DEVELOPING AN AUTONOMOUS RECONNAISSANCE ROBOT TO NAVIGATE INDOOR SPACES USING 2D LIDAR BASED SLAM

Kesler Mathieu, Maggie Lau, Hung Tran, Ameera Iftekhar

1. INTRODUCTION

In certain hostile environments, first responders have to navigate through unknown zones that can lead to their demise if the area isn't canvassed beforehand. To provide some context, first responders scenarios can range from police officers trying to deescalate hostage situations to firefighters performing disaster relief efforts after the onset of a hurricane and even military personnel commandeering battlefield strongholds from enemy possession. According to the National Institute of Science and Technology [1], over 83,000 firefighter related injuries (which totaled to $11.8B in 2005) are due to the lack of real-time analytics that can provide first responders critical insight into a building's deteriorating infrastructure.

A viable solution is for first responders to deploy a lightweight, rugged chassis rover that has a relatively small form factor and is certainly cost effective, from a sensor standpoint, to perform reconnaissance and return back to its starting point for retrieval/collection. Our autonomous vehicle will be subsequently paired with live streaming video to a remote server so that operators can analyze and give further command to ground-level constituents. Autonomous navigation is not a trivial feat given that a machine must skillfully make its way around an environment that it has no prior context to. Given this fact, we will be leveraging as many off-the-shelf library and hardware components to expedite the development of our use case. We define our measure of success to adhere to the following: ***within a 5ft x 5ft enclosure, the rover must navigate from its starting grid cell and successfully come within 1 sq ft of the target grid cell while avoiding fixed obstacles; the rover must then return to its starting grid cell and successfully come within 1 sq. ft of said target while avoiding the same static obstacles***. In order to accomplish this feat, the rover must execute a series of accurate turns, in conjunction with straight-line travel, to come within 1 sq. ft of the target. Adding or re-configuring obstacles upon successive trials will be performed to test robustness of the solution.

2. BACKGROUND

Simultaneous Localization and Mapping or SLAM is concerned with the objective of creating a map of an unknown environment while the rover must simultaneously ascertain its location in that space. SLAM, by definition, 'stitches' a collection of point cloud maps, which are essentially generated from distance and angle data from the rover to a nearby object. There are a number of physical sensors that can accomplish this task but one common approach is to leverage a LiDAR sensor. A LiDAR sensor essentially works by shooting a beam of light from an internal emitter to an outbound target. The light beam returns to the LiDAR assembly, specifically the receiver, and its elapsed time is measured. Using the distance formula, we can intuitively calculate the distance from the assembly to an outbound target. LiDAR is an excellent choice for SLAM algorithms to generate a high fidelity map as certain models can sample the environment as much as 8,000 times per second!

Post environment perception, we must transform this real-time mapping into an occupancy grid map. An occupancy grid map is a 2D array that contains the likelihood in which a cell contains an object. A grid cell containing the value of 0 indicates that the cell is indeed a free space in which a rover can safely navigate to that coordinate space. A cell containing the value of 1 indicates a strong likelihood that an object exists, meaning the rover should not traverse to this grid space. A value of -1 indicates that this region is unexplored (i.e. the laser beam emitted from the LiDAR sensor did not pass through that grid space). The inception of this binary grid structure allows us to create a linkage of free neighbors, which is essential for planning a contiguous free path.

The final step to reach a level of autonomy has to deal with path planning. There are a number of path planning algorithms that essentially try to solve the same task: to find the shortest path in the form of a continuous sequence of non-blocking points that leads to its goal destination while avoiding points along that path that are deemed or marked as obstacles. The path planner can be broken down into 2 phases. The first is finding neighbors where given the rover's current location, an occupancy grid is read to determine if adjacent cells are indeed free. The next is calculating the shortest path in which a collection of recursively neighboring cells are explored until target is found. The shortest path is formulated as the lowest cost from starting to endpoint. Taken in conjunction, we get an overview of how autonomous navigation can generally be achieved - perception accomplished through SLAM that's coupled with a shortest path algorithm.

