CSci 5552 Term Project: Final Report
SLAM Navigation Robot

Fox, Kyle
foxxx551@umn.edu

Siew, Peng Mun
siewx007@umn.edu

Sun, Kerry
sunx0486@umn.edu

May 11, 2016

Abstract

An indoor Simultaneous Localization and Mapping (SLAM) algorithm was implemented for a Pioneer P3-DX Robot with a SICK laser scanner. The designed implementation navigated a square corridor while extracting line and landmark features from laser scan data using the Hough Transform. The Extended Kalman Filter was implemented to continuously estimate the robot's pose and its associated covariance using detected features. Results using propagation only, update with landmarks, and update with landmarks and Structural Compass were found and compared. The result using landmark and Structural Compass update together was found to deliver the best results.

1 Introduction

Autonomous motion for robots is essential for robotic applications nowadays. A robot has to know its position relative to its environment with sufficient accuracy to function safely. This problem is challenging because the robot does not have direct access to this information. Instead it must utilize uncertain measurements of its environment and its own behavior to estimate its own position. One method to address this problem is known as Simultaneous and Localization and Mapping, or SLAM. This method was developed by Hugh F. Durrant-Whyte and Smith and Cheeseman.[1][2] Durrant-Whyte showed that estimates of landmark positions were correlated with each other. This result was used to improve landmark position estimates. Smith and Cheeseman established the relationships between observed objects and previously measured positions using noisy measurements. Thrun et. Al introduced probabilistic localization and mapping methods to provide a reasonable solution to indoor SLAM.[3] These works as well as the Extended Kalman Filter (EKF) provide a foundation for this project. In this paper, we implement an indoor SLAM algorithm to navigate a Pioneer robot around a square corridor and conduct robot pose estimation. The final output is a map of the robot's surroundings based on the pose estimates and the obtained laser scans.

2 Design Approach

For this project, testing was conducted with a Pioneer P3-DX robot from Adept Mobilerobots. This robot comes equipped with a differential drive motion controller with 500-tick wheel encoders in its front two wheels. In addition, the Pioneer comes equipped with 8 forward facing sonar sensors which are used for obstacle avoidance. For detecting landmarks and mapping our environment, a SICK LMS200 laser scanner is mounted on top of the Pioneer. This scanner has a 180° field of view with half degree resolution and a maximum sensing range of 32 meters.

For communicating with the Pioneer and SICK, we used Mobilerobots' ARIA C++ library. The general structure of our code implementation is based around a navigation state machine written in C++ which periodically obtains odometry and laser scan data from the Pioneer and SICK. This state machine utilizes Matlab scripts through Mathworks' Matlab Engine API to interpret scan data and conduct SLAM propagation and update steps. The general flow of the code is shown in Figure 1.

After connecting to the Pioneer, SICK laser scanner and Matlab console, the code enters the event-driven finite state machine used for navigation. The purpose of the navigation state machine is to guide the robot in a loop around a square corridor while taking only left turns. In order to do this, the state machine consists of

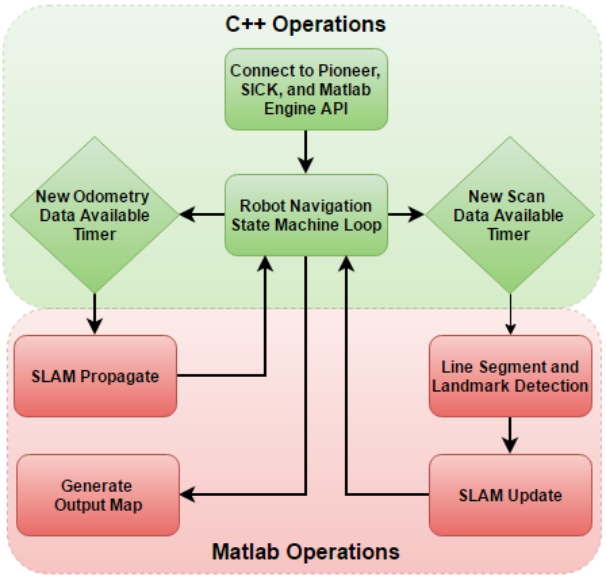

Figure 1: Code Structure Flow Diagram. Sections that are primarily implemented in C++ shown in green while red sections are implemented in Matlab

