Search And Rescue Autonomous System (SARAS)

Andrew Scheffer
drewskis@umich.edu

Joshua Symonds
jsymonds@umich.edu

Kathryn Wakevainen
kwakev@umich.edu

Tianle Wu
perper@umich.edu

***Abstract*—Search And Rescue Autonomous System (a.k.a SARA), refers to a highly integrated, fully automated, and functionally precise robotics operational system which aims to guide a malfunctioning robot back to its destination. The goal of the SARA is to autonomously locate a stranded "Blind" robot, which is unable to read its LiDAR data, within an unknown environment and return it to a home base. This rescue is performed autonomously by a "Seeker" bot. The SARA system is designed to be deployed in most of the complicated landscapes on Earth. Space exploration and rescue missions are also possible.**

I. Keywords

Map: The originally unknown environment that both the Blind bot and Seeker bot are placed within. For the purposes of this report, this environment is an 4x3 or 3x3 grid of 62cm wall segments that enclose the environment and act as obstacles within it.

Home: A safe location in the map, defined as the starting location of the Seeker bot.

Blind bot: A robot in an unknown environment without access to its sensor data and therefore cannot safely navigate the environment back to home alone. The Blind bot is still able to send and receive information.

Seeker bot: The robot with its sensors fully functional that must explore the environment, locate the Blind bot, and navigate itself and the Blind bot back Home.

Potential Lost Area (PLA): Areas where the Blind bot may be after the Seeker bot loses sight on it during the RETURNING HOME TOGETHER state.

II. Introduction

Often in robotics, robots are used to perform tasks where it would be unsafe for a human to be in the environment. Common examples of environments unsafe for humans where robots are used in our stead include deep space exploration, deep sea exploration, areas with radioactive material, environments with inhospitable temperatures, etc. It's common for the environment these robots are in to not be known prior to their deployment so its crucial for them to be able to operate within an unknown environment. However, in the event of a sensor malfunction, the robot may no longer be capable of learning about its environment or localizing itself within the environment. Due to this, if a robot encounters a sensor malfunction while in these environments it can no longer safely navigate its environment and is essentially stranded since humans may not be able to retrieve the robot manually and it cannot navigate to a safe location on its own. Sending a second robot into the environment with fully functioning sensors to help the malfunctioning robot navigate allows for us to rescue it without compromising human safety. The Search And Rescue Autonomous (SARA) System is able to be deployed in an unknown environment where a robot with sensor malfunction is stranded; it would then explore the unknown environment, find the robot with sensor malfunctions, and navigate both itself and the malfunctioning robot back to some predetermined location safely.

III. Related Work

A. Swarm Robotics

A swarm refers to a robotic system which controls multiple physical robots. Although currently there is no standard implementation libraries for swarms, we were inspired by Swarm Robotic Behaviors from Dr. Schranz et al. Their description of swarms consists of multiple homogeneously or heterogeneously interconnected robotics. Since individual robots have processing, communication and sensing capabilities locally onboard, they are able to interact with each other, and react to the environment autonomously. [3] Although the controlling concept is similar, there are still several fundamental differences between SARA system and swarms. First, in swarm, each robot's local behaviors incorporate interactions with the physical world, including the environment and other robots.[3] The Blind-bot in SARA is malfunctioned, which indicates the possible loss of sensor and vision. Its designed to only be able to communicate with the Seeker bot. Other environmental factors would be handled by the Seeker bot. Second, a majority of design principles for swarms inspired by biological features, such as self-healing and self-reproducing.[3] In SARA, retrieving and rescuing are the only mission focused by the Seeker. As for the Blind bot, it simply follows the command and moving to the waypoints directed by he Seeker bot.