3. LITERATURE REVIEW

3.1. BreezySLAM

BreezySLAM is an open-source Python library for Simultaneous Localization and Mapping (SLAM). This particular version of SLAM uses Python C extensions to wrap existing SLAM algorithms, and is therefore able to run almost as fast as C. This algorithm has been tested on the Virtual Robot Experimentation Platform (V-REP) Simulator and the Neato XV-11 using 2D modelings for maps. The purpose of creating this library was to have SLAM be accessible to more people since Python is one of the more popular and accessible languages among students and researchers. Python can also be more easily run on a Computer-on-a-Module (COM), Raspberry Pi, Gumstix Overo. There are no SLAM algorithms made for Python, so this library had to be compared to and from C algorithms. DP-SLAM, while an incredibly effective C algorithm, takes up a lot of memory as it is an exhaustive full version of SLAM in C. DP-SLAM is therefore an impractical choice for many common hardware robotic implementations as this much memory cannot easily be stored on a COM or System-on-a-Chip (SoC). CoreSLAM, since it only stores a single position and map at a time, uses much less memory. BreezySLAM was developed from translating CoreSLAM into Python and replacing sections with C extensions in Python. It was tested initially in V-REP on the K-Junior model in the simulator, using remote-API over ROS so it can be more accessible outside of Ubuntu. After development and testing in V-REP's K-Junior model, BreezySLAM was then implemented on the Neato XV-11, a robotic vacuum cleaner. NeatoPylot, an AutoPylot program, was developed to allow users to drive the Neato using a joystick, as well as provide a server for LIDAR data. It can be run on Raspberry Pi or COM. The implementation of BreezySLAM was successful on the Neato XV-11. Next steps are the develop BreezySLAM for Java and Matlab, as a version for C++ has at this time been create, to allow BreezySLAM to be accessible to all. [2]

3.2. Path Planning in Robotics

Path planning utilizes a set of waypoints that a robot must travel along to create a path. This path is generated using several criteria and optimizations. It is a very important implementation in autonomous robots as path planning allows robots the ability to find an optimal and collision-free path. Common features of a path-planning algorithm include that given start and end goal position or pose, the algorithm must return a set of states, which may include positions and/or velocities, that the robot must use to reach the end goal, the algorithm must generate a collision free path, the path must optimize some heuristic or parameter, and the path should be physically traversable by the robot. Path-planning algorithms can be separated into two types, graph-based and sampling-based. Graph-based algorithms overlay a topological graph on a robot's configurational space and use this to search for an optimal path. Sampling-based algorithms either represent the configuration space with a roadmap or build a tree that is generated by randomly sampling states in the configuration space. [3]

3.3. Using the SLAMTEC RPLIDAR on a Raspberry Pi

This article provides an overview of LiDAR and insights on integrating LiDAR with a Raspberry Pi. LIDAR is one of the sensors used in autonomous vehicles (AV). It allows the AV to know how far away things are 360 degrees around it. It works by sending out pulses of laser light which bounce off surrounding objects and is read by an optical receiver. Based on the time it takes for the reflected light to arrive at the receiver, the distance to an object can be calculated. The LiDAR discussed in this article is the SLAMTEC RPLIDAR. It connects to the Raspberry Pi via the USB port. Drivers and other software are needed to collect and visualize data from the LiDAR scan. It also has the capability of integrating with a robotic Operating System (ROS). The LiDAR has detection range of 12 meters, angular resolution of 1 degree, and sampling rate of 8000 samples per second. The output of the scan is a set of tuples containing the distance and angle. [4]

3.4. RPLiDAR A1 Development Kit User Manual

This development kit user manual provides instructions on connecting the LiDAR to a host computer running Windows. The sensor requires a Virtual COM port (VCP) driver to make the USB port device appear as another COM port to the PC. SLAMTEC also provides a GUI demo that runs only on Windows. The main purpose of RoboStudio is for testing and evaluating the LiDAR sensor. The RoboStudio software and user manual can be downloaded here. [5]

3.5. Common Challenges with SLAM

Localization errors accumulate: Because the sequential movement is estimated with SLAM, the location of the robot will have some margin of error. Even if the algorithm had a 99% accuracy, there is still 1% error. Over time, this error will accumulate and can cause large deviations from the robot's actual location. Localization failures cause the position on the map to be lost: The localization is susceptible to errors and noise from the data and does not consider characteristics of the robot's movements. For example, a calculation can show a robot moving at 1 m/s suddenly jumping forward by 10 meters. This would generate discontinuous position estimates. High computational cost: With the high frequency of sampling and the calculations needed to develop the map and keep track of localization, the computational cost for SLAM can be very high. This is especially true if the robot is using image data from cameras. Additionally, the hardware is often limited on robots that are doing SLAM because of their size and practicality in cost. [6]

4. METHODOLOGY: A 9 WEEK TIMELINE

Week 1: 3/6 - 3/13

- Must procure parts related to individual contribution
- Agree on an affordable testing enclosure configuration
- Perform a system architecture/ data modeling sanity check.