two primary states; go straight and turn left when able. While toggling between these states, two timers run in the background to determine when new odometry and laser scan data are available. The timer for new odometry data triggers every 30 milliseconds at which point the left and right wheel linear velocities are read to propagate the robot pose with the SLAM algorithm. The new laser scan timer triggers every 800 milliseconds at which point a line segment and landmark detection algorithm is run. This algorithm finds unique features in the environment using the distance and bearing measurements from the laser scan. The results of this algorithm are used by the navigation state machine to decide whether to transition to a new state, and by the SLAM algorithm to update the robot pose. The SLAM update step consists of an update step utilizing landmarks found in the environment along with a structural compass update based on the orientation of the robot to detected lines. At the completion of the code, the navigation state machine is turned off and a map of the robots path and its surroundings is generated.

Some of the assumptions that our design relied on are as follows.

1. At no point will an object suddenly block the robot's path. In this event the robot will cease all movements until the object is removed.
2. The path the robot is to navigate is perfectly square with 90 degree turns. While our design is robust for turns ranging approximately 45 to 135 degrees, it would require additional adjustments after the turn to re-orient itself. If the turn is outside of this range, the robot may be unable to orient itself to the left wall.
3. All turns on the robot's desired path will be the first left turn available which is large enough for the robot to fit into.
4. The desired robot path must always have a solid wall on the left side of the robot. This is due to the navigation algorithm relying on the orientation of the wall on the left to orient the robot.
5. All walls in the building the robot is to navigate in are at 90° angles. This is to accommodate the structural compass update step which checks for walls at the corresponding angles.
6. All scans taken are equally valid for generating the map and there are no moving objects along the robot's path. If objects do move in the robot's view, they will appear in the final map and will not be filtered out as a non-static feature.

3 Navigation Algorithm

The state machine developed to navigate a square corridor has two primary states which it toggles between. The "go straight" sequence of states is in effect whenever there is no opening detected to the left. When an opening is detected on the left, the state switches to the "turn left" sequence of states. Once the turn is completed, the state machine switches back to the "go straight" state. The "go straight" and "turn left" sequence have a series of sub-states that the robot will cycle through based on the information it receives from the odometry and laser scan data.

The primary "go straight" state sequence contains the following sub-states.

1. Go straight forward
2. Re-orient robot's heading with the wall on the left
3. Adjust robot's heading to the left

The "Go straight forward" sub-state is the predominant state in the "go straight" primary sequence.

While in this state, the robot will drive straightforward at a rate of 400 mm/s. While the robot is moving forward, it is constantly checking whether an object is within 400 mm of the front of the robot. If an object is detected, the robot slows its forward velocity to 100 mm/s. If an object is within 200 mm, the robot will attempt to turn away from the object if it isn't in front of the robot. If the robot cannot re-orient itself away from the object, it will cease to move until the object is removed.

The "Re-orient robot's heading with the wall on the left" sub-state is triggered whenever it is detected that the robot's heading is not parallel to the wall on its left. This is determined based on the results of the Line segment and Landmark Detection algorithm using the laser scan data. If it is determined the robot's heading is more than 2 degrees off, the robot will turn in the appropriate direction to keep its heading parallel to the wall.

Finally the "Adjust robot's heading to the left" state is triggered whenever it is detected that the robot has drifted more than 1.3 meters from the left wall. The "Re-orient robot's heading with the wall on the left" sub-state limits how often this state is entered as keeping the robot's heading parallel to the wall limits how far it will drift away from the wall. Occasionally however the 2 degree error in the previous sub-state causes the robot to drive too far away from the left wall. This sub-state simply makes the robot adjust its heading slightly to the left which keeps it from heading any further from the left wall. Staying relatively close to the left wall is important for the navigation algorithm to function correctly. This is due to the robot's heading being determined by the perceived orientation of the left wall, which becomes more uncertain the farther the robot is from the wall.

