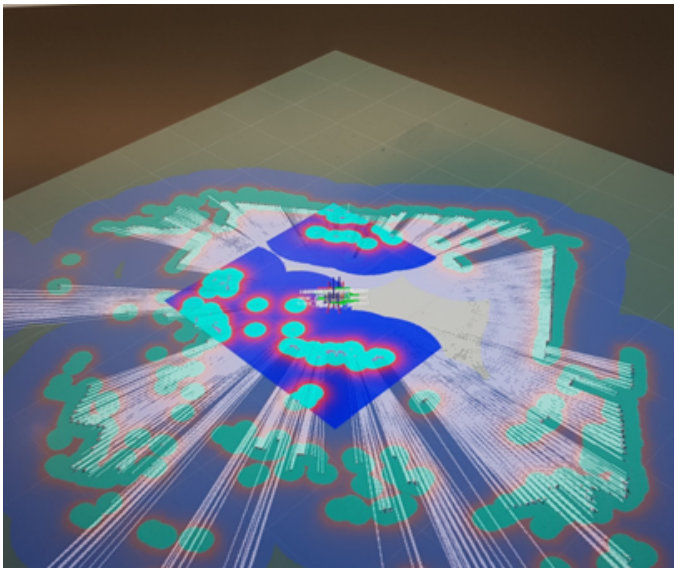
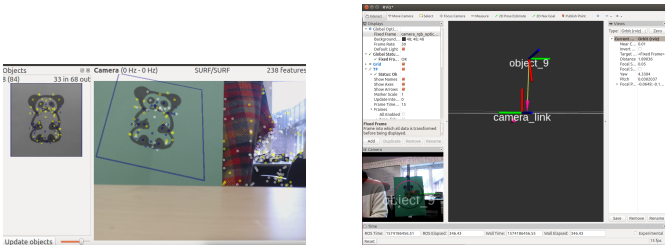
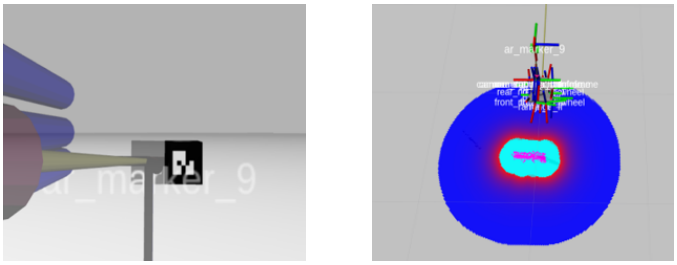


Fig. 5: Cost map on rviz where the small rectangle illustrates the local cost map and large map illustrates the global cost map.



(a) GUI of find_object_2d (b) Shown found object in rviz

Fig. 6: Test using `find_object_2d` were able to successfully identify a target pattern, but incurred an unacceptable performance penalty due to the processing burden of 3D depth sensing.



(a) Marker spawned (Gazebo) and detected by camera (b) Marker shown in cost map on rviz

Fig. 7: Simulation with spawning AR marker and detection under ROS rviz and Gazebo.

reality which in turn results in a error gmapping's inferred localization and introduces artifacts onto the map.

One approach we experimented with to solve this was the use of static environmental maps [31] and a probabilistic localization system [32]. We were able to implement this using the same techniques we used for controlling the `teleop_twist_keyboard` node - namely, the main controller would launch a `map_saver` node upon completion of the `SEARCH_OBJECTIVES` state, then `map_server` to serve the static map and `amcl` to provide localization (`gmapping` would be stopped at this point). Because our system is a model for a robots that can retrieve objects from unknown dangerous environments, we ultimately decided that a dedicated map-building step and the subsequent delay while saving, reloading, and relocalizing in that map would be detrimental to our system's goal of fast retrieval.

B. Pattern Matching

We also tried to use pattern matching for the goal zone as in the real world, not everything can have AR tags on it. It is more common to use the specific pattern of each object to trace its position and orientation. We tried pattern matching by using the package `find_object_2d`. This package is very powerful. It only needs us to input a picture of the object then it will automatically find the feature points of that image and do the pattern match. This package has a good performance with only RGB camera. During a test, it was able to find the matched pattern very precise and fast (see Fig. 6). The problem is when we need it to return the position and the orientation of the found object, it takes around 3 seconds every time to update its message. The reason causes this low efficiency is that it uses the depth registered camera to calculate the position of that matched pattern. We couldn't find a good method to solve this problem so we abandoned it and used AR tags only for recognition.

C. Simulation

In order to overcome physical limitations and conduct a pre-test before an on-scene test, we customized the visualization and simulation environment in ROS rviz and Gazebo. Modular approaches such as AR tag detection, SLAM, and navigation are validated in the virtual environment so that the finalized methodology can be implemented on the real robot. We built cube models and attached AR tags on them with Solidworks, but some of the texture information was lost when transferring them into URDF format for use in Gazeb, causing some faces of the cubes to disappear when viewed from certain perspectives. This diminished the reliability of AR Tag detection in simulation and we were not able to fix this by paper submission time.