Week 2: 3/13 - 3/20

- Be able to wire up, debug and get a simple reading for: Accelerometer, MotorDriver, MotorEncoder, LidarInterface. Create a simple test script and upload it to Github so that code can be shared and validated on rover
- Given that simple script is created, spend time creating a class design to abstract away internals and provide needed methods/functionalities.
- Have an initial mock up of a custom Path Planning algorithm.

Week 3: 3/20 - 3/27

- Complete scripts for Accelerometer, MotorDriver, MotorEncoder are due.
- LiDAR Integration.
  - Connect RPLiDAR to Raspberry Pi.
  - Load driver onto Raspberry Pi to enable communication with RPLiDAR.
  - Develop/load code to spin the LiDAR motor and collect measurements.
  - Prepare a buffer to store measurement data.
  - Process/plot the data to ensure it makes sense.
  - Publish the data for use to navigate the robot.
  - Test configuration of RPLidarInterface with BreezeSLAM lib to validate if we're able to get a consistent reading without hassle

Weeks 4 through 7: 4/3 - 4/24

- Create entry point file that adheres to: Perception, Mapping, Localization, Route Planning and Motor Actuation. Consider this resource as such to help define such system
- As illustrated in *fig.1* below, the system design flow will be getting data from the LiDAR sensor, using the data to create a point cloud map, using the point cloud map to create an occupancy grid, using the occupancy grid for path planning, engaging the motor drivers to move the rover according the the path planning.

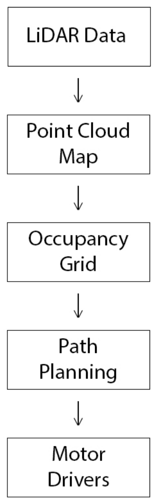

Fig. 1. System Design Flow

Week 8: 4/24 - 5/1

- Test rover within the enclosure and see if the system is dynamic enough. Upon success, proceed with further trials by starting it at different points and rearranging obstacles to confirm efficacy.

Week 9: 5/1 - 5/8

- Work on final video submission and write-up.

5. DIVISION OF LABOR

5.1. Ameera Iftekhar

The rover comes with integrated motor encoders to report the distance traveled of each motor. The motor driver will have a script written in python to control the direction of the vehicle reactively to maps created by sensor data in order to avoid obstacles. This script will be structured similarly to how the authors of the Cornell paper structured their robotic controls, using principles learned in class of wheel odometry. In order to communicate with the Raspberry Pi, one must make use of the pigpio library in python to have the motor encoder interface with the GPIO pins on the module to induce movement. This script will be tested on the motors in the Viam Chassis Rover. Success will be determined in two stages; whether or not the script is able to properly communicate actions to the motor drivers, and whether or not the motor drivers can work in conjunction with the sensor array on this rover to avoid obstacles.

5.2. Maggie Lau

I worked with the accelerometer to get the data from the sensor and returned the change in acceleration for use with SLAM. This accelerometer data told us how far the robot had traveled in respective planes. Along with the LiDAR and motor encoder data that my teammates were working on, I used the BreezySLAM package to map the environment while simultaneously determining our position in this environment. SLAM consisted of multiple parts, including data reading, data mapping, robot state, state update, and map update.

5.3. Hung Tran

I was responsible for getting the LiDAR to function with the Raspberry Pi and publishing data that was usable for navigating the robot. This involved loading the correct driver(s), creating Python code to enable the LiDAR to collect distance and angle data, and processing the data to ensure accuracy. I relied on SLAMTEC's documentation for getting the LiDAR up and running. I developed python code to plot the data and create visualizations for the final report. Finally, I investigated ways to publish the scan data for other robot functions to use.

