Turtle-SLAM: A Simultaneous Localization and Mapping Implementation and Analysis

Daniel Choate, Reed Bodley, Yifan Li

Abstract

This paper presents an implementation and analysis of 2D simultaneous localization and mapping (SLAM) using a particle filter based algorithm. The primary contribution is the development of a detailed 2D map of the third floor of the Joyce Cummings Center (JCC), which will serve as a foundation for future probabilistic robotics research at Tufts University. A secondary contribution is the creation of a streamlined procedure for utilizing the gmapping SLAM algorithm to generate maps of various environments using a Turtlebot, from small rooms to large floor plans. The accompanying tutorial is designed to guide students through the setup of the required ROS environment, remote control through SSH, and processing of the generated maps using RViz and post-processing algorithms. By teleoperating the Turtlebot and applying an optimized parameter set for gmapping, the method achieves consistent and reliable mapping results, even in the presence of sensor noise and environmental disturbances. This tutorial ensures that the mapping process is easily reproducible across different systems, providing a robust framework for future mapping tasks.

1 Introduction

Simultaneous Localization and Mapping (SLAM) plays an important role in enabling autonomous robots to explore and navigate unknown environments. Localization and mapping, as the two most critical components of SLAM, work synergistically to allow the robot to determine its position while simultaneously constructing an accurate representation of the environment. In this project, we implemented a 2D SLAM algorithm using a TurtleBot2 equipped with a LiDAR sensor, focusing on the Gmapping algorithm based on a particle filter for real-time map building and robot localization. Our work aims to address the challenge of mapping in unknown complex indoor environments with dynamic elements.

The Joyce Cummings Center (JCC) at Tufts University was chosen as the test environment, with a particular focus on the entire third floor of its public space for SLAM implementation. Our initial goal was to deploy Gmapping to generate a 2D map of the area using LiDAR data as input. Despite challenges such as moving objects and complex spatial layouts, we successfully created a

comprehensive map of the environment, demonstrating the ability of TurtleBot to adapt and navigate effectively. After obtaining the map, we applied image processing algorithms and manual techniques to enhance its appearance, ensuring a smoother and higher-quality result.

The main contributions of this project include:

- Implementation of a 2D SLAM system using the Gmapping algorithm on a TurtleBot2.
- Localization and mapping of the third floor of the Joyce Cummings Center in real time.
- Development of functional 2D maps that can serve as a basis for future probabilistic research and navigation tasks.
- A step by step tutorial of the ROS Turtlebot setup and teleoperation for future SLAM research.

This report provides a detailed overview of the project methodology, implementation process, challenges encountered, and the final performance of the SLAM system.

2 Background

One of the main problems surrounding the research motivation of simultaneous localization and mapping, or SLAM, is the kidnapped robot problem. This problem refers to a scenario where a robot is dropped in an unknown environment, specifically in a place where GPS is unavailable. The problem highlights the need for the robot to localize itself, as well as generate a map of its surroundings as the robot moves.

Several techniques and algorithms have been created with an aim to solve the kidnapped robot problem, such as feature recognition [6], camera based SLAM or vSLAM [12], as well as Lidar-based SLAM [7]. This investigation will focus on Lidar SLAM, specifically the well-established algorithm, Gmapping [1].

Light detection and ranging, or Lidar, data uses laser scanning to determine the range of an object [3]. With Lidar sensors spanning multiple elevation levels, the process of generating high resolution maps becomes efficient.

The Gmapping algorithm, developed by Grisetti, Stachniss, and Burgard, uses a particle filter based approach to estimate robot pose. A set of particles is drawn first by initial estimation of the robot's pose, and this estimate is improved by incorporating the most recent observations or inputs [1]. The particles are evaluated on how well they fit with the map obtained thus far, and new particles are estimated through importance weighting as well as adaptive resampling throughout the estimation process [1].

Shadows in LiDAR scans, caused by occluded or obstructed regions, pose challenges in environments where accurate mapping is essential. Traditional grid-based techniques often struggle with shadow effects, leading to data loss

or reduced mapping quality. McDermott and Rife [8] propose using spherical voxels instead of traditional grid-based methods to mitigate shadowing effects. Spherical voxels improve scan matching accuracy by capturing a more complete representation of the environment, particularly in cluttered or shadowed regions.