While the robot is in the "go straight" state, it is constantly checking the left most direction of the laser scan to see if the closest detected object is more than 5 meters away. If an opening is detected, a secondary script is run to check to see if the width of the opening is at least a meter wide so that the robot can fit plus a buffer. If the opening is determined to be large enough, the navigation state machine toggles to the "turn left" sequence. A depiction of the "go straight" sequence can be seen in Figure 2.

Once the navigation state machine has switched to the "turn left" sequence, the following sub-states are entered in the following order.

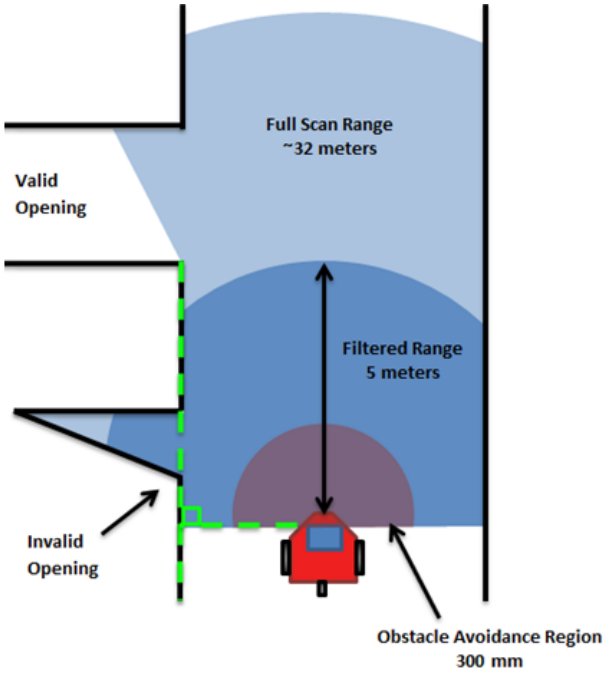

Figure 2: Depiction of the Navigation Algorithm as the robot is in the "go straight" state. While going forward the robot tries to keep the wall on its left parallel to its heading while checking to see if there is an opening to its left. If the opening is wide enough for the robot, the navigation state machine will toggle to the "turn left" state sequence.

1. Go straight pass corner
2. Stop robot
3. Turn left 90 degrees
4. Go straight until wall detected on left

In the "Go straight pass corner" sub-state the robot drives forward a set distance of 250 mm in order to ensure that the robot has passed the corner and has room to turn. The robot then enters the "Stop robot" state in which the robot stops in order to make the proceeding turn more consistent. The robot then turns left exactly 90° and enters the "Go straight until wall detected on left" state. While in this state the robot drives straight forward, checking to see when the corner of the wall is re-detected on its left side. Once the left wall is re-detected, the robot has completed its left turn and the navigation state machine toggles back to the "go straight" state. The robot then continues toggling between the "go straight" and "turn left" states resulting in a continuous navigation of the square corridor.

4 Line Segment and Landmark Detection

The purpose of the line detection algorithm is to detect important features in the robot's surroundings. From these features, unique landmarks can be determined which the robot can use for approximating its pose. The algorithm begins by creating a binary image file is generated using the laser scan data. Any laser distance readings which are greater than 5000mm are discarded. Multiple Hough transform iterations are then carried out to identify the most probable positions of lines based on the 2D point cloud. Hough transform implements a voting method in determining the most likely position of lines based on the number of points located at each possible lines. An example Hough table is shown in Figure 3, with the square boxes identifying peaks, which correspond to the most probable line structures within the laser scan.

