Spring 2023 EECS106B Final Project: Navigation in a Cluttered Environment

Claire Gantan *Team 4* | Andy Fang *Team 4* | Patrick Yin *Team 4* | Charles Xu *Team 4* | Xiang Fei *Team 4* | Jingyu Cui *Team 4*

***Abstract*—Navigation in an unfamiliar, but cluttered environment is a challenge that combines the issues of finding an optimal path with understanding the surrounding environment quickly enough that the robot is able to safely navigate around obstacles. We aim to build upon previous work in learning-based waypoint navigation (LB-WayPtNav) by Professor Somil Bansal by integrating SLAM into his pipeline to provide the necessary metric information to reason about the environment's geometry. As a proof of concept, we show that SLAM is able to successfully navigate situations that proved challenging for the purely learning based approach presented by Bansal.**

I. Introduction/Motivation

This project is a step towards the ultimate goal of developing safe and efficient robot navigation in all environments, regardless of if they are unknown or cluttered. We want to be able to apply the robust plans generated from model-based control into unknown environments. Some motivating applications of safe and efficient robot navigation include delivery robots, autonomous driving robots, and cleaning robots (like Roomba). Delivery robots can be useful for delivering food on the street (such as Kiwibots), medical supplies in a hospital (such as Emancro), an autonomous waiter robot (such as the Matradee L), or packages in a warehouse (such as Amazon).

In class we discussed various methods of planning and navigation for the Turtlebots. For navigation, we used RRT and an optimization planner to plan around obstacles for a simulated Turtlebot to navigate around. However, the drawback of these methods is that we must know exactly where the obstacles are from the beginning in order to create constraints for the planner to create the path. In practice, this is not possible if we are using a robot to explore some new environment.

Professor Bansal's work [1] in LB-WayPtNav tackles the problem of navigation in a novel environment by applying a perception module that tries to generate waypoints for the robot to navigate in a cluttered environment. However, the problem with this approach is that the perception module lacks the metric information necessary to understand the geometries of its surroundings, which leads it to generate infeasible paths.

Our approach is to see how we can leverage SLAM [2] [3] [4] to fill in the gaps for the perception module and provide this metric information. Note that we do not want to use SLAM all the time as it provides an unnecessary computational burden in when navigating in uncluttered, open areas. Our original plan was to integrate SLAM into Professor Bansal's existing pipeline by having SLAM takeover in situations that are difficult for the perception module to reason about. This decision would be made by another classifier network trained on data for cases where the perception module is expected to fail [5]. Unfortunately, due to roadblocks in reproducing Professor Bansal's work, we were unable to figure out a baseline with LB-WayPtNav to compare SLAM against. As a result, we ended up focusing on a proof-of-concept to show that SLAM can be used to successfully navigate difficult situations where LB-WayPtNav would fail to show that it is a viable option to use in conjunction with Bansal's pipeline.

II. Related Work

A. Combining Optimal Control and Learning for Visual Navigation in Novel Environments

Professor Bansal's research looked at combining learning-based perception with model/dynamics based control in order to smoothly navigate through cluttered environments [1]. Learning-based perception is used to define logical and efficient waypoints to get the Turtlebot to a target location, and the best possible path to get to that waypoint is calculated through a dynamics-based controller. In the paper, they found this method to be more effective in real-world scenarios than solely using SLAM or End-To-End mapping. We plan on building off of this pipeline and improving accuracy in navigation by adding in SLAM.

B. Discovering Closed-Loop Failures of Vision-Based Controllers via Reachability Analysis

Reachability analysis aims to develop a method to systematically and tractably figure out which states of the robot will lead to closed loop vision failures. It does so by recasting this problem as a Hamilton-Jacobi reachability problem, which is a verification method that would be used to compute a backward reachable tube [6]. In other words, we want to figure out, given the target states, what states the robot can be in to be able to reach the target state. This research identified specific cases where the robot was unable to reach the goal state, which are the cases we hope to address by implementing SLAM.

C. Potential SLAM Methods

1) Simultaneous Localization and Mapping: Through the Lens of Nonlinear Optimization: This paper presents an attempt at bringing together the advantages of filtering-based SLAM and optimization-based SLAM through EKF-SLAM.

The goal is to make SLAM both efficient and accurate, while modularizing it to be applicable in various applications [3]. EKF-SLAM is a well-known algorithm that can be used in experimentation as a baseline benchmark for the rest of the newer SLAM algorithms mentioned below.

