CSci 5552
Project - Sixth Sense
Report

Yashasvi Sriram Patkuri, patku001@umn.edu
Prashanth Kurella, kurel002@umn.edu
Mandakinee Singh Patel, patel908@umn.edu
Abhinav Mehta, mehta275@umn.edu

May 12, 2020

Robot: An intelligent machine that can sense and act on the world.

1 Introduction

Given a differential drive robot with a laser scanner in an static unknown 2D environment, the goal is to

1. Create a map of the obstacles in the environment leaving no area unexplored.
2. Simultaneously localize and map the landmarks in the environment.
3. Not collide with any obstacles while doing so.

The aspects of the project span a minimal set of features required for sensing in a robot. The main challenges are,
Correctness: Processing sensory data correctly and handling corner cases
Performance: Processing it fast enough for it to be real-time
Safety: Minimizing probability of a collision in uncertainty
Measurement noise: Making reliable lifelong improving maps in uncertainty

2 Related work

Nunez et al.[7] have laid out the basic steps for identification of landmarks from measurements. Borges et al.[2] have set up the concept of extracting lines from 2D laser range scanner measurements. We base our landmark detection frame work on on the foundations laid by these two papers.

3 Simulator

Instead of using ROS/Gazebo (which is heavily bloated for the purpose) or simulator in the class repository, we use a self-written simulator in 'Kotlin'. It started out a port of the latter and therefore it has basic necessities like separate thread for robot, noise in measurements etc... We use 'Processing' as the rendering library, which is essentially a thin wrapper around OpenGL. This lets us easily make good visualizations quickly while abstracting the unnecessarily complex OpenGL specifics. We use 'Kotlin' because it is terse, statically-typed and fast enough for the purpose. We use 'EJML' as the linear algebra library.

This in-itself was a challenging task and took significant amount of time. It provided insights in the importance of a single clock, noise addition, control integration, laser sensor distance calculations, thread management and an overall feel for the system. This made debugging and visualizing rather easy. Although we wrote our simulator we make sure that we do not access any state that would otherwise be unavailable in real-world viz. true state of robot, propagation and measurement noise mean and co-variance etc...

4 Our Approach

Here, we layout the details of all the steps involved in achieving the goals of this project.

4.1 Estimating Noise Co-variance

We assume that the propagation and measurement noise has zero mean but is initially unknown. Therefore, we fit a Gaussian curve on the observed noise in each quantity. Since the Gaussian fit will have 0 mean, we just need to estimate the co-variance. For this, we give a control for the robot for some time and get its true pose and compare it with estimated pose to get the propagation noise.

$$n_{prop} = {}^{G}P_{R} - {}^{G}P_{Rt} \tag{1}$$

Similarly we compare the true and estimated landmark position to get the measurement noise.

$$n_{m} = z_{t} - C^{T}({}^{G}\Theta_{R})({}^{G}P_{L_i} - {}^{G}P_{R}) \tag{2}$$

Note that in this step we assume that we know the true pose of robot and landmark positions. The equivalent of this in real world is just to use a measuring tape to get them. The movement is generated by giving random linear and angular velocities. Angular velocity is always kept positive so that the robot only takes full ellipses and doesn't move completely outside the scene. The noise is collected at regular intervals of the movement and once enough samples have been collected we fit a normal distribution to both positional and laser measurement noise. The covariance is calculated using the formula,

$$cov_{x,y} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{N-1} \tag{3}$$

In our experiments, we collected around 100 noise samples, on a regular interval of 500 milliseconds. Figure 1 shows the results of the Gaussian fitting over the measurement noise. Note that it's a loose fit primarily because the mean is fixed at 0,0 and also because we want to keep scope for larger noise. It's better to have more uncertainty than to have less and then crash.