IV. Methodology

The SARA system requires for the Seeker bot beginning by exploring the unknown environment using exploitative Simultaneous Localization And Mapping (SLAM) to navigate the environment. Once the Seeker bot has found the Blind bot - defined by the camera on board the Seeker bot detecting the AprilTag on board the Blind bot - the Seeker bot is to compute safe paths for both itself and the Blind bot to navigate back to its starting location without collision. The system is successful when the Seeker bot is able to return to its starting position and navigated the Blind bot to a safe location within 20cm of the starting location. Fig. 1 is a picture of the Blind bot for reference.

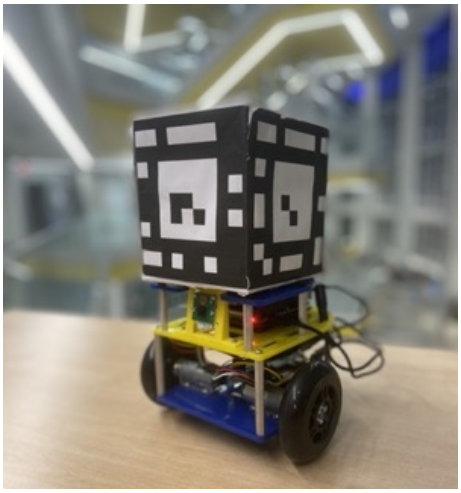

Fig. 1. Blind bot with fiducial markers attached

A. Architecture

This project involved the interaction of multiple complicated components. Fig. 2 showcases specific system components of the functional SARA system (including both the Seeker bot and the Blind bot).

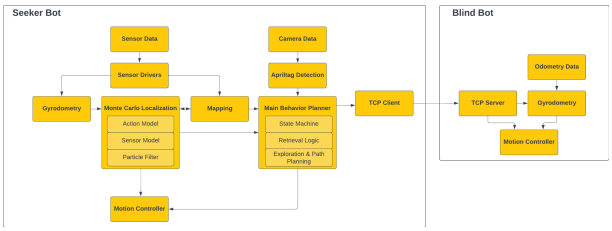

Fig. 2. System components of the Seeker bot and Blind bot

As shown, the architecture of the Seeker bot is necessarily much more complicated than the architecture of the Blind bot. This is because the Seeker bot is not only responsible for maintaining it's own position in the map using SLAM, but is also responsible for updating the position of the Blind bot using our pose-estimation pipeline. The Seeker bot also contains a main-behavior controller which effectively interfaces all of these independent subsystems in a way that can accomplish the mission objective of sending towing the Blind bot back to a safe location.

The Blind bot architecture is very simple. Our team assumes that the Blind bot still has access to reasonably accurate odometry data and can follow relative waypoints based on its odometry data. The way our system is structured (using the towing method) allows for some error in the Blind bot's motion controller. This is because updates to the Blind bots position in the configuration space are made relatively often. As shown, the only communication interface between the Seeker bot and Blind bot is a single TCP socket. Using a defined user protocol, the Seeker bot can efficiently update the Blind bot about its position in the world frame and send customized global waypoints to the Blind bot's motion controller.

B. Ideal Implementation State Machine

To facilitate the complexity of our project, we updated the state machine to include extra states that encode the behavior, shown in Fig. 3. The states INITIALIZING, EXPLORING MAP, RETURNING HOME ALONE (RETURNING HOME), FAILED RESCUE (COMPLETED EXPLORATION), and CRITICAL FAILURE (FAILED EXPLORATION) encode the behavior of original Botlab where the MBot explores an unknown environment using SLAM and returns to its starting position upon successful exploration of the entire environment. The states RETURNING HOME TOGETHER, RETRIEVING, and SUCCESSFUL RESCUE allow for the SARA system to rescue the Blind bot and navigate it back to home. INITIALIZING is the starting state, and CRITICAL FAILURE, FAILED RESCUE, and SUCCESSFUL RESCUE are the terminal states. If any nonterminal and noninitializing state encounter a critical error such as an obstacle collision or if attempts to navigate to an unreachable location, it transitions into the CRITICAL FAILURE STATE which is shown in the state machine as the dashed red arrows. A successful trial of the SARA system will terminate in the SUCCESSFUL RESCUE state.