2D SLAM Algorithm Performance and Comparison

Additional research explores the performance of 2D SLAM algorithms like Gmapping, HectorSLAM, and Cartographer in various environments. Recent research has indicated that scan matching methods which use wheel odometry can be very reliant on the precision of the wheel compared to methods which use strictly particle filters [13] However, scan matching algorithms such as HectorSLAM have been seen to be less memory-intensive compared to particle filter-based algorithms like Gmapping [4]. Abdelrasoul et al. [1] analyze Gmapping in low-cost robots, emphasizing the trade-off between particle count and computational load. This study provides insights into optimizing Gmapping parameters for efficient occupancy grid mapping in resource-limited systems.

Li and Zhu [5] evaluate SLAM algorithms in simulated orchard environments, highlighting how rough terrain affects algorithm performance. These studies emphasize the importance of fine-tuning SLAM parameters and evaluating SLAM algorithms under specific environmental conditions, which will inform our project when comparing Gmapping and HectorSLAM.

Advanced Applications of Lidar SLAM

Sawada and Hirata [11] explore LiDAR SLAM on autonomous ships, demonstrating superior localization performance compared to traditional GPS-based methods, particularly under harsh environmental conditions. This study showcases the adaptability of LiDAR SLAM for challenging applications such as coastal or underwater mapping. In terms of busy environments, there has also been work to address pedestrian detection, integrating feature extraction and Kalman filtering for reliable tracking [2] Implementing such detection systems alongside SLAM algorithms can enhance mapping accuracy in dynamic settings. The work with Lidar map creation also extends to Neural Radiance Fields, where McDermott and Rife [9] introduce an innovative integration NeRF's with LiDAR mapping, formulating a probabilistic framework to address uncertainties in LiDAR data. Additionally, their probabilistic formulation could inspire future enhancements, such as adapting the model to 2D SLAM systems operating in noisy or ambiguous settings.

These studies collectively address challenges like shadow mitigation, parameter optimization, and pedestrian interference within SLAM frameworks. By focusing on LiDAR-based algorithms such as Gmapping, our project finally aims to draw an efficient real-time map in a dynamic indoor environment while exploring solutions like shadow-mitigating spherical voxels and integrating object recognition techniques.

3 Methodology and Technical Framework

The methodology for this investigation consists of three parts: the hardware and data setup, ROS framework, and the map post processing stage.

3.1 Hardware and Data Setup

For our experiment, we used a TurtleBot2 (codename Michelangelo) connected to the 'eve' computer via USB from the MuLIP laboratory. The computer operates on the Ubuntu 20.04 64-bit operating system, utilizing the x86_64 architecture, and is powered by an 11th Gen Intel® Core™ i7 processor..

The 'eve' computer is responsible for running all ROS and RVIZ programs, while our personal computer was used to SSH into 'eve' and control the robot's movements using 'teleop'.

The TurtleBot2 has the following specifications:

- Dimensions: 14 x 14 x 16.5 inches
- Weight: 13.9 lbs
- Base: Yujin Kobuki

The turtlebot gathers Lidar data using a Kinect sensor, which uses image data to simulate Lidar readings.

3.2 ROS Framework / Tutorial

As part of our project submission, our group has included a ROS gmapping tutorial which will be provided to the teaching laboratory for future robotics research. This tutorial is meant to act as a step-by-step guide for students of any level to utilize the turtlebots at Tufts University to create maps in areas of their choosing. The tutorial is included in appendix A.

Setting up the ROS environment consists of 5 main steps:

1. Initializing turtlebot connection
2. Teleoperation of turtlebot
3. Gmapping SLAM algorithm
4. RViz map creation and visuallization
5. Map saving/exporting

Using SSH terminal commands, our group was able to run the entire SLAM process on our personal computer, as well as teleoperating the turtlebot. To simplify project setup and development, our group utilized an environment docker, previously created by Brennan Miller-Klugman. This docker allows us to create a container with all of the necessary turtlebot dependencies, libraries, and configurations pre-installed. The docker provides an isolated environment for

running Turtlebot software, preventing conflicts with other libraries on the host system. Also, the docker can be easily shared and run on any system with docker installed. The steps to launch the docker are also included in Appendix A. Similarly, the docker ensures that the project will run in the same way regardless of which machine it is executed on.

