Autonomous Exploration and Mapping Payload Integrated on a Quadruped Robot

Julian Y. Raheema[1,2], Michael R. Hess[1] Raymond C. Provost[1], Mark Bilinski[1], and Henrik I. Christensen[2]

[1] Naval Information Warfare Center Pacific, San Diego CA 92152, USA,
`julian.y.raheema.civ@us.navy.mil`,
URL: `https://www.niwcpacific.navy.mil/`

[2] University of California San Diego, Cognitive Robotics Laboratory,
La Jolla CA 92093, USA

Abstract. The world is rapidly moving towards advancing and utilizing artificial intelligence and autonomous robotics. The Boston Dynamics quadruped robot, Spot, was designed for industrial and commercial tasks requiring limited autonomous navigation. There are a few successful methods to enable an autonomous explorer Spot system. Still, none of them achieved the application level due to complexity, weight, and occupying the entire back of the Spot robot, leaving little to no space to integrate other sensors or payloads for different purposes. The state-of-the-art autonomous system solution requires at least three depth cameras, a single LiDAR scanner, an extended GPU-dependent computer, extended power, specific hardware, an extended battery, a base station for command and control, and 3-4 people to operate. Our research aims to reduce the complexity of hardware, software, and operation to achieve a usable autonomous system. Our research uses fewer and less expensive sensors and, importantly, operates in GPS-denied scenarios, with only one person to manage and no base station needed to command and control. Our autonomy stack runs on our single Spot CORE i5 computer mounted on the back of the robot. The operator controls the area of exploration, defines the operational zone and sends the command to have Spot explore, generate 2D and 3D maps of the environment, and return to the start location to await the operator's command. It is possible to enhance the Spot robot with a simplified autonomy module and utilize it in many tasks, including disaster response, nuclear inspection, mine inspection, etc. Other less extreme use cases include autonomous 3D and 2D scanning of facilities for inspection, engineering, and construction purposes. We describe the system design, its implementation, and a number of experiments.

Keywords: Autonomous, Exploration, Follower, Robotics, SLAM

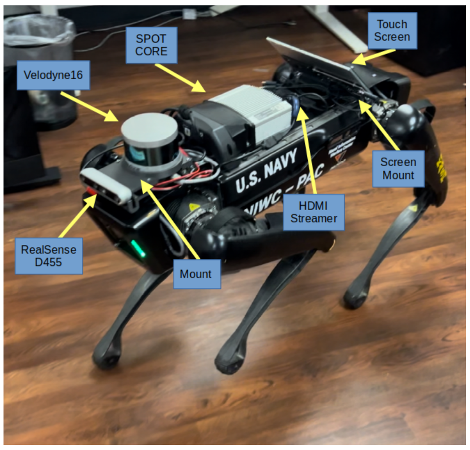


Fig. 1: The Boston Dynamics Spot robot with the autonomy hardware package mounted on its back: A Velodyne 16, RealSense 455i, Spot CORE, and Touch Screen.

1 Introduction

The advancements in creating and using technology are changing rapidly. However, more intelligent robots that perform autonomously in challenging environments are still rare. Using 3D scanning of facilities has numerous use cases, from repair jobs over archaeology to nuclear safeguarding. One possible approach is through 2 to 3 engineers using FAROs scanners on a tripod positioned at multiple locations with enough scan overlap to allow stitching to build a complete 3D model. This process may require 3–4 scans for an area of 10x10 meters, while large / more complex locations require many scans to generate a complete model. The second challenge is scanning while not interfering with regular operations. Manual 3D scanning is more difficult when the facility is in use since employees or customers are performing their duties inside a facility, leading to degradation in the quality of the scan or delaying it until it is clear. Because of the above-mentioned challenges, we desire to automate this task and create a capability for future applications. We present an autonomous system module that can be integrated into a legged robot to navigate through multi-level facilities and complex environments to execute a given task.

The paper presents a lightweight approach to bringing a fully autonomous capability to Spot robots. Our autonomy stack can navigate, explore, and avoid obstacles and generate 2D and 3D maps for an unknown environment. Our system design allows the operator to command the robot using a laptop, and we mount a screen on the Spot robot for visualization.

We discuss our System Architecture by going through our autonomous system's hardware and software stacks required to make a robot autonomous. We leverage a LiDAR sensor and RealSense camera to generate the 2D and 3D maps, a Spot CORE computer for computation, and a touch screen to visualize the collected map during the operations or after. We then outline our operating system, ROS framework, and required ROS packages to run our autonomy stack. We examine the graph-based SLAM algorithm [1], Real-Time Appearance-Based Mapping (RTABMAP) [2], used to localize and map the robot during operation, our navigation stack and the ROS packages used to navigate the environment safely and accurately, and the exploration package algorithm and return home behavior. Finally, we discuss the command and control of our robot and the results of our experiment and evaluation.