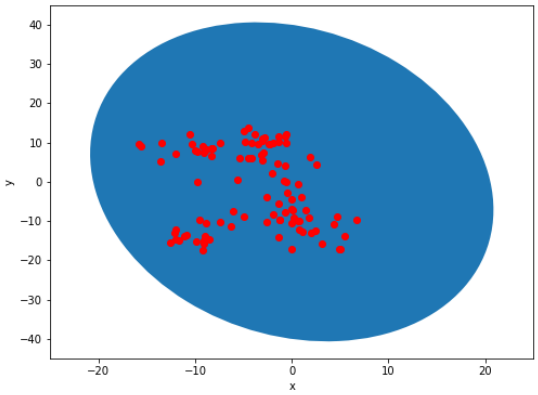

Figure 1: Covariance Estimation. Red points are the measurement noise. Blue region is the 95% confidence interval of the Gaussian Distribution.

4.2 Obstacle and Landmark Detection

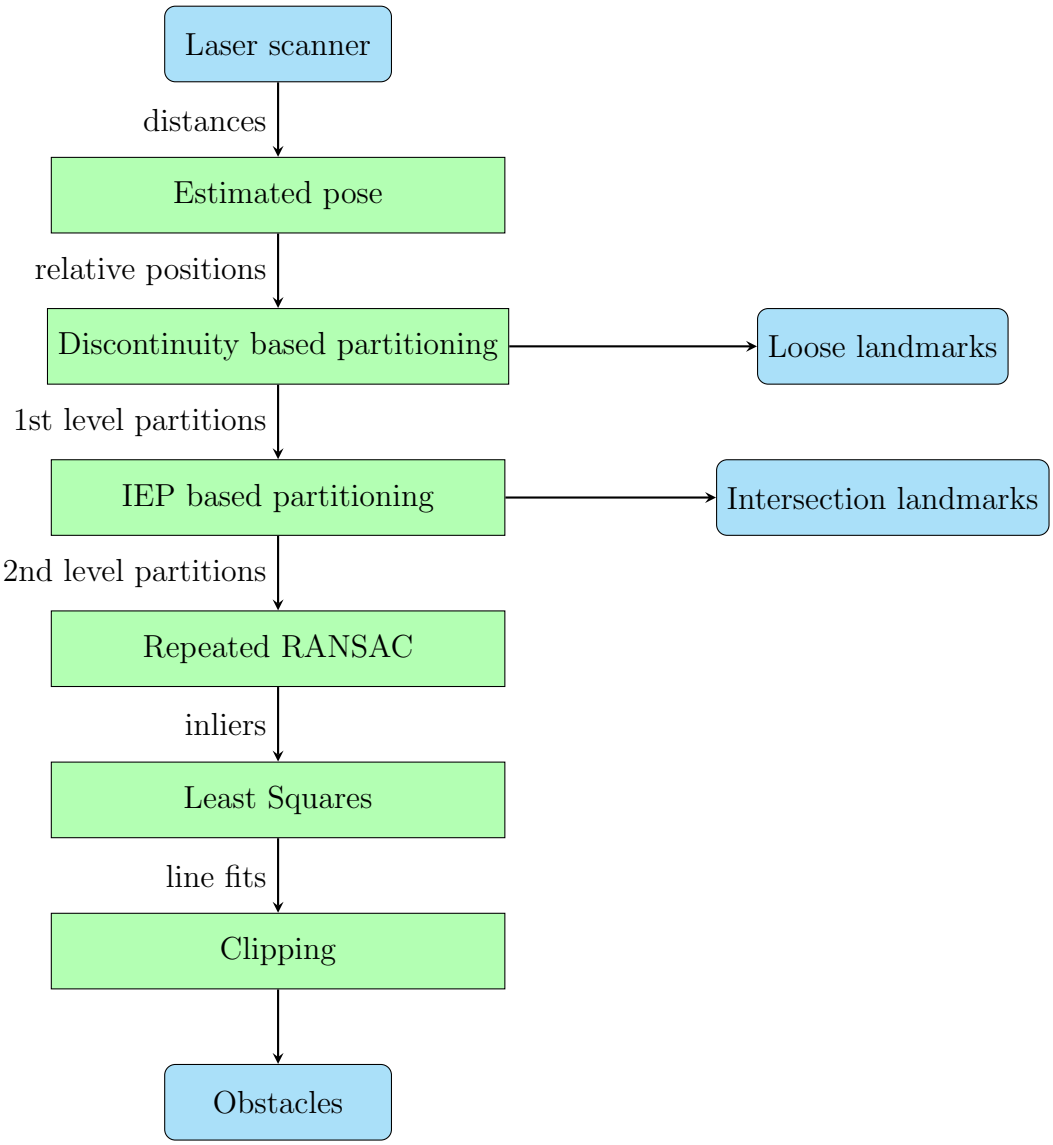

Figure 2: Pipeline of obstacle and landmark detection

Figure 2 shows the pipeline of obstacle and landmark detection.

Obstacle vs Landmark An obstacle is a an object in space that the robot likes to avoid colliding with. A landmark is a point in space that is uniquely identifiable either by a direct or an inferred measurement. A laser scanner returns a list of distances on each measurement. A wall is modelled as a line segment. Depending on its resolution, this can be up to a few hundred distances. Given an estimate pose of the robot each distance maps to a point in space. But not all of them are uniquely identifiable in this setting. The ones that are identifiable through this particular sensor are the intersections of walls and loose ends of walls. Therefore we define only such points to be our landmarks, which we eventually store

in our SLAM state. Nevertheless we use other points to estimate and keep track of wall poses. This is an important design decision.

Partitioning using Discontinuity We first partition the list of distances based on discontinuities. This lets us identify different groups of walls as a first pre-processing step.

$$|d_i - d_{i+1}| > \varepsilon \tag{4}$$

Equation 4 describes the criterion for identification of discontinuity, where d_i and d_{i+1} are two consecutive distance measurements, and ε is tuning parameter which is determined based upon the noise levels in measurements, and the environment. Upon detection of N discontinuities, the measurements are split into N+1 partitions of distances. This is illustrated in Figure 3. A subtlety is that laser sensors have a finite range and can return invalid values when a laser does not hit anything in that range. This has to be dealt with properly. Since distances are consecutively taken with a small increment in angle of the laser, each partition corresponds to a group of walls which shall be further divided in next step.