An important aspect to note is the issue that deals with transferring displays from the 'eve' computer, to the personal computer being used for teleoperation and analysis. The normal SSH commands do not allow transfer of visual displays, so a special SSH command is required, as shown in the tutorial (Appendix A).

Gmapping, as explained in the bakcground, is a popular SLAM algorithm which uses a particle filter based approach to estimate the robot's pose and map the environment. A pseduocode for gmapping is shown in Appendix B.

Following setup and SSH commands, the docker environment, turtlebot connection, teleoperation, and RViz displays, the gmapping algorithm can be initiated. The ROS documentation for the gmapping algorithm can be found here. [10], initiated on the turtlebot with simple terminal commands. The tutorial in Appendix A initiates a gmapping demo, which utilizes all default values for the parameters below. Users have the option to change these parameters based on the specifications of the environment and processing core. Some of the optimized parameters for the gmapping algorithm used for our investigation are as follows:

- throttle_scans = 1 (Process 1 out of every this many scans)
- map_update_interval = 5.0 (How long (in seconds) between updates to the map. Lowering this number updates the occupancy grid more often, at the expense of greater computational load.)
- maxUrange = 80.0 (The maximum usable range of the laser)
- kernelSize = 1 (The kernel in which to look for a correspondence)
- lstep = 0.05 (The optimization step in translation)
- astep = 0.05 (The optimization step in rotation)
- iterations = 5 (The number of iterations of the scanmatcher)
- lskip = 0 (Number of beams to skip in each scan. Take only every (n+1)th laser ray for computing a match (0 = take all rays))
- linearUpdate = 1.0 (Process a scan each time the robot translates this far)
- angularUpdate = 0.5 (Process a scan each time the robot rotates this far)
- resampleThreshold = 0.5 (resampling threshold)
- particles = 30 (Number of particles in the filter)

- xmin = -100.0 (Initial map size (in metres))
- ymin = -100.0 (Initial map size (in metres))
- xmax = 100.0 (Initial map size (in metres))
- ymax = 100.0 (Initial map size (in metres))
- delta = 0.05 (Resolution of the map (in metres per occupancy grid block))
- transform_publish_period = 0.05 (How long (in seconds) between transform publications)
- occ_thresh = 0.25 (Threshold on gmapping's occupancy values)

Once the gmapping process has begun, the turtlebot may begin the navigation stage. To be discussed in the results section, there are many factors of teleoperation that have a direct impact on SLAM accuracy and results. Based on our experimentation, we recommend simple movements, separating translation and rotation (for example, translate forward 1 meter, followed by a 360 degree rotation, repeated through the entire scene). Similarly, speeding up the movements of the turtlebot teleoperation also has a direct impact on accuracy and results.

Following full navigation of the environment, the map must be saved and exported for post processing steps. This can also be done through the personal computer for simple exporting and saving, as indicated in the tutorial (Appendix A).

3.3 Post-Processing

Once the map is saved and exported onto the personal computer, the .pgm file must be put through a post processing stage. As initial results will show, many times the SLAM process is inherently noisy. The outlines of walls and windows cause issues within the SLAM process, as well as stray chairs, desks, and people moving throughout the scene. Our post processing algorithm consists of 3 main steps, and an optional 4th step:

1. Map pixel pre-processing
2. Gaussian Blur
3. Bilateral Filter
4. Threshold and Dilate (optional)

The necessary pre-installed packages include openCV and numpy. During the map pixel pre-processing stage, the user specifies the area of the .pgm file they would like to crop. During the SLAM process, the generated map may contain a high level of empty space, especially when the initial size of the map is much larger than the mapped environment. Following the preprocessing stage

is a Gaussian blur. During the Gaussian blur, the image is smoothened to reduce noise and small details. This blurs the entire image uniformly, without considering edges. The process assigns higher weights to pixels near the center and lower weights to pixels farther away, shown in the following equation,

$$I'(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(x+i, y+j) \cdot G(i,j,\sigma)$$