1.1 Autonomous System Background

We split our background study into two folds. The first will explain the payload hardware and software aspect, and the second will discuss exploration algorithms. For the payload system, we found that the SubT DARPA Challenge occurred in three stages from 2018 to 2021, with virtual and physical competitions in underground environments like tunnels, caves, and industrial complexes. Teams deployed autonomous robots with sensors to explore, map, and navigate the terrain, detect objects, and navigate obstacles under timed conditions. In one hour, the robot must navigate, explore, detect, and report artifacts in a multiple-kilometer path in the tunnel. One of the teams, Ian D. Miller, and his crew [3] propose a multi-quadruped robot system for this challenge using sensors such as Nvidia Jetson with GPU, 64-beam LiDAR, two pmd picoflexx sensors for short-range depth to help with planning, Intel RealSense T265 stereo tracking camera, stereoLabs Zed mini stereo camera, and other non-autonomy related sensors. For the mapping algorithm, they used GTSAM [4]. Another team [5] used three cameras, LiDAR, Vectornav(IMU), external battery, and Cartographer ROS SLAM package to generate 2D and 3D maps. Team CERBERUS [6] was the first-place winner for the SubT challenge. The team used ANYmal legged robot as their robot platform, and their autonomy stack consists of NUC i7 and multiple cameras, LiDAR, and CompSLAM [7] Team CoSTAR [8] autonomy stack has 3 RealSense cameras, LiDAR, high-intensity LEDs, an IMU, as well as other sensors. We admire and respect all the teams who were the first to create intelligent solutions and no doubt they did an incredible job. However, for application, these setups rely on large amounts of computation and expensive sensor setups – high cost, bulky equipment packages, and large team to operate. Moreover, some software parts are proprietary, making it difficult for others to reproduce the system.

In the second fold, background research for the robot exploration, we found Robert Fisher [9] discusses the complexities of recognizing partially and completely obscured structures and introduces a framework for comparing observed properties with prototypical models to enhance recognition accuracy. We also see Brian Yamauchi [10] applying a Frontier-based exploration approach that enables mobile robots to autonomously navigate and map complex environments. This method enhances the robot's ability to detect frontiers—regions between known and unknown spaces—allowing for efficient exploration and improved mapping accuracy through techniques like laser-limited sonar to minimize errors caused by specular reflections. Another method using Feature-Based exploration [11] by P.Newman et al involves determining the robot's current location and surroundings and deciding where to go next to maximize exploration. The Ayoung and Ryan paper [12] propose a novel PDN that employs a reward framework that balances exploration and revisitation based on SLAM localization uncertainty. This framework allows the algorithm to make informed decisions about when to explore new areas and when to revisit previously mapped regions to improve localization accuracy. In another study by Iker et al [13] reviews techniques for autonomous robot exploration, focusing on strategies like frontier-based exploration, information-theoretic approaches, and Next-Best-View (NBV) planning. It highlights the role of sensor fusion and SLAM for accurate mapping while discussing challenges such as real-time decision-making, resource constraints, and handling uncertainty. Photogrammabot [14] is a ROS-based wheeled robot that navigates autonomously, creates a mapping, and integrates it with mixed reality technologies, demonstrating its effectiveness in creating high-quality spatial models for virtual and augmented reality use cases. As we have seen in most studies, simulation is the only place a robot and computation exist. They are rare to find on quadruped robots. Our paper presents a novel system design with a cost-effective sensor payload. It applies an operational zone frontier-based exploration algorithm to achieve an autonomous exploration and mapping system for an actual quadruped robot.

2 System architecture

2.1 Hardware Architecture

Our stack utilizes Spot, a legged robot, as its base. This robot features five built-in depth cameras that enable it to detect obstacles and ensure safe navigation. The robot is able to achieve a top speed of 3.5 miles per hour(mph), run for 90 minutes, and can carry up to 20 pounds of equipment. The platform's ability to climb stairs and navigate complex environments enhances its versatility and adaptability. The hardware component is an essential part of the system.

Our motivation for reduced hardware design is to allow space for other payloads to be integrated into the Spot robot in a modular fashion, thus the functionality of the robot could be adapted to different scenarios by simply swapping between available modules. Our goal is to shrink the space utilized by the autonomy hardware such that we have a modular bay to integrate our FARO scanner payload. Our survey identified that a 16-channel LiDAR paired with an RGBD camera is sufficient to generate the maps for an indoor environment. We selected a Velodyne puck (VLP-16) and one RealSense depth camera (D455) as input sensors for SLAM. We selected the Spot CORE from Boston Dynamics as the primary computation means. This computer is pre-loaded with an installation of Ubuntu 18.04. The Spot CORE has an Intel i5 four-core CPU, 16GB of RAM, and 512GB of disk space. Although it is not powerful enough for the entire autonomy stack, we wanted to push the limits of what was capable on the Spot CORE and identify points for optimization in our autonomy stack.