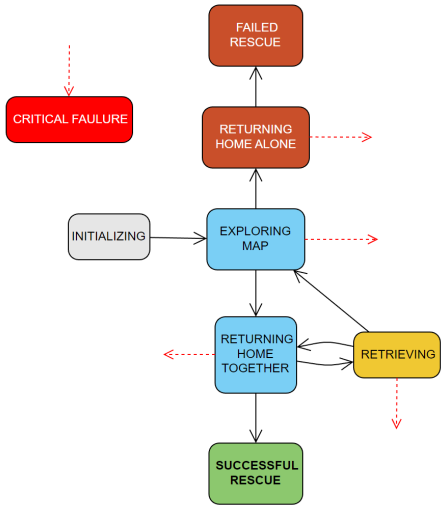

Fig. 3. State machine for SARA System

Since INITIALIZING, EXPLORING MAP, RETURNING HOME ALONE, FAILED RESCUE, and CRITICAL FAILURE were covered within the scope of Botlab we will not discuss the behavior of these states in extensive detail unless an update has been made to the state.

*1) **EXPLORING MAP**:* Beyond the original implementation of EXPLORING MAP, when in this state we now also constantly check if the Seeker bot sees the Blind bot. If the Seeker Bot ever sees the Blind bot by identifying the AprilTag on top of the Blind bot, it transitions to the RETURNING HOME TOGETHER state. Otherwise if the Seeker Bot explores the entire map without seeing the Blind bot, it transitions into the RETURNING HOME ALONE state.

To make it so that the Seeker bot is more likely to see the Blind bot while exploring the map, it periodically does a full 360° rotation to observe its entire immediate environment with the camera.

*2) **RETURNING HOME TOGETHER**:* RETURNING HOME TOGETHER is the state that encodes behavior related to navigating both the Blind bot and Seeker bot back towards Home. When in this state, the Seeker bot starts by planning a path back to Home using the map its created using SLAM while exploring and A* path planning in the exact same way as is done in RETURNING HOME ALONE in the scope of original Botlab. The Seeker bot then begins driving backwards along this path back home and periodically sends its location to the Blind bot. During this, the Blind bot is following the Seeker bot; it does this by travelling along the waypoints published by the Seeker bot. If the Seeker bot returns Home without losing the Blind bot, it has successfully navigated itself and the Blind bot back to Home and then we transition to the SUCCESSFUL RESCUE state. If the Seeker bot ever loses sight of the Blind bot for a sufficiently long time while driving towards Home, it abandons its path back towards home and transitions into the RETRIEVING state to find the Blind bot again. Since if the Seeker bot loses sight of the Blind bot while driving Home causes for us not to successfully complete the rescue from this instance of RETURNING HOME TOGETHER, it's important for the Blind bot to be highly visible to the Seeker bot while in this state. The Seeker bot drives backwards in this state so that the Blind bot is maximally visible while it follows behind.

*3) **RETRIEVING**:* If the Seeker bot loses sight of the Blind bot while navigating back Home in RETURNING HOME TOGETHER, the Seeker bot must begin looking for the Blind bot again. However, simply transitioning back to EXPLORING MAP in order to resume exploration would not work since the Seeker bot has already explored much of the map at this point and this information would help it look for the Blind bot more efficiently. We use the information we already know about the map to help the Seeker bot look for the Blind bot by tracking the potential lost areas (PLA) that the Blind Bot may be in. PLA are areas within the explored map that the Seeker bot has not looked at since it lost sight of the Blind bot.