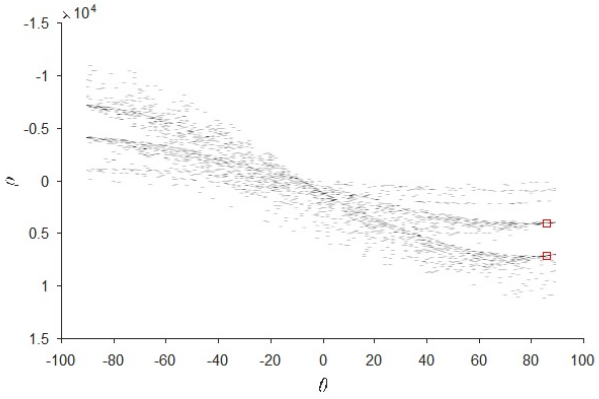

Figure 3: Visualization of Hough Table

In order to implement multiple iterations of the Hough transform within a reasonable time frame, the standard Matlab function for the Hough transform was modified. Changes to the original function included removal of unneeded error checking and parsing procedures. After making these changes, the computation time for a single Hough transform was reduced by a factor of 2.5 in comparison to the original Matlab function.

Instead of using a single Hough transform, three Hough transform iterations are performed, where the points associated with each identified line are removed before proceeding on to the next iteration. The lines extracted after three iterations are shown in Figure 4. The same line extraction performance might have been achieved by using a single Hough transform and doing more complex routine for line selection and outlier elimination. The lines obtained from using a single Hough transform with a lower minimum peak threshold is shown in Figure 5. As a design choice, the iterative Hough Transform process was used as it behaved more robustly in experimentation.

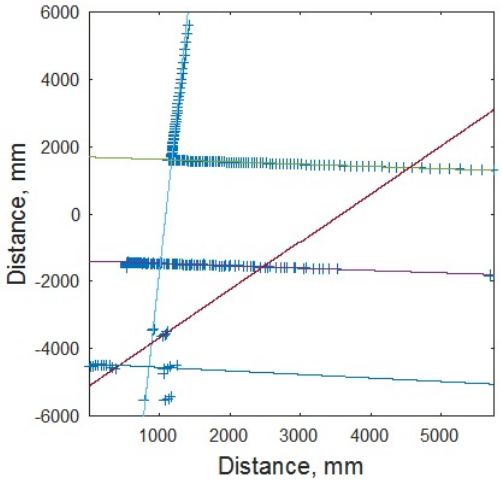

Figure 4: Iterative Hough Transform Result

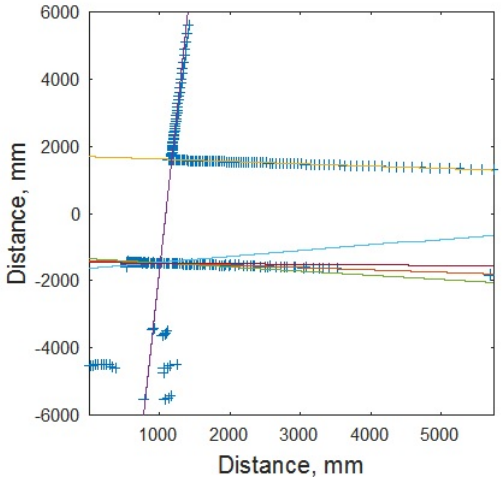

Figure 5: Single Iteration Hough Transform Result

Based on the line equations obtained from the Hough transform, points that have a minimum distance of less than 50mm between them are assumed to be part of a single line. Then lines with less than 30 points in them are discarded. This is done to filter out false lines.

After obtaining all possible lines, a least squares fitting method is implemented with the grouped line points to get a better line estimation.

The points located on each line are then grouped based on their relative distance to the neighbouring points. Any line segments with fewer than 10 points or with a length of less than 35mm are deemed to be inaccurate and are thus ignored. Based on the final groupings, the endpoints and line segments can be obtained.

The distance between each pair of lines' endpoints are then compared to determine if landmark exists at their intersection. If the relative distance between two endpoints are less than 350mm, it is safe to assume that a landmark is located within that region. The angle between the two line segments are then used to identify whether it is a corner or a door edge, where a different algorithm is used for the two features to determine the most probable position of the landmark.