We mounted a touchscreen on the back of the Spot to allow more natural interaction with the system's functionalities. We added an HDMI streamer to the Spot CORE for development and debugging purposes. we used the Spot controller for teleoperation and safety E-Stop when needed. Figure 1 shows the Velodyne sensor mounted using custom-made 3D printed parts, the Velodyne interface box and the RealSense camera. The HDMI streamer enabled us to visualize and monitor the Spot CORE computer without running a long HDMI cable while navigating the robot. We used the streamer for debugging and analysis during testing. It allowed us to visualize the waypoints, maps, and other sensor outputs on a larger screen rather than using the Spot screen.

2.2 Software Architecture

Autonomous operation has several benefits, including coverage of an enclosed area. Our autonomous system is designed to be independent of specific hardware or software. For example, if we want to upgrade our Velodyne LiDAR, we can easily switch from 16 to 32 or 64 lines. Similarly, if we find a better depth camera, we can replace the existing one without any issues. Our system relies on sensor data and currently utilizes the RTABMAP SLAM and the Octomap SLAM, which utilize LiDAR data. This setup allows for continued mapping even if the camera sensor fails. In the SubT challenge, teams face the challenge of injecting fog or smoke into the environment, which can distort the depth camera sensor and disrupt the mission. Our system allows for running multiple SLAM algorithms as long as the hardware can handle the computational demands since SLAM requires significant computer resources. Similarly, if we wanted to change the exploration algorithm, it would be straightforward because our software design decoupled from the other subsystems. It would take the map generated from the SLAM algorithm and deploy the exploration technique. All of our system designs have the advantage over other designs because they don't rely on GPUs or specific hardware.

We chose to leverage RTABMAP [2] as it enables easy integration of odometry, camera depth data, LiDAR scans into a joint map that is accessible in 2D and 3D. The maps are used by a lightweight frontier-based exploration strategy [?] to generate a trajectory for the robot, which in turn is sent to the robot controller package from ClearPath [15].

The map is post-processed using the Point Cloud Library [16] to remove outliers. All the processing is leveraging ROS for integration [17]. The full architecture of the system is shown in figure 2.

3 Localization and Mapping

Robot localization is crucial in enabling autonomous systems to navigate their surroundings effectively. A robot can accurately plan and execute paths while

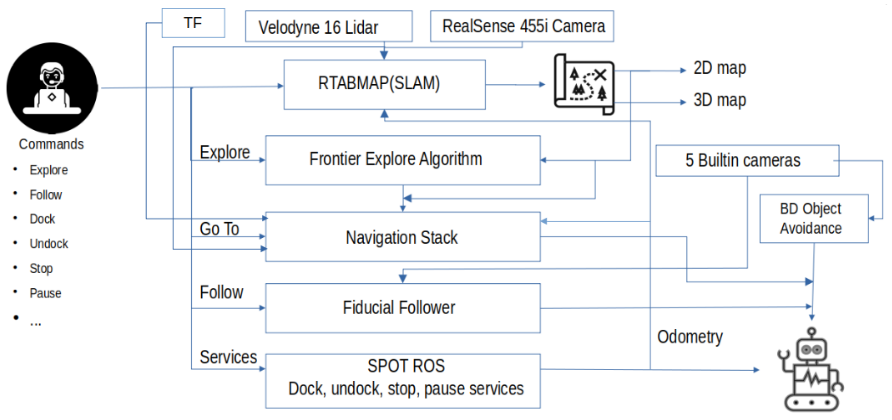
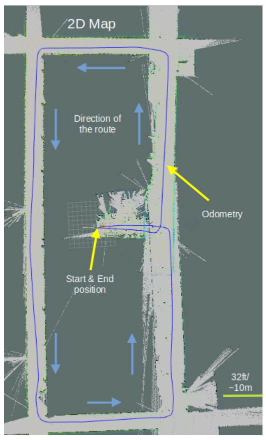


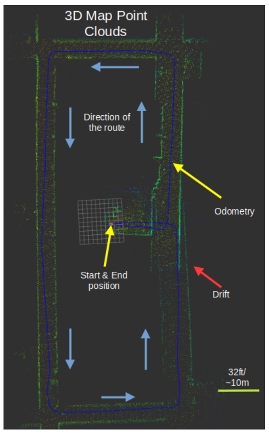
Fig. 2: Software Autonomy Stack diagram; RTABMAP to generate 2D and 3D maps, Frontier Algorithm for Exploration, Navigation stack to move the robot from one point to another, Fiducial Follower for chasing Apriltag, and other services that are more related to Spot robot.

avoiding obstacles or navigating through unknown territories by determining its position within an environment. SLAM algorithms play a crucial role in this process by fusing sensory data from various sources such as depth cameras, LiDAR, and with or without GPS to generate precise maps and localize the robot's position within them.