5.4. Kesler Mathieu

As a developer, I was responsible for the path planning portion of this assignment. After the robot is able to localize and gain a map of the 2D environment using SLAM, the next step is for this vehicle to take the necessary continuous steps in order to get from its starting point, circumvent the enclosed environment, and return back to the starting place. From my research, I couldn't find the necessary off-the-shelf python libraries that I can easily leverage for this use case so I implemented a custom approach leveraging on this course on path planning basics. Additionally, I wrote the business logic that coupled the path planner to discrete rover movements (i.e. left, right, forward, backward) using VIAM SDK. From a team lead perspective, I was in charge of designing and ensuring the validity of our system architecture, took responsibility to procure the hardware and enclosure for experimentation, and lead weekly meetings to ensure team deliverables were on track.

6. RESULTS

Collection of angle data has been taken in three trials (Trial 1, Trial 2, Trial 3) where an external, gyroscope was placed on top of the rover to record the orientation at each phase of movement.

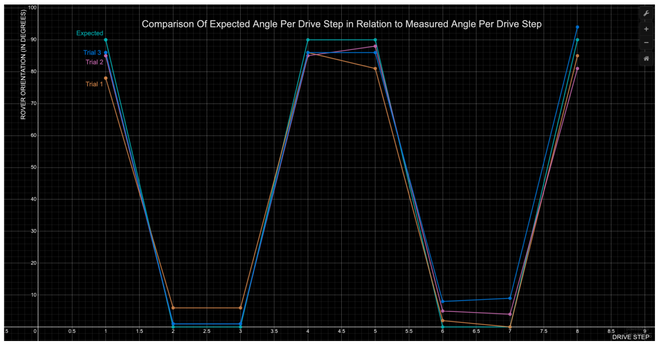

Fig. 2. Graphical Comparison Of Expected Angle Per Drive Step in Relation to Measured Angle Per Drive Step

Drive Step	Movement Descriptions	Expected Orientation Angle	Trial 1 Orientation Angle	Trial 2 Orientation Angle	Trial 3 Orientation Angle
1	Turn Left and Drive 1 Foot	90°	78°	85°	86°
2	Turn Straight and Drive 1 Foot Backwards	0°	6°	1°	1°
3	Turn Straight and Drive 1 Foot Backwards	0°	6°	1°	1°
4	Turn Right and Drive 1 Foot	90°	86°	85°	86°
5	Turn Left and Drive 1 Foot	90°	81°	88°	86°
6	Turn Straight and Drive 1 Foot	0°	2°	5°	8°
7	Turn Straight and Drive 1 Foot	0°	0°	4°	9°
8	Turn Right and Drive 1 Foot	90°	85°	81°	94°

Fig. 3. Chart Comparison Of Expected Angle Per Drive Step in Relation to Measured Angle Per Drive Step

$$e_i = (expected\,angle_i - measured\,angle_i)$$

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

The mean absolute error (MAE) with regards to the rover's orientation per trial are: **5.50, 4.00, 4.38** respectively. These values are relatively close to 0 which demonstrates the angles for each line segmentation are relatively close to what's expected thereby illustrating favorable navigation when deployed into other static indoor spaces.