When the Seeker Bot transitions into the RETRIEVING state, it initializes a PLA map of the areas within the known SLAM map that it generated while in the EXPLORING MAP state. While in the RETRIEVING state, the Seeker bot constantly is checking the area that it can see for the Blind bot. If it does see the Blind bot, it transitions back to the RETURNING HOME TOGETHER state to attempt to navigate itself and the Blind bot back home again. If it does not see the Blind bot, it removes the areas it can see from possible PLA and continues searching. It searches the environment by choosing the best PLA to search and navigates towards the PLA using A* navigation. The best PLA is defined to be the PLA that minimizes
$cost(PLA, Seeker, Blind) = Dist(PLA, Blind)^2 + Dist(PLA, Seeker)$
where PLA is the coordinates of the PLA, Seeker is the coordinates of the Seeker bot, Blind is the coordinates of the Blind bot, and Dist(x,y) is the euclidean distance between two coordinates. If the Seeker bot investigates all remaining PLA and still does not find the Blind bot, it transitions into the EXPLORING MAP state and begins exploring unknown areas of the map again. It does this to accommodate for the possibility that the Blind bot mistakenly navigated into an unknown region of the map.

C. Robot Localization

The specs for four AprilTags that attached to each side of the Blind bot are listed in the below table:

Purpose	Size	Family
Front	5.2cm x 5.2cm	tagStandard41h12
Back	5.2cm x 5.2cm	tagStandard41h12
Left	5.2cm x 5.2cm	tagStandard41h12
Right	5.2cm x 5.2cm	tagStandard41h12

During the process of communication, Seeker bot would send waypoints to and Blind bot. However, since the waypoints were generated in seek bot's frame, we need to calculate the homogeneous transformation matrix which convert the waypoints from seek bot to Blind bot's frame. The general idea of conversion comes with the following formula:

$$T_S^W \; T_B^S \; = T_B^W \tag{1}$$

where
W: world frame
S: Seeker bot frame
B: Blind bot frame

From the equation above, the transformation matrix T_S^W could be easily represented as the SLAM pose of the Seeker bot. To generate the transformation from Seeker bot to Blind bot, several under-processing calibrations are needed to compute the fixed transformation metrics.

*1) **AprilTag detector readings based on pi camera**:* While utilizing existed AprilTag detection library, [1] we implemented our own conversion functions to achieve homogeneous transformation. The camera specification would make a significant different while calibrating. The hardware specs for pi camera module v2 was found on the official raspberrypi website.[2]

*2) **Camera to AprilTag fixed transformation**:* In the physical design structure of MBot, the pi camera was attached to a fixed position (facing front and centered) relative to the Seeker bot. Therefore, during the transformation calculation, this matrix would always be a constant term for us. We developed the formula as below:

$$T_S^W \; T_C^S \; T_t^C = T_t^W \tag{2}$$

where
W: world frame
S: Seeker bot frame
C: camera frame (attached to Seeker)

t: AprilTag frame (attached to still wall)

Similar to above steps, T_S^W refers to the SLAM pose of Seeker bot, T_t^C refers to the calibrated AprilTag detector transformation matrix, and T_t^W is the AprilTag position in the world frame. To limit the possible errors, we attached the AprilTag to a steady wall, which is a still object inside the frame. Then, to eliminate the consideration of rotation, we faced the Seeker bot perpendicularly to the wall (also AprilTag). To ensure the precision, we controlled the Seeker bot to move forward and backward multiple times with random distance, and collect the required data. Then, we took the numerical mean of all the generated results to keep precision and eliminate outliers.
Below is our approximation matrix for Camera to Tag:

$$\begin{bmatrix} 1 & 0 & 0 & 2.04971996 \\ 0 & 1 & 0 & 1.15655385 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

*3) **AprilTag to Blind bot**:* Parallel to the transformation from camera to AprilTag, AprilTag to Blind bot is also fixed, as tags are attached to the head of MBot in four directions. We developed the formula as below:

$$T_S^W \; T_C^S \; T_t^C \; T_B^t = T_B^W \tag{3}$$

where

W: world frame
S: Seeker bot frame
B: Blind bot frame
C: camera frame (attached to Seeker)
t: AprilTag frame (attached to Blind bot)

Similar to the approach in camera to AprilTag, with addition to T_B^W, which refers to the SLAM pose in Blind bot's frame. During calibration, we fixed the Blind bot's position. Then, facing perpendicularly to the Blind bot's front AprilTag, we get the Seeker bot's camera readings. Repeat those steps several times for better accuracy. After the front face, we turned the Blind bot into other three directions and followed the above processes.