As for SLAM and navigation, the Husarion ROSbot 2.0 was spawned on the Gazebo by rosbot specification. However, there were limitations of spawning AR tags in the virtual environment, which were made on Blender 3-D modeling. Even if the actual size of the AR tag cubes was 10 cm for each side, the spawned ROSbot on the Gazebo could not detect the marker. As a result, we had to increase each side up to 20 cm so that the cube can be recognized by the robot's camera in the simulation. For this reason, the cube was detected by the laser scanner as an obstacle shown on the cost map as

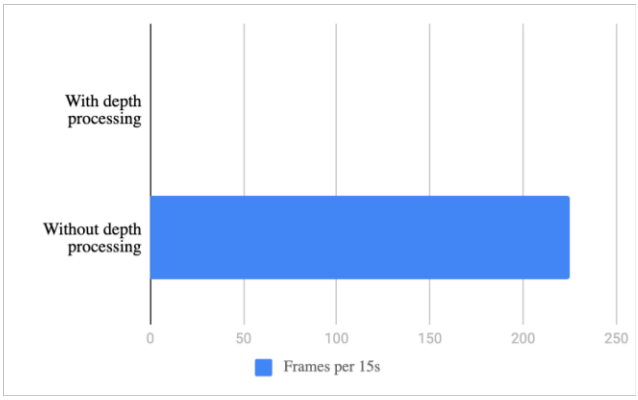


Fig. 8: Disabling depth processing dramatically improved throughput of camera frame messages.

shown in Fig. 7 (a). Additionally, even though the marker was detected, the pose of the marker shown by `tf` was represented abnormally as shown in Fig. 7 (b). This 3-D modeling and detection test with AR tags in the simulation environment should be further studied.

D. Performance Optimization

Initial tests exhibited poor performance with both AR Tag recognition and path following. The robot often would take a long time (over 15 seconds in casual observations) to recognize AR tags that were directly in front of the camera. Suspecting that this could be caused by a low camera frame rate, we measured the frequency of camera messages using `rostopic hz /camera/image_raw`, and observed that no new messages were published after 15 seconds. As mentioned in Section V-B, we suspected that this delay could be attributable to the processing cost incurred by the depth processing feature of the `astra_camera` package [33]. Disabling this processing resulted in a dramatic improvement in frame rate, as shown in Fig. 8.

To address the second deficit we observed (poor path following), we made modifications to the default `move_base` tuning parameters specified by the `rosbot_description` module [27]. Adjustments are shown in Table II. The adjustments to `min_vel_x`, `max_vel_theta`, and `min_vel_theta` cause the robot to turn less sharply, making it more likely that the treasure cube will remain on the bumper of the robot. The adjustment to `pdist_scale` makes the path following care less about following the planned path closely as opposed to reaching the obstacle, which reduces oscillation while following the path. The adjustment to `yaw_goal_tolerance` effectively removes the requirement that the robot match a target orientation when arriving at a destination pose, therefore removing unnecessary in-place rotations.

TABLE II: Parameter Tuning

Parameter	Default	Tuned
`min_vel_x`	`-0.1`	`0.05`
`max_vel_theta`	`2.0`	`1.0`
`min_vel_theta`	`-2.0`	`-1.0`
`pdist_scale`	`0.4`	`0.3`
`yaw_goal_tolerance`	`0.1`	`3.14`

VI. Conclusion

A. Discussion

In conclusion, we built a autonomous system with Husarion ROSbot 2.0 that can do mapping in an unknown environment, detect the AR tags and localize their position, then make path planning to push target object to the goal zone. We also did navigation, simulation and pre-test based on each module with RVIZ and Gazebo before integrating this system. Our system was able to successfully execute the mission stimulated in the problem statement (see Fig. 9), although opportunities to improve exist, as we discuss next.

B. Future Work

Our system serves as a proof-of-concept solution given our problem statement, but there remains substantial future work that could be done to make this solution more robust. Keeping in place the assumptions mentioned in Section III, our solution could be improved with control adjustments to better keep the treasure cube in the bumper zone. Our system, which did not contain software provisions for ensuring the cube remaining in the bumper, sometimes failed to keep the cube in place.

There are several assumptions that our system made that should be relaxed in order to achieve the vision mentioned in Section I. First, our system for detecting the target object and goal zone should be generalized to objects not affixed with AR Tags (as real world objects are unlikely to have them). The methods for identifying these two problem features would depend on the context - as an example, for space mining, the treasure object could be identified based off spectral analysis of nearby asteroids. Second, our system should be strengthened to be able to account for dynamic obstacles, as real world scenarios like disaster relief would likely not have an entirely static environment. Third, we propose that a multi-robot solution (as in the foraging problems mentioned in Section II-A) could be more effective in quickly retrieving valued objects than a single robot.