Where $I'(x,y)$ is the new pixel value at position (x,y), $I(x+i,y+j)$ is the original pixel value at position $(x+i,y+j)$, $G(i,j,\sigma)$ is the Gaussian kernel value at position (i,j) for a given standard deviation σ. This is also done with the 'GaussianBlur' function in openCV. Next, the bilateral filter sharpens critical transitions between occupied and free spaces after general smoothing from the Gaussian blur, shown in the following equation,

$$M'(x) = \frac{1}{W_p(x)} \sum_{x_i \in \mathcal{N}(x)} I(x_i) \cdot G_d(\|x - x_i\|) \cdot G_r(\|I(x) - I(x_i)\|)$$

Where:

$$G_d(\|x - x_i\|) = \exp\left(-\frac{\|x - x_i\|^2}{2\sigma_d^2}\right)$$

$$G_r(\|I(x) - I(x_i)\|) = \exp\left(-\frac{\|I(x) - I(x_i)\|^2}{2\sigma_r^2}\right)$$

$$W_p(x) = \sum_{x_i \in \mathcal{N}(x)} G_d(\|x - x_i\|) \cdot G_r(\|I(x) - I(x_i)\|)$$

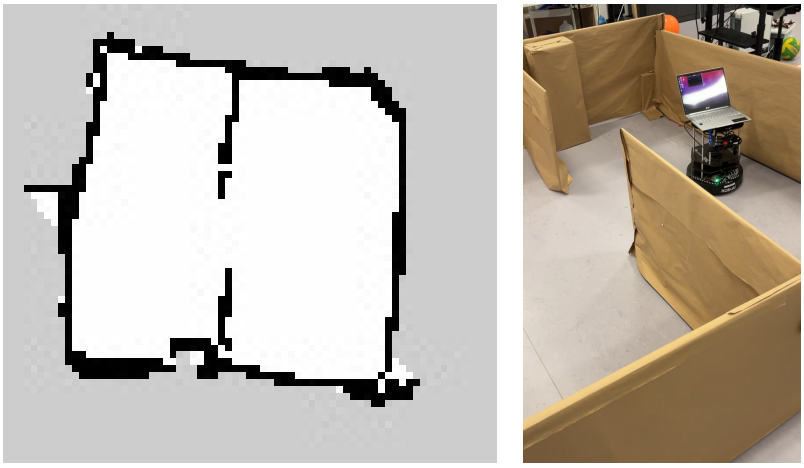
G_d represents a spatial Gaussian based on the euclidean distance between x and x_i, and G_r is the range Gaussian based on the intensity difference between pixels. W_p is the normalization factor, and σ controls the strength of the Gaussian weights. This is also implemented using the 'BilateralFilter' function in openCV. Optionally, the user can threshold the image so it is only in black and white, followed by dilating the image, which attempts to increase foreground in an image, and decrease the background. This can be done through openCV's 'Dilate' and 'Threshold' functions. Results of the post processing steps are shown in the results section.

Instead of processing the image using a blurring and filter process, we also investigated using a manual photo editor, Photopea. While this type of method can result in an extremely smooth map, accuracy can also be dependent on human error.

4 Experimental Results

4.1 Initial Testing

Once we implemented the gmapping algorithm on a turtlebot, we could begin mapping environments of our choosing. We began with the small cardboard outline in the middle of the MuLIP laboratory. The small size allowed us to see the precision of the SLAM-algorithm on a small scale. This initial environment also gives us the ability to fully control obstacles in the way of the turtlebot and the general shape and size of the walls was clear.



(a) Gmapping results for MuLIP carboard maze (b) MuLIP cardboard maze

Figure 1: Results of initial gmapping tests in the MuLIP laboratory

While this map contains outlines resembling the walls of the cardboard room, it was clear error and noise remained. It is important to note, however, that the small scale of the space does also make the imprecision and noise factors more apparent. We then moved to test a larger space, keeping in mind the need to eliminate noise. The next initial test that we took was in a computer lab on the second floor of the JCC, room 255. We chose this room for the simple rectangular outline as well as the rows of desks to test the algorithms ability to handle different objects. It took about 20 minutes to drive the robot around the edge of the computer lab and then through each of the rows of chairs before we felt that everything had been mapped to completion. The results are shown in Figure 2.