D. Robot Communication

In order to implement fast and reliable communication between the two robots, our team had to design our own user-facing protocol build on top of a more standard Transmission Control Protocol (TCP). Our team used a common server-client architecture to facilitate the passage of messages between the robots. As illustrated in Fig. 4, the Blind bot acts as a TCP server that accepts messages of type ROBOT_GLOBAL_POSITION, ROBOT_GLOBAL_WAYPOINT or ROBOT_NULL_TERMINATOR. The first two messages encoded poses (of the form x, y, θ) whereas the ROBOT_NULL_TERMINATOR message type signified that a stream of waypoints should be sent to the Blind bots motion controller. The Seeker bot acted as a client in this framework that created and sent these messages to the Blind bot server.

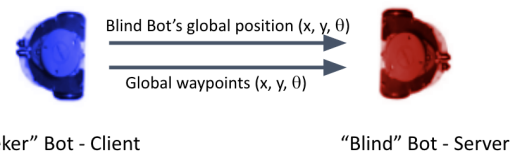

Fig. 4. Server-client TCP communication interface

E. Minimum Viable Product

Unfortunately, we were unable to successfully implement the full scope of our planned project in the time frame we were given. While we were able to fully implement our AprilTag identification and use that data to localize the Blind bot to the full extent of our original scope as well as fully implement inter-robot communication, we were unable to fully implement our high level logic for the RETURNING HOME TOGETHER and RETRIEVING states of our ideal implementation of the state machine.

1) RETURNING HOME TOGETHER: Instead of the original scope of this state's behavior, we implemented RETURNING HOME TOGETHER using an iterative approach to navigate both bots back home. Once transitioned into this state we begin the following loop:

1: Seeker bot saves its current location, C,
if it does not have one saved
2: Seeker bot plans a path back towards home
and takes 1 step along this path
3: Wait until Seeker bot has finished moving
4: If the Seeker bot is now a safe distance away
from C, publish C to the Blind bot
5: If C was published, the Blind bot travels to C
6: Wait until the Blind bot has finished moving
if it was sent C
7: If C was sent, reset C to prepare for the next iteration

2) RETRIEVAL: The scope of the RETRIEVAL state was severely reduced. Instead of the original searching algorithm to attempt to locate the Blind bot, the RETRIEVAL state now simply plans a path to the last known location of the Blind bot. Ideally, this would always allow for the Seeker bot to travel to the Blind bot's location, identify it, and transition back into the RETURNING HOME TOGETHER state to bring both bots back home. We do maintain that if the RETRIEVAL state fails to locate the Blind bot, that it transitions back into the EXPLORING MAP state to continue searching.

V. Results

A. Quantitative Results

Perhaps the most crucial component of the SARA system is the ability to accurately and efficiently update the position of the Blind bot in the world frame. In an effort to quantify the positional error of the AprilTag detection system, our team compared the true position of the Blind bot along the x-axis to the detected position of the Blind bot using our detection system. To do this error quantification, our team physically measured a starting distance from the Blind bot to the Seeker bot, then used the Seeker bots odometry to measure a whole set of linear distances from the Blind bot. This is acceptable because the Seeker bots odometry was shown to be extremely accurate for linear distances of less than 4 meters in previous Botlab reports. Fig. 5 shows the results of this experiment and indicate the positional error of our Blind bot detection pipeline for a set of distances along one axis.

Fig. 5. Position error of AprilTag based robot detection system along the x-axis of the Blind bot

As seen in Fig. 5, the positional error of our system increases as distance increases. This error, however, is extremely promising for relatively short distances. These data heavily influenced logic within our state diagram, suggesting the need for a state to position the Seeker bot within an acceptable distance of the Blind bot for an accurate detection.