2) A Multi-State Constraint Kalman Filter for Vision-Aided Inertial Navigation: This paper looks specifically at implementing EKF and SLAM for vision-aided navigation. Their approach (MSCKF) is able to determine geometric constraints for observed obstacles and features, without using 3D feature positions in its state vector [4]. This allows for a computationally efficient and accurate algorithm that was tested in a real-world setting by calculating feature pose estimates given while the camera was attached to a moving car. This approach could be ideal for implementing a computationally efficient SLAM algorithm that could be run alongside of the LB-WayPtNav in real world applications.

3) Keyframe-based Visual-Inertial SLAM Using Nonlinear Optimization: This paper demonstrates a SLAM algorithm (OKVIS) that integrates error terms from an IMU into visual SLAM to create a robust, non-linear optimization problem and avoids the need to include any tuning parameters. The tight coupling between the stereo camera and IMU sensor was shown to have more accuracy than methods just using a camera or a loosely coupled system [5]. This approach could be used in our application if the RGB camera or LiDAR sensor alone was not enough to create a map accurate enough for the narrow environments we hope to address.

III. Methods

A. Success Criteria

Our initial goal was to design an effective method to integrate SLAM into the original learning-based pipeline to enhance the capability of the navigation system in situations where the learning-based navigation system would fail on its own. More specifically, this would be cluttered and/or narrow areas where metric information would be crucial for navigation.

The success criteria our approach aimed to solve was showing that SLAM could successfully navigate to goals in scenarios where the original learning-based pipeline, LB-WayPtNav, would have failed. Specifically, these scenarios would be scenes that are cluttered and/or narrow.

B. High-Level Overview

Shown in Figure 1 is the figure of our proposed method.

On a high-level, we take the same LB-WayPtNav module from [1] and port over a separate SLAM module. Then we propose a module which switches between the two LB-WayPtNav and SLAM module once some criteria is met. In practice, we only implemented the SLAM module since there were software and driver compatibility issues which prevented us from being able to run and reproduce the LB-WayPtNav codebase. Note that in our setup, SLAM takes in LiDAR information, which is different from LB-WayPtNav, which takes in an image.

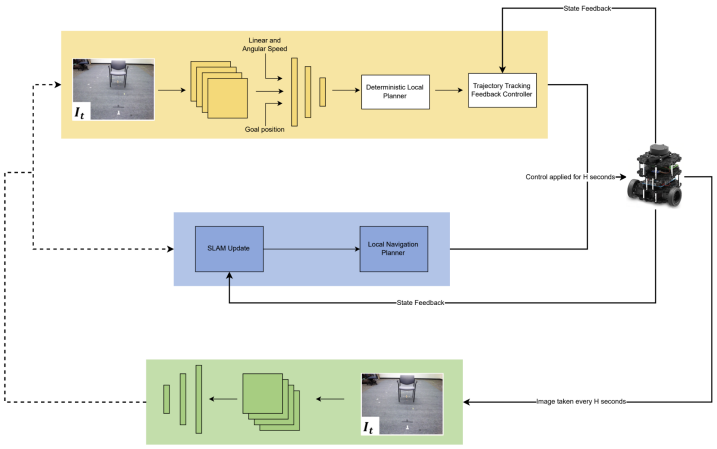

Fig. 1. Our Proposed Method

There were no specific equations which our method used.

C. Components of Proposed Method

The SLAM module uses the particle filtering pairing algorithm from ROS gmapping package. Particle filtering is another method of state estimation similar to Kalman Filtering and its variants. However, the advantage of particle filtering is that it is more applicable to a general array of situations and isn't limited to the case where the model is linear and the noise is Gaussian. Furthermore, particle filtering uses simulation and Monte Carlo methods to guess where the robot currently is. In practice, particle filtering tends to perform at least as well as Kalman Filtering. However, the cons of particle filtering are that, because it does not have a deterministic result like Kalman Filter's linear estimator, there is a chance that it will not terminate. In addition, the simulation aspect of the algorithm also means that there is a lot more computation that is required with this method, although in our experience on the Turtlebot, this did not seem to be an issue. For the navigation aspect of the SLAM module, we used the move base package from ROS. This package splits navigation into two components, a global planner and a local planner. Both take in the sensor measurements from the LaserScan and SLAM component to generate a costmap in the form of an occupancy grid. The global planner also takes in some global map generated by SLAM previously. Based on the costmap, the global planner plans a trajectory that would theoretically take the robot from its current state to the goal pose using a Dijkstra's based algorithm. This trajectory would be fed into the local planner for it to try to identify a set of velocity inputs that would allow the Turtlebot to path towards the goal. This local planner uses the dynamic window approach, which rolls out velocities for a set amount of time and picks the best option after sweeping through the possibilities. In the event that either planner is unable to figure out a trajectory, the robot will go into recovery mode and spin in place to try to update the costmaps and figure out if there is a feasible path to take.