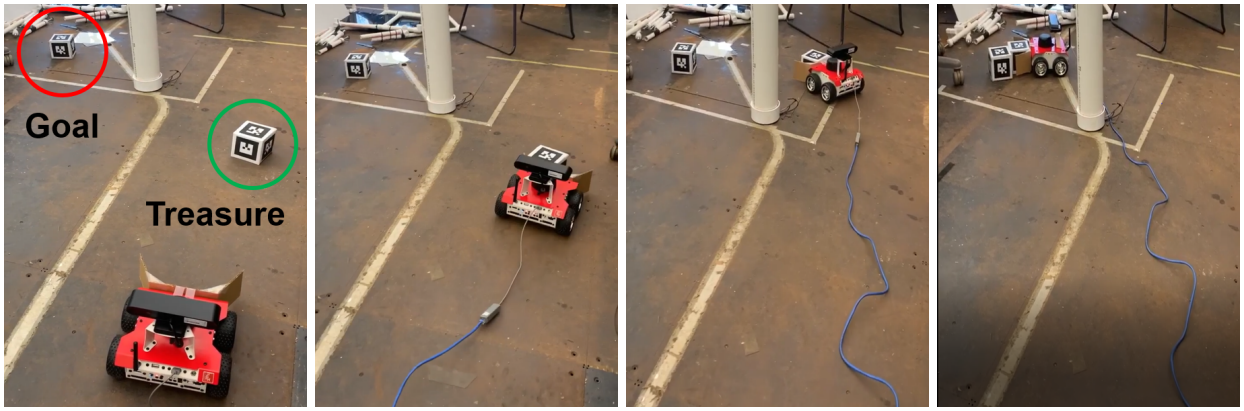
Acknowledgements

We would like to thank Dr. Alberto Quatrinni Li, our professor, for his excellent teaching and support of our project, as well as Amazon Inc. for the donation of Husarion robots to the class.

Appendix

The following content is available for viewing (on an electronic copy, click to view):

- Video from an rviz view of the robot completing a retrieval task



(a) (b) (c) (d)

Fig. 9: Demonstration of treasure retrieval movement under environment with obstacles (a) The robot is located at the starting position and detected the treasure and the goal (b) The robot moves to the treasure (c) The robot avoids obstacle based on the mapping and path planning (d) The robot safely arrived at the goal by delivering the treasure.

REFERENCES

[1] N. Ruangpayoongsak, H. Roth, and J. Chudoba, "Mobile robots for search and rescue," in *IEEE International Safety, Security and Rescue Rototics, Workshop, 2005.*, Jun. 2005, pp. 212–217. DOI: 10.1109/SSRR.2005.1501265.

[2] R. Stopforth, S. Holtzhausen, G. Bright, N. S. Tlale, and C. M. Kumile, "Robots for search and rescue purposes in urban and underwater environments - a survey and comparison," in *2008 15th International Conference on Mechatronics and Machine Vision in Practice*, Dec. 2008, pp. 476–480. DOI: 10.1109/MMVIP.2008.4749579.

[3] C. Lundberg, H. I. Christensen, and A. Hedstrom, "The use of robots in harsh and unstructured field applications," in *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, Aug. 2005, pp. 143–150. DOI: 10.1109/ROMAN.2005.1513771.

[4] C. Wong, E. Yang, X. Yan, and D. Gu, "An overview of robotics and autonomous systems for harsh environments," in *2017 23rd International Conference on Automation and Computing (ICAC)*, Sep. 2017, pp. 1–6. DOI: 10.23919/IConAC.2017.8082020.

[5] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng, "Cooperative Mobile Robotics: Antecedents and Directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, 1997, ISSN: 09295593. DOI: 10.1023/A:1008855018923.

[6] P. E. Rybski, A. Larson, H. Veeraraghavan, M. Anderson, and M. Gini, "Performance evaluation of a multi-robot search & retrieval system: Experiences with MinDART," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 52, no. 3-4, pp. 363–387, 2008, ISSN: 09210296. DOI: 10.1007/s10846-008-9222-9.

[7] T. H. Labella, M. Dorigo, and J. L. Deneubourg, "Efficiency and task allocation in prey retrieval," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3141, pp. 274–289, 2004, ISSN: 03029743. DOI: 10.1007/978-3-540-27835-1_21.

[8] S. Thrun, W. Burgard, and F. Dieter, "Probabilistic Robotics," 2005.