Compared to the first test on the cardboard room this map appears to have straighter walls and sharper corners, likely due to the larger room and the relative scale of noise. The rows of the chairs can be vaguely seen in this image as stray black dots in the middle of the room. Since these are not walls of the

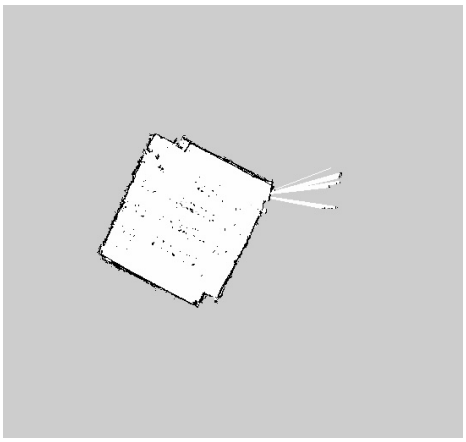


Figure 2: Computer lab room 225 created using SLAM

room, they would ideally be removed in the post-processing stage. An important discovery that we made during this test was that windows allow the Lidar to pass through without any disruption. The Lidar maps the outside world near the window instead of a flat wall. After mapping this computer lab, we elected for the first floor of JCC to maximize space and ground traffic. Driving the turtlebot around the first floor took roughly 45 minutes and the result can be seen in figure 3.

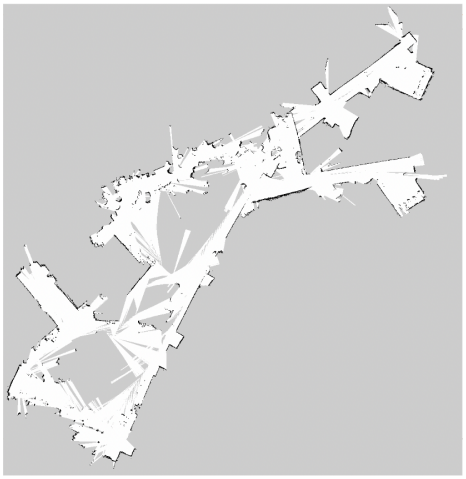


Figure 3: First floor of the JCC using SLAM

This test revealed more process issues to our attention. First of all, we missed several areas of the environment we assumed would be picked up. Another source of noise that we discovered was the importance of wheel odometry. There

were two separate instances during our mapping of the first floor where the turtlebot ran into an obstacle and could not be free with only teleop controls. As a result, we had to pick up the turtlebot and move it free from the object. Since wheel odometry is a vital input for the gmapping algorithm, this causes the robot localization to become incorrect. This is apparent to us in this image because when we arrived back at our starting position, it appeared in a different spot on the map than it was previously, which can be seen in the top right corner of our 1st floor map. The first floor also has many chairs, windows, and pedestrians, increasing noise and inaccuracies to be addressed.