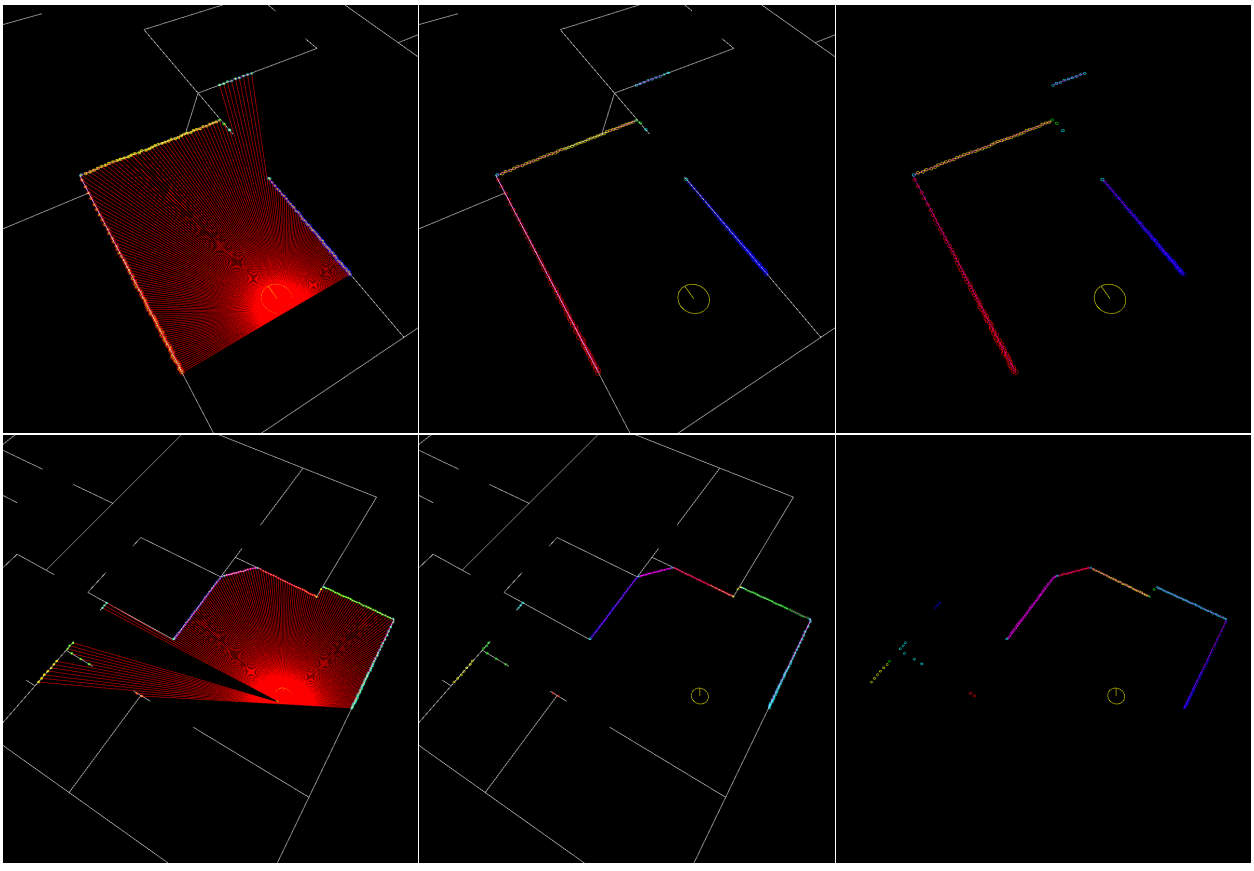

Figure 3: Partitions based on discontinuity and iterative end point method. A set of points of same color represent a partition

Partitioning using IEP If RANSAC were to be performed such partition the threshold value need to be big which in-turn increases the uncertainty of the wall poses. We leverage the constraint that each partition can correspond to a group of walls connected serially. We use *Iterative End Point Fitting* [10] to try to find as many end points of such serially connected walls. Using these endpoint we further partition each partition into second level partitions. This significantly reduces the uncertainty and increases the stability of identification of the individual walls and their intersections. This is illustrated in Figure 3. The pseudo-code for the algorithm is presented in Algorithm 1.

Line fitting using RANSAC and Least Squares We have a set of partitions after second level partitioning using IEP. For each partition we perform RANSAC [5] to fit a line. With the remaining points in the partition we fit another line and repeat this until there are no more enough points left. Each partition thus yields a set of lines each having a set of points as inliers. We use least squares line fitting [1] to improve the fit given by RANSAC. The advantage of IEP is further illustrated here because most of the second level partitions shall yield a single line, thus the RANSAC threshold can be quite low. The order of distances in each partition is still preserved. Therefore each line output by least squares fitting is clipped using projection of end points of each partition on the line itself. This is illustrated by the pink lines in the Figure 4.

Landmark Detection The landmarks are defined as loose ends of walls or intersection of walls. The loose ends are found just after first level partitioning using discontinuity. However there are some subtleties. An end point of first level partition is a landmark unless until

1. It corresponds to the first/last laser of sensor, since we can not sense beyond these.
2. It's corresponding distance is very close maximum laser distance.
3. If the distance corresponding to neighbour laser not in partition is less than its own.
4. The number of points in the partition it is less than a minimum threshold.

Iterative end point method returns end points of serially connected walls. These points excluding the first and last are counted as landmarks unless, the wall corresponding to them is too small or adjacent walls ends are more than a certain threshold apart. These are illustrated by the cyan circles in Figure 4

4.3 Simultaneous Localization and Mapping

We assume noise in the control and measurements. If this noise is left unchecked, it will accumulate leading to a large gap b/w estimated and true states. 'Simultaneous localization and mapping' [4][9] using Kalman Filter is a way to simultaneously estimate both robot pose and landmark positions. It has three steps namely propagation, augment and update.