Landmarks located at the leftmost and rightmost edge of the scan region are then discarded as this is the region where the laser scan is the least accurate. In addition landmarks located further than 4500mm from the robot are also discarded due to their lack of accuracy. The extracted landmarks with least squares fitting is shown in Figure 6.

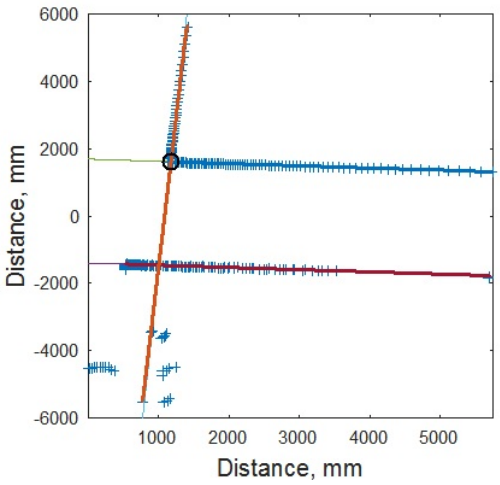

Figure 6: Detected lines with least squares fitting. Landmarks are indicated with circles

5 Simultaneous Localization and Mapping(SLAM)

The process of having a robot locate itself and simultaneously generate a map of its surroundings relies heavily on the Extended Kalman Filter. In addition and other algorithms such as Mahalanobis distance test and the Structural Compass technique. This process can be broken down into 3 parts as follows: Propagation, Landmark Update and Structural Compass Update.

5.1 Propagation

In the propagation step, only odometric measurements are used with the kinematic model of the robot. Specifically, the robot's linear velocity and angular velocity are calculated using the left and right wheel linear velocities. The uncertainty standard deviation of the robot's linear velocity was approximated to be 0.05 times the robot's estimated linear velocity. In addition, we found that the robot's rotational velocity tended to have larger uncertainties than the linear velocity. This was most dramatically seen whenever we were turning in place and monitoring the pre-programmed desired rotational velocity to the actual rotational velocity. As a result, we approximated the uncertainty standard deviation of the robot's angular velocity as 0.07 times the robot's estimated linear velocity. These were considered to be reasonable from experimenting with Pioneer robot.

5.2 Update

5.2.1 Landmark Update

During the update step there are 2 stages of updates: landmark update and structural compass update. First in order conduct the EKF propagate step however, we had to describe the uncertainty in the laser scan distance and angle measurements. In order to do this, we set the robot up to look at a flat wall and sampled 1000 sets of laser scan data. By then analyzing the measured distances at each angle across the 1000 scans, we could create simple characterization of the scanner's uncertainty. Since the scanner only returns distance measurements in a 180° arc with no measured angle to correspond to each distance measurement, we had to assign appropriate measurement angle bearings for each distance measurement ourselves in our code. As a result the measured distances combined the uncertainty in the measured distances and measured bearings to objects in the scanner's environment. From this test we found that approximating the bearing and distance uncertainty as 0.01 radians and meters respectively was reasonable.

Before the landmark update process is executed, each laser scan is analyzed to see if any landmarks

features were detected. If landmarks were detected, then the landmark update algorithm is executed. Otherwise state estimates and its covariance will be passed back to propagation step for the next iteration. In the landmark update algorithm, the Mahalanobis distance test is used to measure the distance between a point and a distribution. With this test we can check whether any measured landmarks are new landmarks or have previously been detected. By defining upper and lower γ bounds on the Mahalanobis distance calculated between the measured landmark and each landmark in the robot's state vector, we can test to see if two landmark states are relatively close together. In our case we assigned a lower γ bound of 4 and an upper γ bound of 50. This meant that our lower bound checked for cases when the measured landmark was not less than 0.0003% likely to be a previously measured landmark based on the chi-squared distribution. If the measured Mahalanobis distance is less than the lower γ bound, we decide that the measured landmark is match to the robot state vector landmark. In this case we conduct the standard measurement update procedure described by the EKF algorithm. If the Mahalanobis distance for a measured landmark exceeds the upper γ bound, then we designate the measured landmark as a new landmark. In this case the measured landmark state is inserted into the state vector, and its associated covariance is appended to the end of the old covariance matrix. In the event that the calculated Mahalanobis distance is between the upper and lower γ bounds, then no update is conducted, as it is uncertain whether the measured landmark is a new or previously found.