4.2 Modified Experimentation

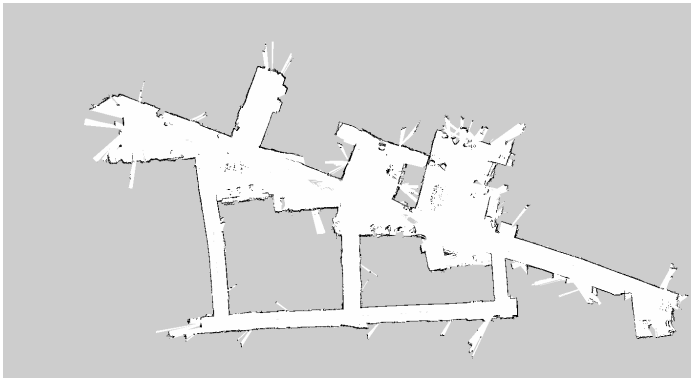


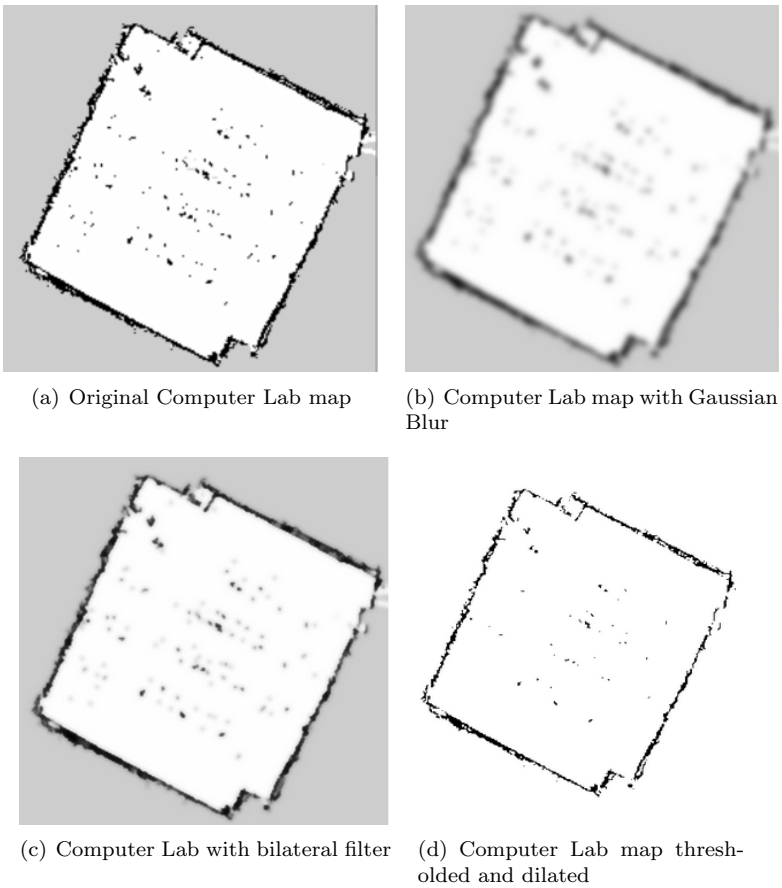
Figure 4: Third floor of the JCC created using modified SLAM processes

Following our initial testing, changes were made to the navigation strategy to eliminate some of the errors seen when mapping the first floor of the JCC. We first moved to the third floor, to work on a terrain with less traffic and obstacles. We altered the speed of the turtlebot to move slower than previously set, so as to allow for proper scan time and odometry readings. Similarly, we changed our movement strategy to allow for less pose estimation errors. Instead of constant movement and simultaneous rotation, we input a 'translate-then-rotate' strategy, where we would translate the turtlebot forward a certain amount, then rotate 360 degrees, which generate much better results for SLAM purposes, as shown in figure 4. Also, we ensured no odometry changes during the navigation experiment, to prevent any localization issues, which previously caused issues on the first floor.

Figure 4 shows the raw map generated following our modified navigation plan. As seen compared to figure 3, the results have improved significantly, with errors pertaining mainly to environmental geometry. Specifically, the number of third floor windows allow for the increase in mapped area outside the walls of the third floor. Also, many points in the middle of the map which seem like noise can be mainly attributed to desk and chair legs on the floor. Our

navigation path started in the top left of the map (the end of the hallway on the third floor) and looped around the whole floor ending at the bottom left of the map (near the collaboration rooms).

4.3 Post Processing



(a) Original Computer Lab map

(b) Computer Lab map with Gaussian Blur

(c) Computer Lab with bilateral filter

(d) Computer Lab map thresholded and dilated

Figure 5: Post processing results for computer lab map

As detailed in our methodology section, each map was put through a post-processing phase, to smooth noise generated by a combination of the Lidar scans and the Gmapping algorithm. Figure 5 shows the results of the Gaussian blur and bilateral filter process on the JCC computer lab room 225, as well as the optional thresholding and dilating stage. We can see a slight increase in the smoothness of the map, but still some remaining noise that does not fully represent the geometry of the true scene.