[9] S. Carpin, H. Kenn, and B. Andreas, "Autonomous mapping in the real robot rescue league," *RoboCup 2003: Robot Soccer World Cup VII*, 2003. [Online]. Available: http://www.isd.mel.nist.gov/projects/USAR/IUB%20Awardee%20Paper.pdf.

[10] C. Baker, A. Morris, D. Ferguson, S. Thayer, C. Whittaker, Z. Omohundro, C. Reverte, W. Whittaker, D. Hähnel, and S. Thrun, "A campaign in autonomous mine mapping," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2004, no. 2, pp. 2004–2009, 2004, ISSN: 10504729. DOI: 10.1109/robot.2004.1308118.

[11] G. Oriolo, M. Vendittelli, and G. Ulivi, "On-Line Map Building and Navigation for Autonomous Mobile Robots," pp. 2900–2906,

[12] R. Emery and T. Balch, "Behavior-based control of a non-holonomic robot in pushing tasks," *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 3, pp. 2381–2388, 2001, ISSN: 10504729. DOI: 10.1109/robot.2001.932978.

[13] H. sp. z o.o., *Husarion rosbot 2.0 website*, https://store.husarion.com/collections/dev-kits/products/rosbot, version 2019, Online; accessed 21 November 2019, 2019.

[14] O. S. R. Foundation, *Ros kinetic kame*, http://wiki.ros.org/kinetic, version 2019, Online; accessed 21 November 2019, 2019.

[15] C. Ltd., *Ubuntu 16.04*, http://releases.ubuntu.com/16.04/, version 2018, Online; accessed 21 November 2019, 2018.

[16] O. S. R. Foundation, *Rviz visualization tool*, http://wiki.ros.org/rviz, version 2019, Online; accessed 21 November 2019, 2019.

[17] M. Corporation, *Visual studio code remote ssh extension*, https://marketplace.visualstudio.com/items?itemName=ms-vscode-remote.remote-ssh, version 2019, Online; accessed 21 November 2019, 2019.

[18] ——, *Visual Studio Code*, 2019. [Online]. Available: https://marketplace.visualstudio.com/ (visited on 11/21/2019).

[19] O. S. R. Foundation, *Gmapping ros package*, http://wiki.ros.org/gmapping, version 2019, Online; accessed 21 November 2019, 2019.

[20] ——, *Ar track alvar ros package*, http://wiki.ros.org/ar_track_alvar, version 2019, Online; accessed 21 November 2019, 2019.

[21] ——, *Move base ros package*, http://wiki.ros.org/move_base, version 2019, Online; accessed 21 November 2019, 2019.

[22] ——, *Actionlib ros package*, http://wiki.ros.org/actionlib, version 2019, Online; accessed 21 November 2019, 2019.

[23] Google, *Cartographer ROS*, 2018. [Online]. Available: https://google-cartographer-ros.readthedocs.io/en/latest/ (visited on 11/21/2019).

[24] S. International, *Karto*. [Online]. Available: http://wiki.ros.org/karto (visited on 11/21/2019).

[25] S. Kohlbrecher and J. Meyer, *ROS hector slam*, 2014. [Online]. Available: http://wiki.ros.org/hector%7B%5C_%7Dslam (visited on 11/21/2019).

[26] S. Särkkä, A. Vehtari, and J. Lampinen, "Rao-blackwellized particle filter for multiple target tracking," *Information Fusion*, vol. 8, no. 1, pp. 2–15, 2007, Special Issue on the Seventh International Conference on Information Fusion-Part II, ISSN: 1566-2535. DOI: https://doi.org/10.1016/j.inffus.2005.09.009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1566253505000874.

[27] H. sp. z o.o., *Rosbost description ros package*, https://github.com/husarion/rosbot_description, version 2019, Online; accessed 22 November 2019, 2019.

[28] K. Zheng, *Ros navigation tuning guide*, http://kaiyuzheng.me/documents/papers/ros$_n avguide.pdf$, Sep. 2016.

[29] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics Automation Magazine*, vol. 4, no. 1, pp. 23–33, Mar. 1997, ISSN: 1558-223X. DOI: 10.1109/100.580977.

[30] A. Özdemir and V. Sezer, "Follow the gap with dynamic window approach," *International Journal of Semantic Computing*, vol. 12, no. 01, pp. 43–57, 2018. DOI: 10.1142/S1793351X18400032. eprint: https://doi.org/10.1142/S1793351X18400032. [Online]. Available: https://doi.org/10.1142/S1793351X18400032.

[31] O. S. R. Foundation, *Map server ros package*, http://wiki.ros.org/map_server, version 2019, Online; accessed 21 November 2019, 2019.

[32] ——, *Amcl ros package*, http://wiki.ros.org/amcl, version 2019, Online; accessed 21 November 2019, 2019.

[33] ——, *Astra camera ros package*, http://wiki.ros.org/astra_camera, version 2019, Online; accessed 22 November 2019, 2019.