Estimating the x, y position of the Blind bot is only half of the complication of our detection system, however. The Seeker bot must also attain very accurate headings of the Blind bot in order for our proposed system to perform well. After implementing our system, the heading error was measured by comparing the true heading of the Blind bot (computed using odometry) and the estimated heading given by our detection pipeline. This heading error is plotted in Fig. 6 where theta values of $-90°$ to $90°$ are considered. During these trials the position of the Blind bot was almost exactly 0.5m away from the Seeker bot.

Fig. 6 depicts the heading error in two separate Blind bot configurations. In the first configuration, the Blind bot only contains one AprilTag (facing forward) indicating its heading. As seen in the figure, the heading error in this configuration starkly increases after the true robot heading surpasses $-45°$ or

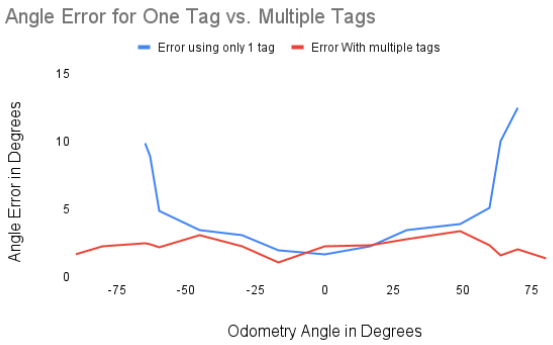

Fig. 6. Plot of the heading error of our detection pipeline for a Blind bot with a single AprilTag and a Blind bot with multiple unique AprilTag

$45°$. This is because at higher angles, the AprilTag becomes more out of plane with the camera plane, making it harder to detect the corners needed to fit a homography. To combat this error, our team decided to use multiple unique AprilTags plastered on 4 sides of a cube atop the Blind bot. The benefit of this configuration is that there will always be at least one AprilTags that is a maximum of $45°$ out of plane with respect to the camera. We found that this modification made the detection much more accurate for larger angles and allowed for the detection of the Blind bot in any heading. The average heading error of the Blind bot in this configuration is $2.2°$ which is more than sufficient for our applications.

When considering both the positional and heading error of our detection system, we concluded that largely spaced waypoint would allow the Blind bot to drift to an unsafe location. Because of this, our team rejected the idea of simply computing the pose of the Blind bot and sending it a list of waypoints to an ultimate destination. Instead, we compute intermediate waypoints that allow our detection system to constantly correct itself. We call this approach the ”tug and pull” method.

B. Qualitative Results

Below are the Botgui pictures of showing exploring and retrieving path.

At first, the exploration step. The yellow part indicates the Seeker bot’s pose while the purple part is the Blind bot’s pose. Note that while during exploration, the Seeker bot has not detected the Blind bot yet. For demonstration purpose, we set initial unknown Blind bot positions to $(0, 0)$ to dramatize when we don’t know the Blind bot’s location exactly, as shown in Figs. 7-10.

Fig. 7. Map exploration

When the Seeker bot first detected the Blind bot, it would turn to face directly towards the Blind bot and send out the first pose estimate and waypoint transformation to the Blind bot. (Fig. 8.)

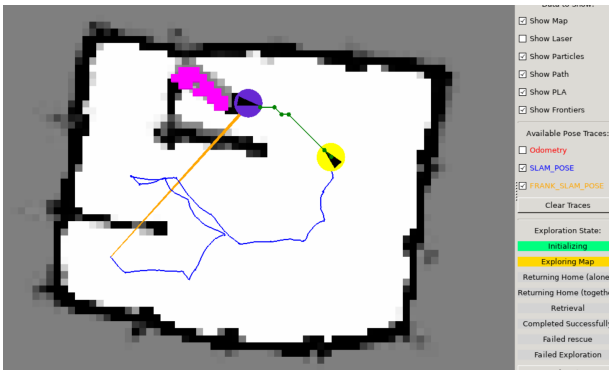

Fig. 8. First detection

During the retrieving process, the Seeker bot would achieve waypoints as inverse pose, performing backward movement while it reached every setpoint. In this way, we can ensure that the Seeker bot always has a positive detection for the Blind bot and send out transformed waypoints to the Blind bot. (Fig. 9)