Our autonomy stack uses the SLAM approach for localization and mapping without relying on GPS for a robot to localize and navigate. We use RTABMAP, a graph-based SLAM that supports visual and LiDAR-based SLAM. We use the TF library for the robot translation [18]. The TF library coordinates the robot's position and orientation over time. For example, the TF of a robot gripper is different from the robot base. Therefore, tracking and computing each robot component frame is both challenging and necessary; thus, the TF package makes the process very convenient. The RTABMAP node subscribes to the Velodyne laser scan topic published, and RealSense camera depth images, and odometry and TF messages published by Spot WRAPPER [19]. The map topics are updated every 1Hz. Once these inputs are available to RTABMAP and synchronized, the short-term memory starts creating a memory for the odometry pose, the sensor's raw data, and other required data for the following steps: the loop closure and proximity detection, local occupancy grid, and global map assembly. The loop closure is a essential part of SLAM algorithms that helps to improve the accuracy of localization. It estimates the map building and the robot position based on revisiting and seeing the previously seen landmark. The RTABMAP node publishes map data, map graph, and 2D Occupancy Grid. The RTABMAP map update rate is roughly every second. In figures 3a and 3b, we demonstrate the 2D and 3D maps and the robot's position.



(a)



(b)

Fig. 3: (a) A 2D map was generated from RTABMAP using a Velodyne sensor and a RealSense 455i depth camera. The blue line represents the robot's odometry path while building the map through the 180+ m course length. (b) A 3D point cloud map was generated from Octamap using a Velodyne sensor. The blue line represents the robot's odometry path while building the map through the 180+ m course length. The red arrow is the drift generated from the Octamap.

4 NAVIGATION

4.1 Costmap and Planning

The navigation stack consists of two main parts: costmap and planning. The costmap is the part of the stack subscribing to sensor data such as depth camera and LiDAR and a map generated from a SLAM algorithm or other resources. Once these sensor data are available for the costmap to process, it will create a world map to enable the planner to generate optimal trajectories that avoid collisions while minimizing distance traveled. The costmap provides a continuous, real-representation of occupied and free space within its environment. The costmap guarantees smooth and efficient navigation for the robot, allowing it to adjust to dynamic environments and achieve its plans effectively.

Planning is the second part of the navigation stack, as it allows robots to decide the shortest route from their current location to a desired destination, considering various aspects such as obstacles, terrain, and environmental dynamics. By utilizing algorithms like Dijkstra's [20] or A* [21] search, robots can optimize their movement patterns while avoiding collisions and minimizing energy consumption, ultimately enabling them to successfully complete tasks in diverse environments.

We based our navigation stack on the ROS Move Base package [22]. This package has an implementation based on ROS actionlib to navigate a robot from one point to another. The Move Base package requires odometry, tf, map, and sensor topics. The move base package loads pre-defined local and global configurations for the planner and costmap. The costmap configuration results as shown in the Figure 4 incorporates the three layers of plugins: Static, obstacle, and inflation. We configured our navigation stack planner to use Dijkstra to navigate from one point to another and enable the navigation of unknown spaces to enhance the exploration feature. We padded the barriers with a 0.25 meter inflation radius. Increasing or decreasing the inflation radius will affect the navigation behavior to be more conservative or aggressive, respectively. We show the local and global costmap in RVIZ, in 6x6 meters, where the robot is at the center of the window.

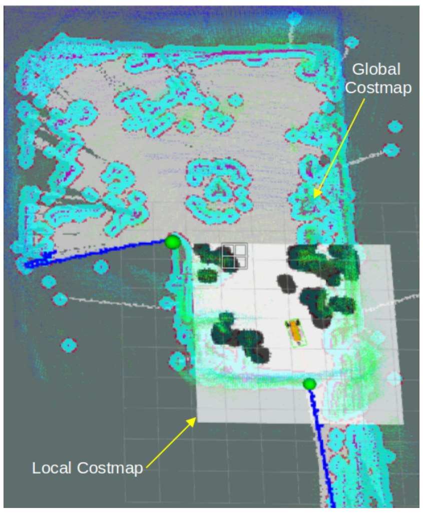


Fig. 4: The navigation stack, move base ROS package uses the predefined local and global cost map and planner to help the robot plan the shortest and safest path using the Dijkstra algorithm.

4.2 Follower Behavior

As mentioned earlier, the Spot robot is capable of using either a controller or an app when manually controlling the robot. Teleoperating a robot with a hand controller is easy, but piloting a robot with a controller for an extended period could be tedious or even painful – causing sore fingers and stressed muscles. Spot SDK has an example where Spot can detect and follow an object of interest. The object detection and follow are based on the TensorFlow model [23] trained on the COCO [24] dataset. The Spot SDK also has an Apriltag [25] detect and follow capability. If any of the five cameras detects an Apriltag, the Spot robot starts to move toward the detected Apriltag. We wrapped and integrated this feature enabling the Spot to follow a specific Apriltag only when the user activates it through the user interface. The Spot camera detect range begins at 0.5 and up to 5 meters in a good illumination room. If the Apriltag and camera distance is less than 0.5 meters or greater than 5 meters, the the detection becomes more challenging therefore, the following feature becomes less responsive. When detection and the distance are valid, the Spot starts navigating toward the Apriltag, orienting its head if the detected Apriltag is from the side or rear cameras. We tested the Apriltag detection and following, and it detects the Apriltag up to 5 meters and avoids obstacles while navigating.