Measurement model The measurements we receive from the laser scanner are defined by the Equation 5.

$$z = C^T({}^G\Theta_R)({}^G P_{L_i} - {}^G P_R) + m \tag{5}$$

Algorithm 1: Iterative End Point Fitting

```
Function IterativeEndPointFitting(PointList[], ε):
    dmax ← 0
    index ← 0
    end ← length(PointList)
    i ← 2
    while i < end do
        d ← perpendicularDist(PointList[i], Line(PointList[i], PointList[end]))
        if d > dmax then
            index ← i
            dmax ← d
        end
        i++
    end
    ResultList ← []
    if dmax > ε then
        recResults1[] ←IterativeEndPointFitting(PointList[1:index], ε)
        recResults2[] ← IterativeEndPointFitting(PointList[index:end], ε)
        ResultList[] ← {recResults1[1:-1],recResults2[1:]}
    else
        ResultList[] ← {PointList[1], PointList[end]}
    end
return ResultList
```

Propagate step We use the fourth order RK-4 [8] approximation of differential drive dynamics to propagate the robot's state through time. This proves to be good enough for our purpose. Equations 6-9 describe the slam propagate step mathematically, here $g(x(t), u)$ refers to the mathematical model for differential drive dynamics. $x(t)$ and Σ_x refer to the mean and co-variance of the distribution that contains the robot's position.

$$g(x(t), u) = \begin{bmatrix} (\nu + \eta_\nu)cos(\theta_t) \\ (\nu + \eta_\nu)sin(\theta_t) \\ (\omega + \eta_\omega) \end{bmatrix} \tag{6}$$

$$k = \begin{bmatrix} g(x(t), u) \\ g(x_2, u) \\ g(x_3, u) \\ g(x_4, u) \end{bmatrix} \quad x = \begin{bmatrix} x(t) \\ x(t) + \frac{\Delta_t}{2} k_1 \\ x(t) + \frac{\Delta_t}{2} k_2 \\ x(t) + \Delta_t k_3 \end{bmatrix} \tag{7}$$

$$\begin{gathered} x(t + \Delta_t) = x(t) + \frac{\Delta_t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ \Sigma_x(t + \Delta_t) = \Sigma_x - \Sigma_x H^T S^{-1} H \Sigma_x \end{gathered} \tag{8}$$

$$\begin{gathered} S = H \Sigma_x H^T + M \Sigma_m M^T \\ K = \Sigma_x H^T S^{-1} \end{gathered} \tag{9}$$