Although the LB-WayPtNav module is not implemented in our experiments, we will describe these in more detail since it

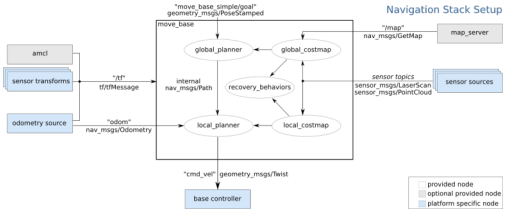

Fig. 2. Move Base Navigation

acts as a baseline again SLAM in our experiments. The LB-WayPtNav module is shown in more detail in Figure 3. The perception module takes in an image from an RGB camera attached to the Turtlebot and passes that information through its convolutional neural network along with its speed and goal position. The module outputs a predicted waypoint. This convolutional neural network is trained on ground truth data generated in simulation. A dynamics-based planning module converts this waypoint into a desired trajectory by fitting a cubic spline to it. A trajectory tracking feedback controller is then used to command the Turtlebot. After H time steps, the same process is repeated until the Turtlebot reaches its desired goal position.

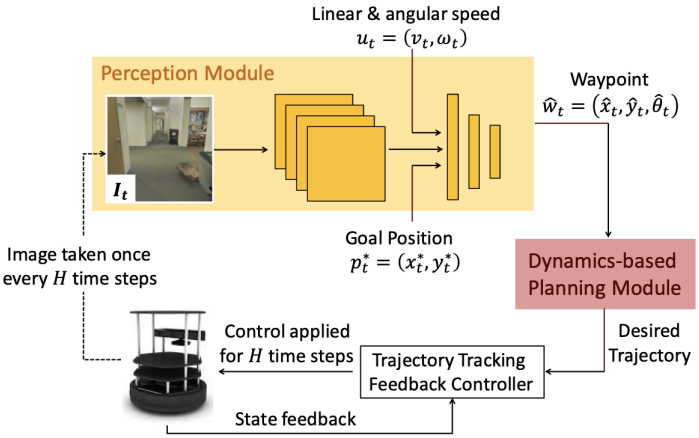

Fig. 3. LB-WayPtNav

IV. Results

A. Experimental Setup

Our initial hypothesis was that using SLAM would allow us to navigate through cluttered environments to a desired position in cases where LB-WayPtNav failed. The hardware we used was a Turtlebot with LiDAR sensors attached. We did not use any datasets. The independent variables are the LB-WayPtNav and SLAM methods. The dependent variables are the success rates of these two methods in various scenes.

B. Qualitative and Qualitative Results

In our experiments, we analyzed three situations where LB-WayPtNav failed and tested to see if the SLAM module could successfully navigate to the goal pose in these cases. Since we weren't able to run LB-WayPtNav for reasons descibed in our Conclusion section, we instead looked at failure cases of LB-WayPtNav as described in [6] and tried to reconstruct them in our lab room. We assume that LB-WayPtNav would fail in our reconstructed scenes and test SLAM on them.

The first example is when the Turtlebot is positioned close to an obstacle, such as the corner of a wall. In this case, the RGB image alone is not enough to have a good understanding of the obstacle's geometry, resulting in the learning module setting waypoints in the obstacle. On the other hand, SLAM is able to navigate around the obstacle to the goal in our reconstructed setting. The original failure case from [6] is shown in Figure 4 and our reconstruction of the failure case is shown in Figure 5.

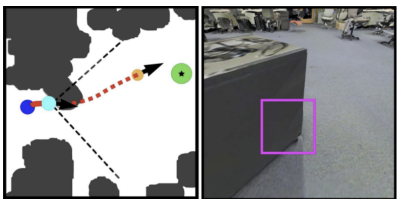

Fig. 4. LB-WayPtNav Failure Case 1

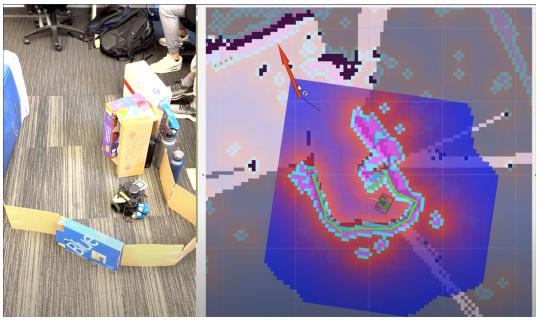

Fig. 5. Our Reconstruction of LB-WayPtNav Failure Case 1

The next failure case we analyzed was the situation when the Turtlebot is in a recessed region with a differently colored wall. In this case, the vision network will consider the wall as being an area that can be traversed and will plan waypoints through the wall. On the other hand, SLAM is able to navigate around the recessed region to the goal in our reconstructed setting. The original failure case from [6] is shown in Figure 6 and our reconstruction of the failure case is shown in Figure 7.