5 Exploration and Return Home Behavior

Robot exploration capabilities are essential in many applications. Some high-level applications for robot exploration include search and rescue, mapping confined environments such as tunnels or hostile environments, environmental assessment after natural disasters, and building a map for unknown environments. We chose to base our exploration method on the greedy frontier algorithm. Initially, a full grid search using BFS is performed and potential frontier points are added to a queue data structure. Then, another BFS search is executed on these frontier points to determine the final set of frontiers. These frontiers are arranged in the order of their Euclidean distance from the current position of the Robot within the queue. Consequently, the algorithm directs the robot to move towards the nearest frontier. Once the robot reaches a frontier waypoint, it makes a 360-degree sweep to build a map of that region [10]. The explore-lite package is lightweight and doesn't create its own costmap, but it uses a generated costmap from a SLAM algorithm, making it light in terms of computational resources and a good combination with the SLAM algorithm. We modified explore-lite ROS package to restrict the area of exploration. We define an allowed search window of (Front, Back, Left, Right) where the window values are real numbers in meters from Spot's current location. When the exploration action is active, the frontier algorithm subscribes to the RTABMAP cost map and generates a list of waypoints for the robot to navigate to. The RVIZ interface shows these waypoints as green spheres. The robot starts navigating to the closest waypoint first and then, once achieved (or NOT achieved after some threshold time limit like 10 seconds), continues on to the next waypoint. We modified the explore-light package to record the starting robot position before triggering the exploration algorithm. The robot pose consists of (x, y, z) translation and roll, pitch, and yaw rotation, but here we only care about x and y translation and yaw data. The return home action is triggered when the user manually stops the exploration or the robot finishes the exploration task and returns to the starting location safely. In figure 5, we show how we visualize the frontier waypoints and the color associated between blue, and red as of in progress and not reachable respectively.

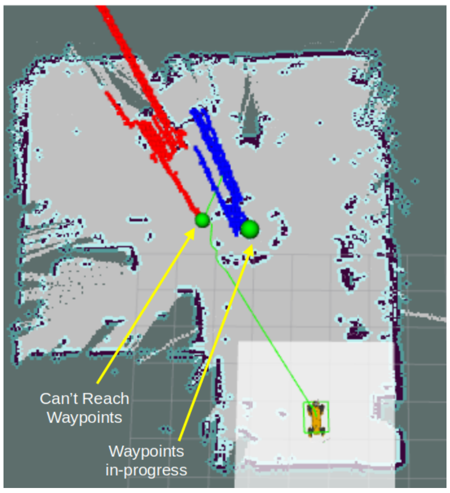


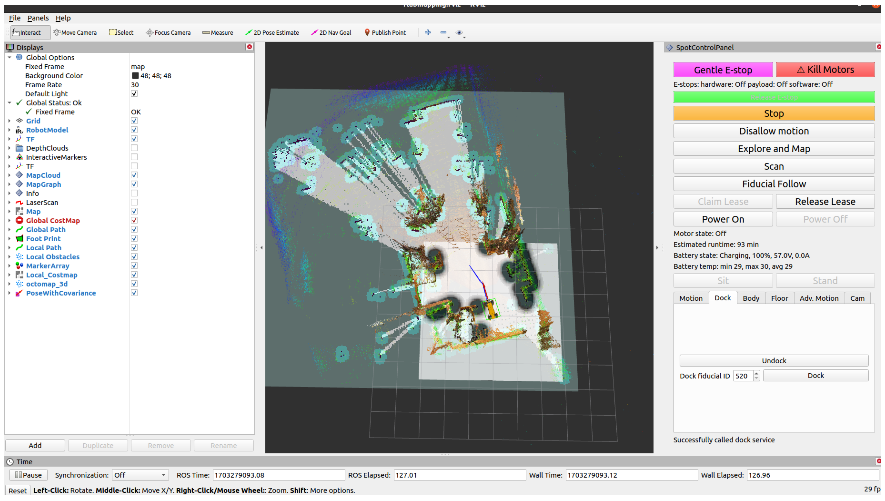
Fig. 5: The modified explore-lite ROS package based on the frontier algorithm enables the robot's exploration capability. The green sphere with blue lines represents the robot's next waypoint to explore, and the red line means the point is unreachable.

6 Command and Control

Users have multiple options to interact with and control the Spot robot. The default way to do so is through the use of the controller. However, users can install the Spot app on an Android phone to control the robot fully. This great flexibility to control over the robot's 6 degrees of freedom (DOF). With this app, operators can adjust settings such as height, speed, docking and undocking, payload, sit/stand mode, camera feedback, and route recording. Many of these services are also available through a computer with the Spot ROS [15] package.