5.2.2 Structural Compass Update

In addition to landmark updates, if any line features were detected, a structural compass update is executed. The objective of Structural Compass update is to ensure the heading of the robot remains accurate. The structural Compass uses lines detected from the Hough Transform algorithm and assumes those lines correspond to wall features of the building. It was also assumed the walls were perfect "Manhattan" lines, which means they are all perpendicular or parallel to each other.

Similarly to the landmark update step, we had to define the relative uncertainty of the measured bearing to the line features for conducting the EKF update for the Structural Compass. The uncertainty of the bearings to the line features used for this update are based on the relative uncertainty of the individual distance and bearing measurements from the laser scanner. After grouping the points through the Hough transform and conducting least squares line fitting however, the resulting uncertainty of the line bearing constructed from these points should be less than the relative uncertainty from each individual measured point. As such we felt that in the worst case scenario, the bearing uncertainty should be the similar to the original uncertainty in the individual bearing and distance measurements. As such, our implementation utilized a relative uncertainty standard deviation of 0.01 radians to describe measured line bearing uncertainty.

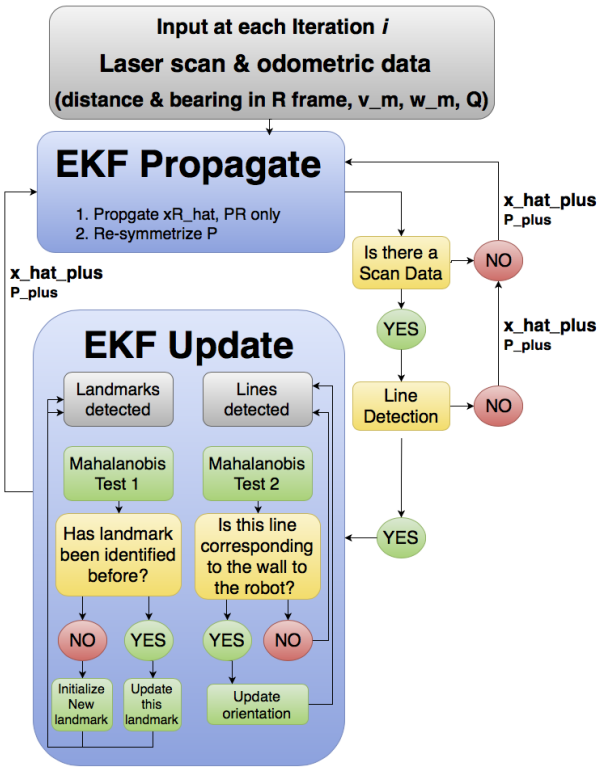

Figure 7: SLAM Algorithm.

When the robot first turns on it defines the corresponding 4 different angles to the normal faces of the building's walls based on the first wall it sees to its left. During each update, predicted angles to these initially defined wall is determined based on the robot's current orientation. By then comparing these predicted angle measurements to the actual angles to the measured line features' normal vectors', we can conduct a Mahalanobis test once again to find any matches. To detect if a measurement is a match we check to see if the Mahalanobis distance is less than a predefined γ bound.

In our implementation we set this γ bound to 500 as we made the initial assumption in our design that all of the walls the robot should see would be at 90° angles which make them all equally good for updating the robot's pose. If the distance is within this bound, the robot's heading is updated using the standard EKF measurement update equations. If the Mahalanobis distance exceeds the defined γ, no update is conducted as the wall does not match any of the pre-defined wall features.