Finally, the third failure case we looked at was the failure of the vision network to discern whether or not a gap was large enough to move through. In all of these situations, we predicted that the metric information offered by SLAM would give the Turtlebot sufficient information to plan around the obstacles and successfully reach the goal state. On the other hand, SLAM is able to navigate through the gap to the goal in our reconstructed setting. The original failure case from [6] is shown in Figure 8 and our reconstruction of the failure case is shown in Figure 9.

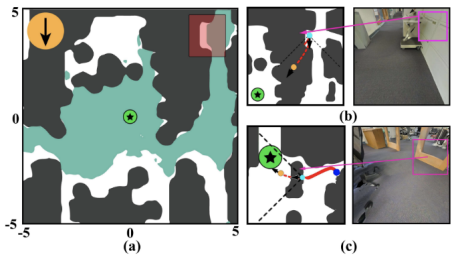

Fig. 6. LB-WayPtNav Failure Case 2

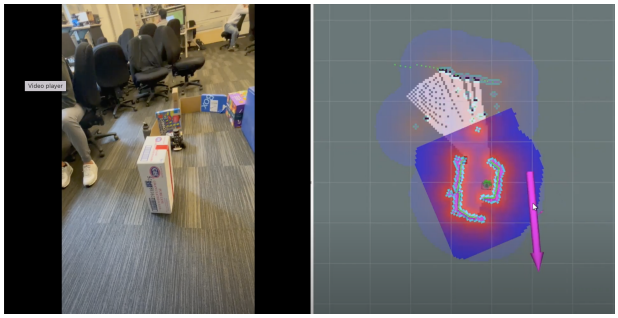

Fig. 7. Our Reconstruction of LB-WayPtNav Failure Case 2

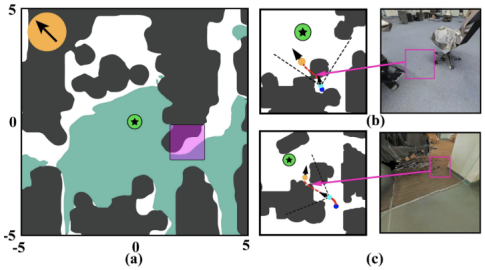

Fig. 8. LB-WayPtNav Failure Case 3

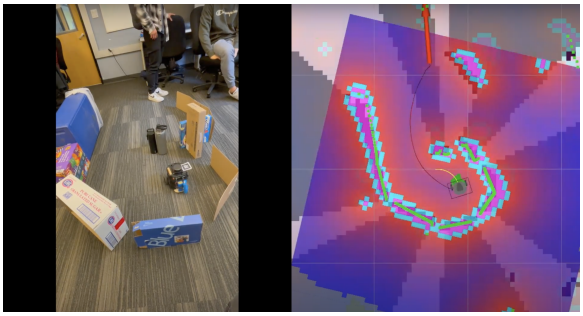

Fig. 9. Our Reconstruction of LB-WayPtNav Failure Case 3

We can summarize these results quantitatively in Table I. In all three scenarios, we see that SLAM can successfully navigate to the goal.

Goal-reaching success on various scenes		
Environment/Scene	LB-WayPtNav	SLAM
case 1: obstacle avoidance	Fail	Success
case 2: recession	Fail	Success
case 3: gap	Fail	Success

TABLE I

WE COMPARE LB-WAYPTNAV AND SLAM ON THREE ENVIRONMENTS WHERE LB-WAYPTNAV FAILS. IMPORTANTLY, NOTE THAT WE DON'T ACTUALLY RUN THE LB-WAYPTNAV BASELINE SINCE WE HAD TROUBLE REPRODUCING IT AS DESCRIBED IN THE CONCLUSION SECTION. RATHER, WE RECONSTRUCT THESE SCENES FROM FAILURE CASES OF LB-WAYPTNAV AS DESCRIBED IN [6].