Our main goal is to have the Spot robot primarily commanded and controlled using only one device, such as a tablet, to fire up the autonomy system to explore, map, and perform task-oriented missions. However, since this is not a final system product, our approach includes a touch screen mounted on the back of the robot while still having the option for emergency takeover via the controller if needed. We extend the functionality of the robot by incorporating autonomous exploration, follower, and high-resolution scanning buttons. The selected topics to be visualized are displayed on the left side of the panorama view, while the 2D and 3D maps resulting from this control are shown in the middle. Operators can utilize both the Spot ROS interface and the original controller to control the robot.

Figure 6 shows an example of how the robot's functionality is extended through the use of a touch screen mounted on its back, as well as the option for emergency takeover via the controller if needed. This feature allows for flexibility



Global Options
Fixed Frame: map
Background Color: 48; 48; 48
Frame Rate: 30
Default Light: ✓
Global Status: Ok
Fixed Frame: OK

E-stops: hardware: Off payload: Off software: Off
Motor state: Off
Estimated runtime: 93 min
Battery state: Charging, 100%, 57.0V, 0.0A
Battery temp: min 29, max 30, avg 29
Dock fiducial ID: 520
Successfully called dock service

Synchronization: Off ROS Time: 1703279093.08 ROS Elapsed: 127.01 Wall Time: 1703279093.12 Wall Elapsed: 126.96
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
29 fp

Fig. 6: ROS robot visualization tool RVIZ to interact with the Spot robot.

to control over the robot's movements and actions, making it more valuable in easy to use.

7 Experimental evaluation

This complex system needs extra details when evaluating the performance of the autonomous robot. We must determine the latency and accuracy for the entire stack of mapping, localization, and navigation. The first test is to measure the map quality. To do so, we teleoperated the Spot robot in a 180+ meters closed rectangular course at slow, medium, and fast speeds. The result showed that the medium speed generated less map drift than the slow or fast speed for our robot configuration. The map drift grows more significant the more prolonged the Spot robot is in motion due to the odometry in the legged robot, which accumulates errors faster than a classic wheeled robot.

We tested multi-level building mapping by piloting the Spot for 50 meters, starting the first floor for 50 meters, then through the stairs and a straight line for another 50 meters, going back the same route to the starting location. The resulting map had significant drift in the Z axis, making mapping 3D buildings more challenging at this level of the robot autonomy configuration, which inspired us to create our own odometry sensor and topics in future work. During the localization test, there was a 50cm positional discrepancy between the start and end points over 180+ meters at various Spot speeds.

We tested our autonomous navigation with three different environment settings: Easy, Medium, and Hard. The lab used for testing was 35x30 feet, and we had various obstacles ranging from easy barriers like empty boxes of size 2x4 feet to hard obstacles such as computer tables, chairs, trashcans, and other objects.

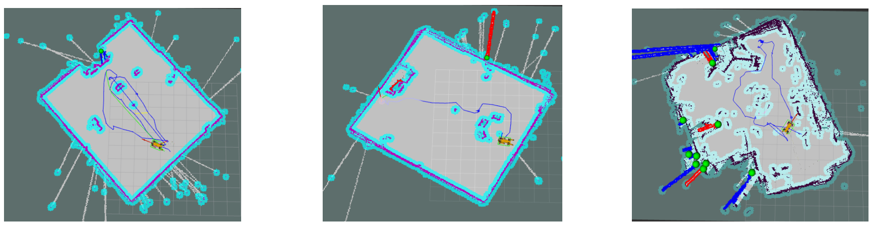

(a) (b) (c)

Fig. 7: Three test environments setup: (a) Easy, (b) Medium, and (c) Hard to evaluate the autonomous exploration and mapping.

Env. complexity	Obstacles Count	Map Coverage Percentage	Minutes to Complete	Area in ft x ft
Easy	5	99%	3.3125	35'x30'
Medium	10	99%	3.9125	35'x30'
Hard	30+	98%	8.8450	28'x25'

Table 1: Table showing the autonomous exploration in three different environment setting: (a) Easy, (b) Medium, (c) Hard.

All our results are based on the average of four runs. We will explain our tests and results in detail as shown in figure 7 and table 1. In the Easy environment setup, our autonomous navigation system demonstrated remarkable efficiency. We conducted our tests in an empty lab area of 35x30 feet x feet with five obstacles strategically placed away from the robot's start location. The system achieved an impressive 99% map coverage, with exploration completed within a mere 3.3 minutes, as detailed in table 1. During the Medium environment setup, we positioned five obstacles in front of the robot, forming a half-circle shape. At the same time, another set of five was placed around the testing area, as illustrated in figure 7 (b). The average outcome of this test was 99% map coverage, with exploration taking approximately 3.8 minutes, as presented in table 1. Our autonomous navigation system showcased its adaptability in the challenging Hard environment setup. This environment, similar to the previous lab but with additional obstacles and narrow passages, posed a significant test for the system. Excluding the inaccessible 7x5 feet area, the system achieved an impressive 98% map coverage. Exploration was completed in approximately 8.8+ minutes, as illustrated in table 1 and figure 7 (c).