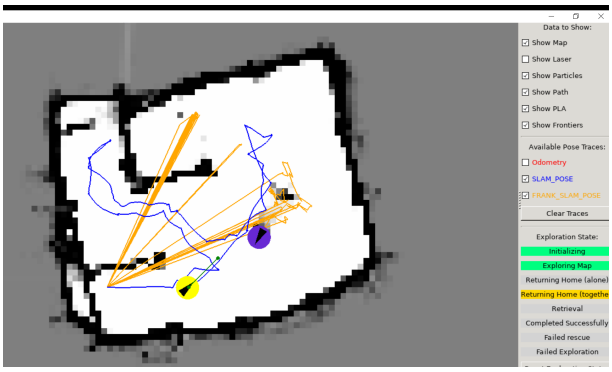

Fig. 9. Retrieving

Finally, both the Seeker and the Blind bot returned to the same waypoint as the result of the retrieving process. (Fig. 10)

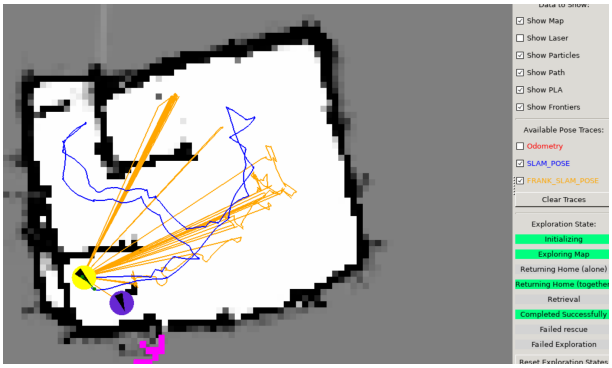

Fig. 10. Return to destination

C. Overview

Overall, the performance matches our expectation as all the appeared errors were in acceptable range and most of the rescue missions were successful. To evaluate the general performance of retrieving, we perform the task several times and record the number of success/failure in the below table.

Trial No.	Success/Failure	Time
1	Success	200s
2	Failure	∞
3	Success	216s
4	Success	208s
5	Success	192s

The four successful missions, mentioned in the table above, all occurred when the Blind bot was in a roughly similar position to the one depicted in Fig. 8 with varying angles and slightly varying positions. The failed trial noted in the table above occurred when the Blind bot was placed in the narrow passageway adjacent to the starting location. We found that our system currently struggles with guiding the Blind bot down narrow locations with sharp turns, which is something that we believe we could easily improve if we had more time to work on this project.

VI. Discussion And Future Work

A. Waypoint Accuracy

Since the Blind bot lost external sensors, such as LiDAR sensor, we were getting the pose of the Blind bot from its odometry readings. However, the odometry pose might be inaccurate due to the possible change of hardware (e.g. the wheel base difference) and calibration error. During retrieving process, since we relied entirely on the odometry pose from the Blind bot, the error would accumulate and made the waypoints following worse and worse over time. Our current solution is to reset the odometry pose each time the Blind bot reaches the next waypoint.

There could be two solutions in the future that we can implement to improve the waypoint accuracy. Firstly, we can add more precise and stable hardware component for odometry poses, such as a more accurate encoder and solid wheel base. This could reduce the error during the calibration and provide us a more accurate odometry pose. Secondly, we can

improve the algorithm with error adjustment. While the Seeker bot detect the difference between the settled waypoint and the actual location of the Blind bot, Seeker bot would send additional commands to the Blind bot to adjust its position closer to the waypoint.

B. Global Optimal Return Path

During the retrieving process, the Seeker bot would generate the return path using $A*$ based on the current map it has explored until it detected the Blind bot. However, this retrieve path only guarantee the local optimal solution. There is a possibility that the global optimal path existed in the unexplored part of the map. Fig. 11 showed below indicates the current problem.