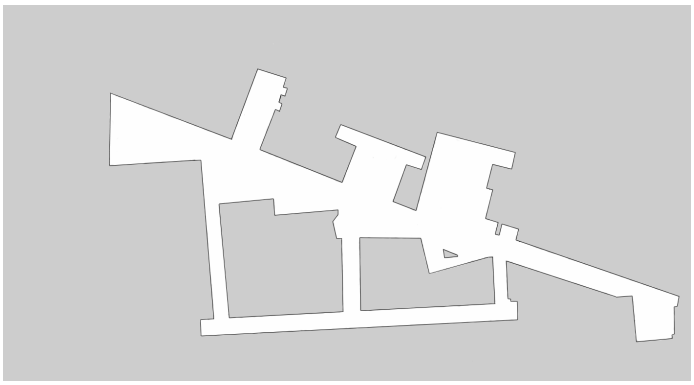


Figure 6: Third floor of the JCC created using modified SLAM processes

To further reduce noise, we employed the manual editing tool Photopea. This enabled us to replace the noisy walls with precise, well-detailed representations, as shown in Figure 6. While this method yields the cleanest results, it's important to acknowledge that it is prone to human error. For instance, the third-floor bathrooms, located near the bottom right, were removed due to lack of detail, leaving a smooth wall in their place. Despite this, the walls are highly accurate, and the layout of the JCC third floor and kitchen remains easily recognizable.

5 Conclusion and Future Work

Through a combination of image processing techniques such as Gaussian Blur, smoothing and dilating and manual editing, we successfully generated clean and smooth maps of the computer lab and the JCC third-floor public area. These methods effectively removed noise and refined the map boundaries, resulting in clear and accurate representations of the environments. The final maps demonstrate significant improvements over the original outputs, providing a solid foundation for reliable localization and navigation in SLAM applications. While the Gaussian Blur and smoothing process showed promising results in terms of reducing error, most accurate maps were presented after manual editing techniques.

There are many ways this project can be extended into future work. The first and most prevalent being a precise map of the first floor, which can be done utilizing our provided tutorial in Appendix A. A major piece of making our maps look clean for this project was done in the post processing phase, which for our best maps was using an online photo editor. Future work could potentially look at other Lidar-SLAM algorithms like Hector-SLAM, and map the same areas as before to compare the difference in accuracy and computational efficiency. We could also look at other SLAM algorithms process image data, or vSLAM. Another avenue for future work could be testing the localization of the turtlebot

with the existing maps to see if it could confirm the JCC third floor versus the fourth floor.

In summary, our group successfully produced a precise and refined map of the third floor of the Joyce Cummings Center, intended for use in future probabilistic research at Tufts University. Additionally, we have provided a comprehensive step-by-step tutorial on SLAM implementation and visualization, designed for the teaching lab to support students of all experience levels. This includes a script to aid in post-processing, reducing noise in the final maps. Looking forward, this work establishes a foundation for future developments in SLAM technology and its practical applications, advancing the broader field of robotics and autonomous systems.

References

[1] Yassin Abdelrasoul et al. A quantitative study of tuning ros gmapping parameters and their effect on performing indoor 2d slam. In *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, Sept 2016.

[2] J. Chen, P. Ye, and Z. Sun. Pedestrian detection and tracking based on 2d lidar. In *2019 6th International Conference on Systems and Informatics (ICSAI)*. IEEE, 2019.

[3] Misha Urooj Khan et al. A comparative survey of lidar-slam and lidar based sensor technologies. In *2021 Mohammad Ali Jinnah University International Conference on Computing (MAJICC)*, July 2021.

[4] Stefan Kohlbrecher et al. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, Nov 2011.

[5] Q. Li and H. Zhu. Performance evaluation of 2d lidar slam algorithms in simulated orchard environments. *Computers and Electronics in Agriculture*, 221:108994, 2024.

[6] A. Majdik, M. Popa, L. Tamas, I. Szoke, and G. Lazea. New approach in solving the kidnapped robot problem. In *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics), Munich, Germany*, pages 1–6, 2010.