We conducted a map analysis to measure the map's quality and accuracy over a highly complex lab area of 31.2x37.9 feet. Ground truth data on the right, such as the figure 8, was captured using a Faro Focus M70 tripod-mounted LiDAR scanner. Each LiDAR scan collects 3D data of everything it can see from its vantage point with accuracy down to 3mm. Multiple scans are needed to capture an entire room, so they must be stitched together to render a complete space.

The data is processed and aligned using FARO Scene. What is shown in figure 8 is an "Overview Map," which is essentially a top-down 2D histogram of the LiDAR data, which renders akin to a 2D-floor plan. Our map generated from the RTABMAP/SLAM map data, such as the figure 8 on the left, is aligned and resized to match the ground truth image size using the affine matrix function. We quantify the map precision, recall, and IOU for the computed map and the ground truth. The precision, recall, and IOU values were **0.9159**, **0.9854**, and **0.9063**, respectively, and the map coverage for the RTABMAP/SLAM was **94.44%**.

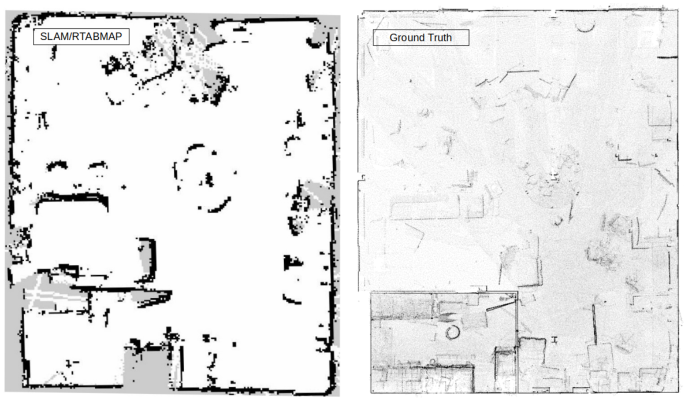


Fig. 8: Map analysis: The image on the right is the ground truth, and the image on the left is the computed map from RTABMAP SLAM

We occasionally noticed that the Spot robot exhibited unsteady movements in an unbalanced way, resulting in some hesitation of the robot while moving from one point to another. Our system has high latency, affecting the robot movement since all our navigation stack and drivers are running on an i5 processor, which is a bit slow for this configuration. We expect this issue to disappear once we upgrade to an i7 or i9 based processor with more cores. The overall result, shown in figure 9, is that we could generate 2D and 3D point-cloud, and 3D depth maps using a fully autonomous robot on lightweight hardware.

8 Conclusion

This paper reported how we built our autonomy stack on the Spot robot using a Velodyne LiDAR sensor and a RealSense 455i camera running on Spot CORE. We used the ROS framework and the RTABMAP SLAM algorithm to localize and generate 2D and 3D maps. We used a greedy frontier algorithm for exploration behavior. We introduced start and stop exploration, operational zones and implemented return-to-start location behaviors on top of the Explore-Lite ROS

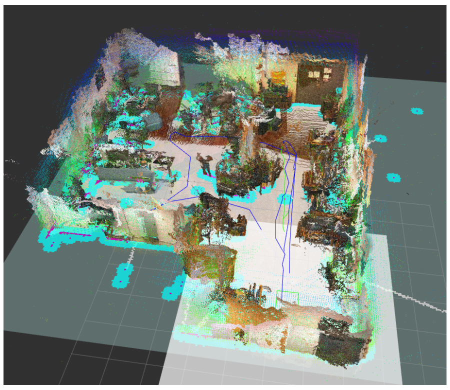


Fig. 9: 2D and 3D maps stacked on top of each other. The blue line is the odometry. The light blue pads are the global cost map surrounding the objects, and the white square box represents the local cost map.

package. We wrapped the fiducial followers feature for Spot SDK into our stack and watched Spot following the fiducial when detected on its built-in cameras. The user can command the robot using touch screen mounted on the back of the Spot. We have demonstrated a cost-effective autonomy stack that empowers the Spot to explore previously unknown areas, generate 2D and 3D maps, and return to the start location for further commands from the user.

9 Future Work

The present system has multiple short-comings and we are actively considering way to improve the system in terms of usability and robustness. Our current odometry system from Spot has significant drift, which limits its accuracy and usefulness for our mapping and navigation tasks. To tackle this challenge, we plan to fuse multiple sensor modalities including IMU to provide more accurate estimates. Nevertheless, there are several additional elements that may impact the effectiveness of our robot's camera-based follower, including variations in lighting conditions. In low light conditions, it can be difficult for the robot to detect Apriltags and follow them accurately. Therefore, we are currently investigating alternative approaches for robot followers that may be more effective in these situations. In addition to improving our odometry and follower systems, we also plan on upgrading the computational hardware in Spot Core. This should provide benefits in latency, which can improve the stability of the robot and enable smoother navigation and motor control. As technology continues to advance, we look forward to integrating better imaging and depth sensors into our autonomy package while maintaining our lightweight configuration.