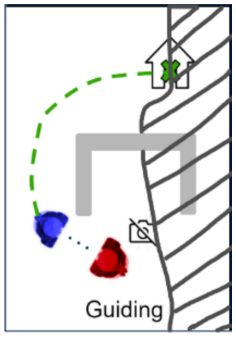

Fig. 11. Shadowed gray part being unexplored map

In the future, we can program the Seeker bot to remember the position it detected the Blind bot and then keep exploring the whole map. After the map is fully explored by the Seeker bot, it will move back to Blind bot and start the retrieving process based on the entire map.

C. AprilTag Detection Sensitivity

Our current implementation of AprilTag detection has the issue that the camera attached on the Seeker bot can only detect the AprilTag within a limited rage, about 5 cm to 20 cm. If the camera is too close, the vision would be restricted and only detected part of the AprilTag. If the camera is further away, the detection would be inaccurate and the transformation matrix feedback would ultimately impact the coordination transformation.

One solution would be switch from pi camera to another higher resolution camera, or fish-eye camera to increase the horizontal detection range. Another more general generation solution is to increase the amounts of detected AprilTags. More AprilTags would improve the detection accuracy in general. Instead of attaching 4 tags around the Blind bot, we can actually put 8 AprilTags in both ordinal and cardinal directions. In this way, we can ensure at least 2 AprilTags were detected each time to increase the camera-to-tag transformation accuracy.

D. Separate Return Location

Our current operation would guide the Blind bot to the starting location of the Seeker bot. However, in reality, the Seeker bot and the Blind bot might return to different locations, such as a separate maintenance site. In the future, we can generate optimal return paths to an unique destination and spread waypoints along that specific path for the Blind bot to follow.

E. Multiple Bots Retrieving

For demonstration and time efficiency, current SARA system only performs one-Seeker vs. one-Blind bot. In the future, there might be situations which require multiple Seekers to coordinate and multiple Blind bots to be rescued. SARA can be extended for mutual communication among Seekers to explore and build the map together, while each robot operates its own rescuing mission. The cooperation among multiple Seeker bots would make the searching and retrieving process more efficiently.

F. Machine Learning Improved Pose Estimation

Using AprilTag detection for pose estimation on the Blind bot is a straight-forward method. However, in the real-world scenario, the malfunctioning robot might not be equipped with AprilTags depends on the factory production. Therefore, for a broader utilization of SARA system, the pre-trained machine learning model built on the Blind bot's physical structure would be a wide-ranging application. One possible solution would be training on the appearance of the Blind bot and importing the model into the Seeker bot. With pi camera (potentially a higher resolution camera) visual detection, the Seeker bot could recognize the Blind bot and operate rescuing process.

G. Depth Map Utilization

While the current usage of a LiDAR sensor fits most of the working scenarios, it still suspects to the limited detection range and the environmental noise. The utilization of depth-camera generated depth map would model both indoor and outdoor objects with high-resolution image, which improves the accuracy of detection.

VII. References

References

[1] Aleksandar Petrov, Andrea F Daniele, and Rohit Suri. *lib-dt-apriltags*. https://github.com/duckietown/lib-dt-apriltags. 2021.

[2] Raspberry Pi. *Buy A raspberry pi camera module 2*. URL: https://www.raspberrypi.com/products/camera-module-v2/.

[3] Melanie Schranz et al. "Swarm Robotic Behaviors and Current Applications". In: *Frontiers in Robotics and AI* 7 (2020). ISSN: 2296-9144. DOI: 10.3389/frobt.2020.00036. URL: https://www.frontiersin.org/articles/10.3389/frobt.2020.00036.

VIII. Appendix

A. Demo Video Link

https://youtu.be/H2apOiehBUE

B. Code Repo Link

https://gitlab.eecs.umich.edu/drewskis/botlab