[7] Patryk Mazurek and Tomasz Hachaj. Slam-or: Simultaneous localization, mapping and object recognition using video sensors data in open environments from the sparse points cloud. *Sensors*, 21(14):4734, 2021.

[8] M. McDermott and J. Rife. Mitigating shadows in lidar scan matching using spherical voxels. *IEEE Robotics and Automation Letters*, 7(4):12363–12370, Oct 2022.

[9] M. McDermott and J. Rife. A probabilistic formulation of lidar mapping with neural radiance fields. *arXiv*, 2024.

[10] ROS Wiki. gmapping, 2024. Accessed: 2024-10-18.

[11] R. Sawada and K. Hirata. Mapping and localization for autonomous ship using lidar slam on the sea. *J Mar Sci Technol*, 28:410–421, 2023.

[12] T. Taketomi, H. Uchiyama, and S. Ikeda. Visual slam algorithms: A survey from 2010 to 2016. *IPSJ T Comput Vis Appl*, 9(16), 2017.

[13] Qin Zou et al. A comparative analysis of lidar slam-based indoor navigation for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):6907–6921, 2022.

Appendix A: Turtlebot SLAM Tutorial

SSH into Eve Computer with Personal Computer (Terminal 1)

1. Make sure your personal computer is connected to the Tufts EECS WiFi.
2. SSH into the Eve computer:

```
ssh -X eve@10.5.12.45
Password: turtlebot112
```

3. Run the Docker command:

```
docker run -it --device=/dev/kobuki:/dev/kobuki --net=host -e DISPLAY \
-v /tmp/.X11-unix:/tmp/.X11-unix --device=/dev/bus/usb/ \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro turtlebot:latest /bin/bash
```

4. Initiate the Turtlebot connection:

```
roslaunch turtlebot_bringup minimal.launch
```

GMapping Demo (Terminal 2)

1. Open a new terminal.
2. Repeat the SSH steps.
3. Repeat the Docker command.
4. Launch the GMapping demo:

```
roslaunch turtlebot_navigation gmapping_demo.launch
```

Run Teleop (Terminal 3)

1. Open a new terminal.
2. Repeat the SSH steps.
3. Repeat the Docker command.
4. Launch the keyboard teleoperation:

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

Run RViz (Terminal 4)

1. Open a new terminal.
2. Repeat the SSH steps.
3. Navigate to the workspace:

```
cd catkinws/   (type "cd cat" and hit 'tab')
source devel/setup.bash
```

4. Inside RViz:
   - Click **Add** → **Robot Model**.
   - Click **Add** → **Map**.
   - Set the map topic to `map`.

Save Map (Terminal 5)

1. Repeat the SSH steps.
2. Save the map:

```
rosrun map_server map_saver -f /home/eve/Documents/NAME_OF_MAP
```

Move Map to Home Computer (Terminal 6)

1. Use SCP to copy the map to your personal computer:

```
scp "eve@10.5.12.45:/home/eve/Documents/NAME_OF_MAP.*" ~/Documents/
```

Appendix B: Gmapping Pseudocode

Algorithm 1 Gmapping SLAM Pseudocode

Initialize:
  Robot pose estimate (x, y, θ)
  Particle filter with N particles
  Empty 2D grid map (occupancy grid)
while Robot is operating **do**
  Read inputs:
    Laser scan data (ranges, angles)
    Odometry data $(\Delta x, \Delta y, \Delta\theta)$
  Update particles using motion model:
  for each particle **do**
    Predict new pose based on odometry $(\Delta x, \Delta y, \Delta\theta)$
    Add noise to account for motion uncertainty
  end for
  Compute particle weights:
  for each particle **do**
    Compare simulated laser scan from particle's pose against the real laser scan
    Calculate weight based on the match (higher weight = better match)
  end for
  Resample particles:
  Use the weights to resample particles (low-weight particles are replaced by high-weight particles)
  Update the map:
  For the best particle (or weighted average of all particles):
    Use laser scan data to update the 2D occupancy grid map
    Mark cells as "occupied" or "free" based on scan ranges
  Estimate robot pose:
  Compute the robot's pose as the mean or weighted average of particle poses
  Publish outputs:
    2D occupancy grid map
    Estimated robot pose
end while