With these improvements to the system we anticipate that the autonomous exploration and mapping capabilities will start to generate reliable 2D and 3D maps. With accurate 2D maps, we can then explore the high fidelity LiDAR scanning with a FARO scanner mounted to Spot. Once Spot has explored a space, the generated 2D map can be used as input to an optimization algorithm that computes raycasting and overlap thresholds to determine the optimal positions for the tripod based LiDAR scanners. With the scanner mounted to Spot, it can then continue its mission and execute these scans after exploration. The final workflow would enable the autonomous capture of millimeter accurate 3D scans of spaces that have never been seen before.

References

1. G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.
2. M. Labbé and F. Michaud, "Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation," *Journal of Field Robotics*, vol. 36, pp. 416–446, 3 2019.
3. I. D. Miller, A. Cohen, A. Kulkarni, J. Laney, C. J. Taylor, V. Kumar, F. Cladera, A. Cowley, S. S. Shivakumar, E. S. Lee, L. Jarin-Lipschitz, A. Bhat, N. Rodrigues, and A. Zhou, "Mine tunnel exploration using multiple quadrupedal robots," *IEEE Robotics and Automation Letters*, vol. 5, pp. 2840–2847, 4 2020.
4. F. Dellaert and M. Kaess, *Factor Graphs for Robot Perception.* Foundations and Trends in Robotics, Vol. 6, 2017. [Online]. Available: http://www.cs.cmu.edu/~kaess/pub/Dellaert17fnt.pdf
5. A. Koval, S. Karlsson, and G. Nikolakopoulos, "Experimental evaluation of autonomous map-based spot navigation in confined environments," *Biomimetic Intelligence and Robotics*, vol. 2, 3 2022.
6. T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, Jan. 2022. [Online]. Available: http://dx.doi.org/10.1126/scirobotics.abk2822
7. J. Nubert, S. Khattak, and M. Hutter, "Graph-based multi-sensor fusion for consistent localization of autonomous construction robots," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 10 048–10 054.
8. A. Bouman, M. F. Ginting, N. Alatur, M. Palieri, D. D. Fan, T. Touma, T. Pailevanian, S.-K. Kim, K. Otsu, J. Burdick, and A. akbar Agha-mohammadi, "Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion," 10 2020. [Online]. Available: http://arxiv.org/abs/2010.09259
9. R. B. Fisher, *From surfaces to objects: computer vision and three dimensional scene analysis.* New York, NY: John Wiley, 1989, no. 138. [Online]. Available: https://www.researchgate.net/profile/Robert-Fisher-3/publication/220695795_From_surfaces_to_objects_-_computer_vision_and_three_dimensional_scene_analysis/links/0c96053899aceee651000000/From-surfaces-to-objects-computer-vision-and-three-dimensional-scene-analysis.pdf
10. B. Yamauchi, "A frontier-based approach for autonomous exploration," *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles*

for Robotics and Automation', pp. 146–151, 1997. [Online]. Available: https://api.semanticscholar.org/CorpusID:206561621

11. P. Newman, M. Bosse, and J. Leonard, "Autonomous feature-based exploration," in *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, vol. 1. IEEE, 2003, pp. 1234–1240.
12. A. Kim and R. M. Eustice, "Active visual slam for robotic area coverage: Theory and experiment," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 457–475, 2015.
13. I. Lluvia, E. Lazkano, and A. Ansuategi, "Active mapping and robot exploration: A survey," *Sensors*, vol. 21, no. 7, p. 2445, 2021.
14. S. Mortezapoor, C. Sch"onauer, J. R"uggeberg, and H. Kaufmann, "Photogrammabot: an autonomous ros-based mobile photography robot for precise 3d reconstruction and mapping of large indoor spaces for mixed reality," in *2022 IEEE conference on virtual reality and 3D user interfaces abstracts and workshops (VRW)*. IEEE, 2022, pp. 101–107.
15. "Spot ROS Driver Usage — Spot ROS User Documentation 0.0.0 documentation." [Online]. Available: https://www.clearpathrobotics.com/assets/guides/melodic/spot-ros/ros_usage.html
16. Ros-Perception, "GitHub - ros-perception/perception_pcl: PCL (Point Cloud Library) ROS interface stack." [Online]. Available: https://github.com/ros-perception/perception_pcl
17. "ROS: Home." [Online]. Available: https://www.ros.org/
18. "tf2 - ros wiki." [Online]. Available: http://wiki.ros.org/tf2
19. "Boston dynamics spot wrapper." [Online]. Available: https://github.com/bdaiinstitute/spot_wrapper/tree/d96bdbf41445215500fb6bc8e7ae278bb1957746
20. E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, vol. 1, no. 1, p. 269–271, dec 1959. [Online]. Available: https://doi.org/10.1007/BF01386390
21. P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
22. "move_base - ROS Wiki." [Online]. Available: http://wiki.ros.org/move_base
23. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," 2016.
24. T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.
25. E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2011, pp. 3400–3407.