Figure 7 depicts the flow of the SLAM algorithm used in this project as described above.

6 Results

To generate the final map of the robot's environment, the laser scans used during the EKF update steps are plotted at positions and orientations based on the estimated robot pose at that time step. In addition the error ellipses for a selection of robot poses is plotted using the state covariance and corresponding robot state estimate. Finally, the detected point landmarks were plotted based on the landmark states in the final state estimate vector from the SLAM algorithm. The SLAM results using the EKF propagation only are shown in Figure 8.

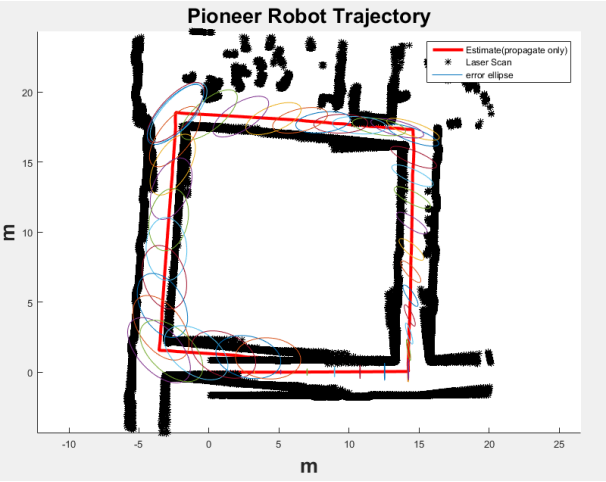

Figure 8: SLAM: Propagation Only.

One can see that the structure of the hallway was distorted due to the inaccurate robot bearing estimates. This is expected since there was no update step implemented which caused the relative uncertainty from the linearized propagation equations to accumulate over time. This can also be seen by the size of the 3σ bound ellipses continuously increases as the estimate is propagating forward.

The SLAM results using the EKF propagation and update with landmarks is shown in Figure 9.

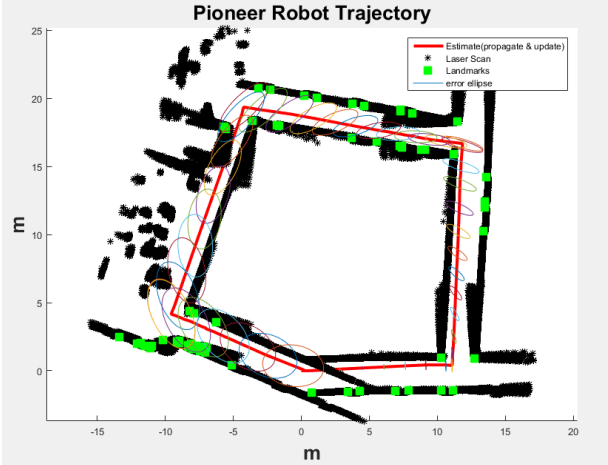

Figure 9: SLAM: Propagation and Update with Landmarks Only.

The path is more distorted than the EKF propagation only. Conventionally with the landmark update, we would expect it would produce a more accurate result than using the propagation only. However, inaccurate landmark detection and matching during the update step may have adversely effected the robot's estimated bearing. This theory is further supported by each corner in our map having multiple landmarks initiated at various coordinates around the true corner position instead of just one. This problem could potentially be solved by further adjusting the Mahalanobis Distance test thresholds and better characterization of the laser scanner's uncertainty.

Figure 10 shows the SLAM result using the propagation and update with landmarks and the Structural Compass.

The result turned out to be much better. Every corner appears to be a 90° right corner, which results in a rectangular shape for whole hallway. All the landmarks were detected at the correct place as expected. Figure 11 is a magnified plot of lower right corner of Figure 10.

One can see the $3-\sigma$ bound ellipse starts shrinking when the robot completes its loop and begins detecting landmarks it had previously discovered.