C. Results and Implications

We found that, in alignment with our hypothesis, SLAM was able to successfully plan a path around the previously mentioned obstacles and failure cases. We also tested SLAM in two situations, one where SLAM had completely mapped out the environment and one where the robot was placed into the environment and made to navigate from there. We found that, in both situations, the robot is able to figure out what obstacles are around it and successfully generate a plan that finds a path to the goal pose. This tells us that SLAM would be able to give us the necessary metric information to find a path in a cluttered, novel environment. One important thing to note, however, is that the success of the planner relies heavily on the tuning of the navigation and SLAM parameters. For example, early on in our experiments, there would often be situations where, because of how the cost map and planning algorithm were weighted, the robot would dive straight towards the shortest path toward the goal pose, disregarding the location of any obstacles. As a result, the Turtlebot would find itself stuck in the corners, unable to recover.

This implies that when SLAM is combined with LB-WayPtNav with an appropriate switching mechanism, we can achieve better goal-reaching performance than just LB-WayPtNav without the computational burden of solely using SLAM. This would be interesting future work to tackle.

V. CONCLUSION/DISCUSSION

Our work suggests that integrating SLAM into the navigation pipeline can be beneficial to aid in cluttered and narrow environments where LB-WayPtNav has difficulty generating accurate waypoints. In each of the three test cases that failed within the simulation results of the reachability paper, we demonstrate that SLAM navigation would be able to navigate through the environment. In the future, these findings could be used to switch between SLAM and LB-WayPtNav within the navigation pipeline, as outlined in the methods section of this paper.

In attempting to implement the LB-WayPtNav to create the switching mechanism, we encountered multiple difficulties that hindered our progress. These difficulties stemmed from the fact that the data and codebase was several years old. One

difficulty was that the S3DIS dataset that the codebase used had been updated recently. As a result, the data format had changed since the codebase was released. As a result, the S3DIS dataset available for public download was no longer compatible with the older codebase. Another difficulty we ran into was that the codebase ran on Tensorflow V1. However, all of our machines had newer CUDA versions of 10.0 and thus ran into tensorflow bugs when trying to run the Tensorflow V1 code. However, even when running the code on CPU (where CUDA errors should have been avoided), the code errors in several places. Due to all these issues reproducing the LB-WayPtNav codebase and the limited time we had, we decided to redirect our efforts towards SLAM rather than reproducing LB-WayPtNav.

In addition to implementing the LB-WayPtNav with newer software and hardware, an intuitive next step would be to implement the switching mechanism using both SLAM and LB-WayPtNav in hardware. The SLAM algorithm in our experiments used particle-filtering based via ROS gmapping, however more updated SLAM algorithms could also improve the usability and efficiency of SLAM within the navigation pipeline. We also used LiDAR for SLAM in hardware, while the LB-WayPtNav approach uses an RGB camera to navigate and compute waypoints. This could pose a problem within hardware implementation in the future, since we would need to include both sensors on the Turtlebot to be able to use both SLAM and LB-WayPtNav. Options for a different SLAM algorithm are mentioned above in the related works section. One example of an updated SLAM algorithm that would be useful to solve both of these issues is DroidSLAM. DroidSLAM can operate with an RGB camera, and has been shown to be more accurate and robust than older SLAM approaches [2].

VI. Bibliography

References

[1] Bansal, Somil, et al. *Combining Optimal Control and Learning for Visual Navigation in Novel Environments*. Proceedings of the Conference on Robot Learning, PMLR 100:420-429, 2020.

[2] Z. Teed and J. Deng *DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras*. CoRR abs/2108.10869, 2021.

[3] Saxena, Amay, et al.*Simultaneous Localization and Mapping: Through the Lens of Nonlinear Optimization.* IEEE Robotics and Automation Letters 7.3:7148-7155, 2022.

[4] Mourikis, Anastasios I., and Stergios I. Roumeliotis. *A multi-state constraint Kalman filter for vision-aided inertial navigation* Proceedings 2007 IEEE international conference on robotics and automation. IEEE, 2007.

[5] Leutenegger, Stefan, et al.*Keyframe-based visual-inertial slam using nonlinear optimization* Proceedings of Robotis Science and Systems (RSS), 2013.

[6] Chakraborty and Bansal *Discovering Closed-Loop Failures of Vision-Based Controllers via Reachability Analysis* IEEE Robotics and Automation Letters 8.5 (2023): 2692-2699, 2023

VII. Appendix

A. Links

Links to Simulation and Hardware Experiments:https://drive.google.com/drive/folders/1sYMnWbgn0RWOuKT0wJclPHZZEAgqK7LG?usp=sharing