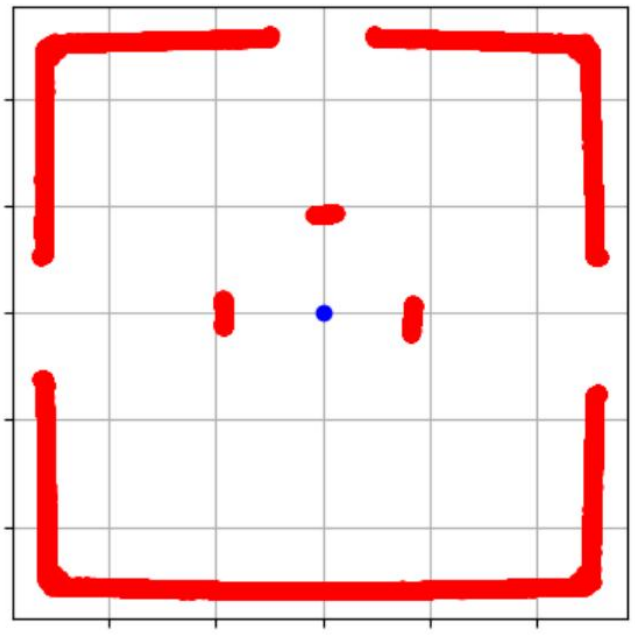

Fig. 4. Point Cloud Map

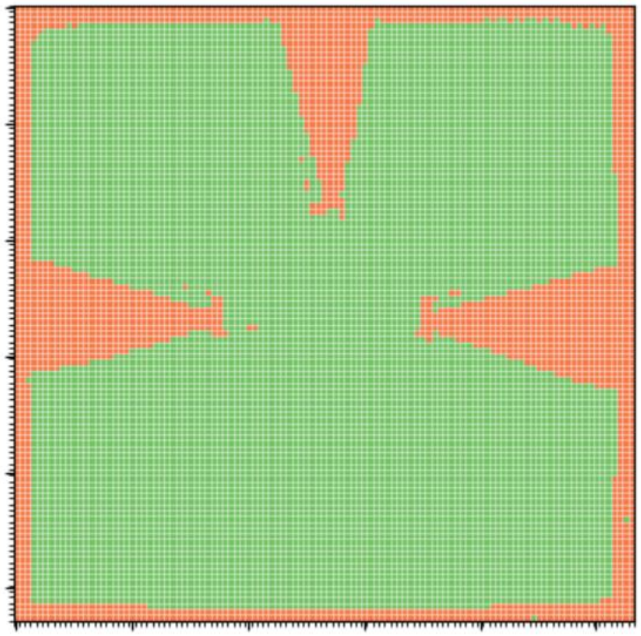

Fig. 5. Occupancy Grid

7. NEXT STEPS

With more time allocation, we would pursue the following objectives. Integration with a reliable SLAM package is paramount. Given it's current state, it's not a complete autonomous vehicle as the starting point and occupancy grid of the enclosure are hard-coded. We looked into the VIAM Cartographer SLAM service. Our initial attempt to integrate with VIAM SLAM encountered a permission related setback on our Raspbian distro. An appealing feature of VIAM platform is the level of support from its technical staff members. Given more bandwidth, we would work with supporting staff to get this SLAM service integrated into our rover so it can dynamically generate the occupancy grid, as well as ascertaining its location in the enclosure, before the navigation process commences. The point cloud map and occupancy grid are shown in figures 4 and 5.

From a hardware modification perspective, in order to ensure the rover is accurately oriented at 90 degrees after every iteration, an improvement to the system would be to include a gyroscope. After every rotation or straight drive the rover performs, it will essentially have a feedback-control loop built in so the rover can continually make slight adjustments until it reaches some threshold of the source of truth, being the gyroscope. In theory, accomplishing this will significantly decrease our MAE values on subsequent trials to be marginally close to 0, thereby showcasing an improvement in navigation accuracy.

Lastly, the use of Dijkstra's algorithm for shortest path synthesis is a computationally expensive task as there is no heuristic involved to present it with a bias to only explore nodes in the direction of the goal node. Transitioning our shortest path implementation to a algorithm that leverages a heuristic function (e.g. the A* algorithm paired with Manhattan distance) would yield benefit from a power management perspective, thereby extending battery life in the case the rover's environment scales beyond a 5x5ft enclosure.

8. CONCLUSION

This report highlights the rover's navigation performance in static indoor spaces. The trials of the rover's orientation resulted in mean absolute error (MAE) values of 5.50, 4.00, and 4.38, respectively, which are remarkably close to 0. This close proximity to the expected angles demonstrates the rover's favorable navigation capabilities when deployed in static indoor spaces. To further improve its capabilities, integration with a reliable SLAM package, inclusion of a gyroscope for accurate orientation, and transitioning to a more efficient shortest path algorithm are recommended. These enhancements will contribute to increased navigation accuracy, dynamic occupancy grid generation, and extended battery life, enabling the rover's successful deployment in larger environments.

9. REFERENCES

[1] M.A. Holmberg, D.G. Raymond and J. Averill, "Delivering building intelligence to first responders," in *National*

Institute of Standards and Technology. United States Department of Commerce, 2013.

[2] S. Levy Bajracharya, “Breezyslam: A simple, efficient, cross-platform python package for simultaneous localization and mapping,” Washington and Lee University, 2014.

[3] Electronics and BITS Goa Robotics Club, “Path planning in robotics,” in *ERC Handbook*. Birla Institute of Technology and Science, 2020.

[4] Dave Astels, “Using the slamtec rplidar on a raspberry pi,” Adafruit, 2019.

[5] Dave Astels, “Rplidar a1 development user kit manual,” Shanghai Slamtec.Co., Ltd, 2021.

[6] “What is slam?,” MathWorks.