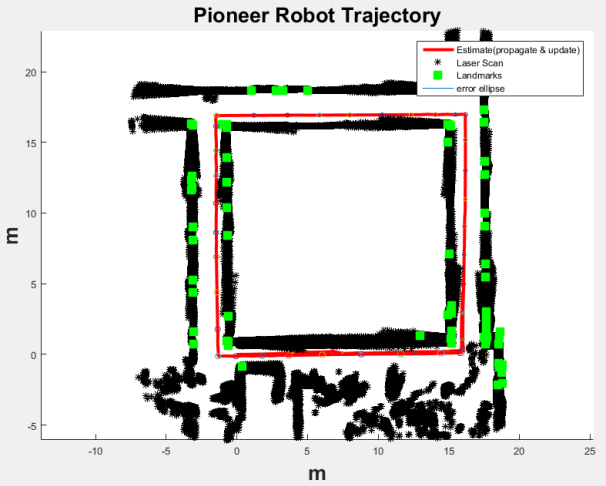

Figure 10: SLAM: Propagation and Update with Landmarks and Structural Compass.

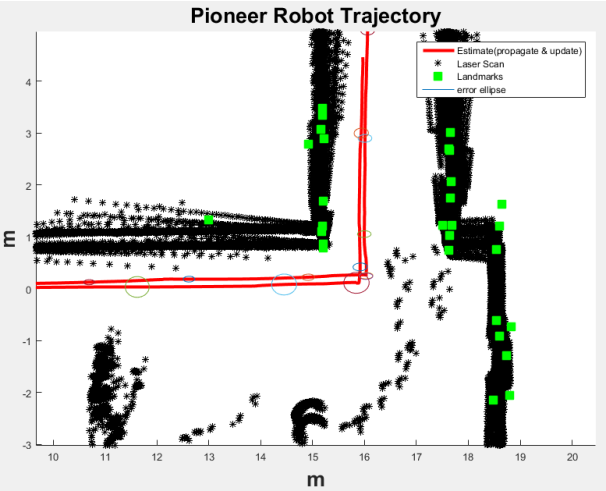

Figure 11: Magnified Plot of lower right corner of structural compass based map when the robot first rediscovers the landmarks from the first loop.

7 Conclusion

An implementation of the SLAM algorithm was successfully implemented on the Pioneer Robot with a SICK laser scanner. The robot was able to navigate around a square corridor while avoiding obstacles. The robot's position was estimated through odometric measurements of the robot's left and right wheel velocities and the robot's kinematics. In addition measurements of unique corner landmarks were determined using a line detection algorithm based on bearing and distance measurements. Using these detected landmarks and line features, the robot's pose was updated through the EKF algorithm. By navigating the robot autonomously around a square corridor, we were able to generate a map and localize the robot in real time. It was found that utilizing both landmarks and Structural Compass line features gave the best results. The final generated map had all of its corners appear at 90° angles and all the landmarks were detected without any issue. Future work would include more accurate characterizations of the laser sensor and robot odometric noise. In addition better characterization of the relative uncertainty of the derived bearing measurements to detected line segments would further improve the Structural Compass updates. Finally, since our current implementation is a single threaded application, both navigation and data processing algorithms cannot run simultaneously. As a result, this implementation is limited as to how fast propagation and update steps can be processed while still maintaining adequate robot navigation control. Future work would include converting this implementation to a multi-threaded application which separates the navigation and data processing into two separate threads. This would allow for faster propagation and update steps which would further reduce our system's uncertainty.

References

[1] Hugh Durrant-Whyte. Localisation, mapping and the simultaneous localisation and mapping (slam) problem. *SLAM Summer School*, 2002.

[2] Randall Smith, Matthew Self, and Peter Cheeseman. *Autonomous Robot Vehicles*, chapter Estimating Uncertain Spatial Relationships in Robotics, pages 167–193. Springer New York, New York, NY, 1990.

[3] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. A probabilistic approach to concurrent mapping and localization for mobile robots. *Autonomous Robots*, 5(3-4):253–271, 1998.