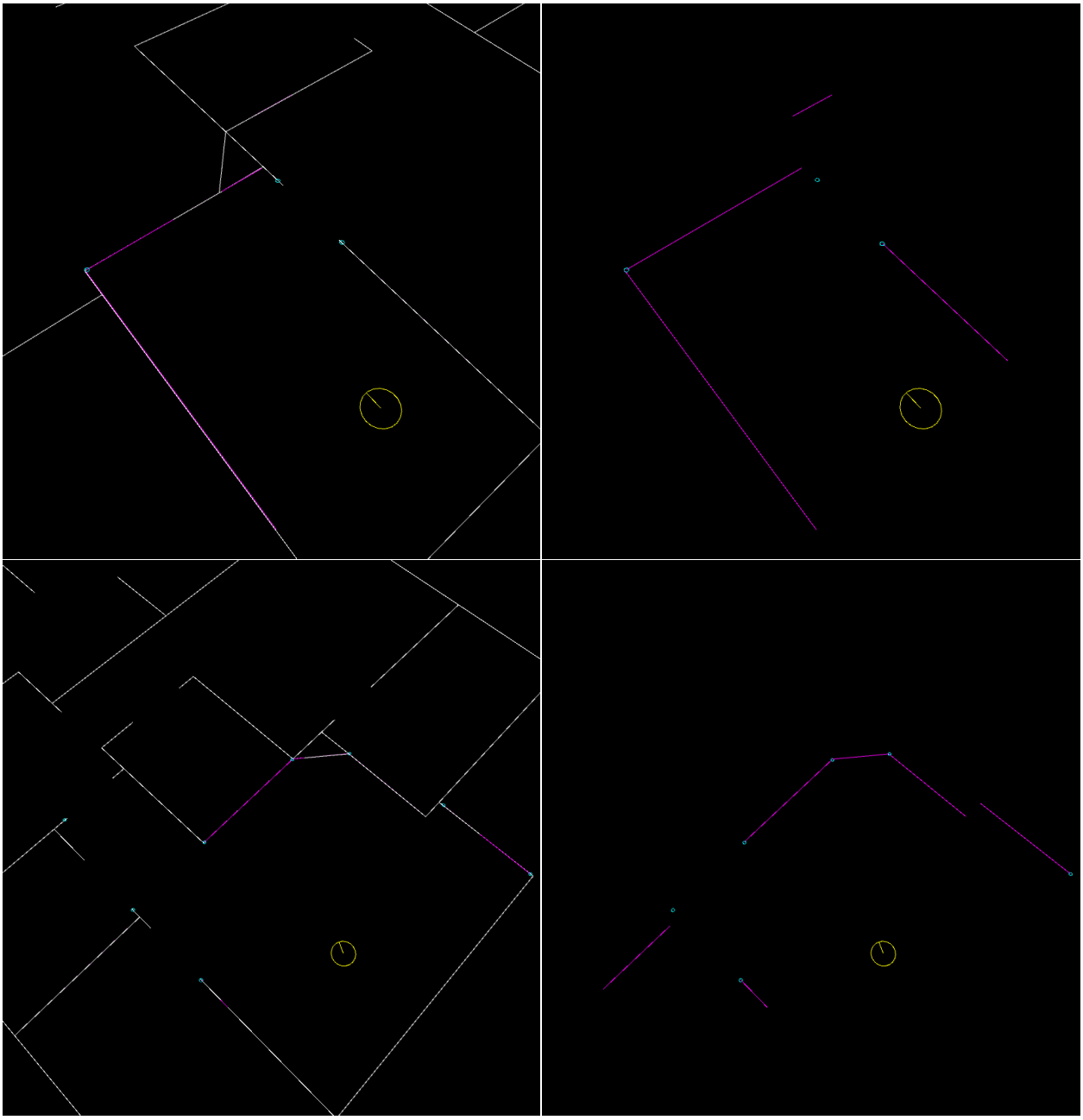

Figure 4: Obstacle and landmark detection. Pink lines are detected obstacles, Cyan circles are detected landmarks

Augment and Update step We use the Mahalanobis Distance for each measurement as a metric of novelty. We consider a measurement to have matched with an existing landmark if their Mahalanobis distance is less than 20 units, augment the state with a new measurement if it's Mahalanobis distance to any other measurement is greater 200 units. Any measurements that do not conform to either category are discarded. Augments are done only after all

updates are performed for a batch of measurements.

$$\begin{aligned} \hat{x}^{+}(t) &= \hat{x}(t) + K(Z - h(\hat{x}, 0)) \\ \Sigma^{+}_{x(t)} &= (I - KH)\Sigma_{x(t)}(I - KH)^T + KM\Sigma m(t)M^T K^T \end{aligned} \tag{10}$$

Periodic Bad Landmark Culling The landmark detection algorithm is designed to only detect intersections of walls and loose ends of walls. However a few measurements that are neither slip past over time due to manual threshold. There measurements are added to the state. They are not useful and increase the size of estimate of the state and co-variance unnecessarily which slows down computation. To mitigate this we periodically check for bad landmarks and remove their blocks from state estimate and co-variance. For each landmark the number of updates it receives is kept track of. A landmark is considered bad if its updates are less than a threshold. This makes it harder for the state from becoming intractably large.

The SLAM state is illustrated in Figure 5. The uncertainty in the estimates is quite small because they are measured at the start of simulation.

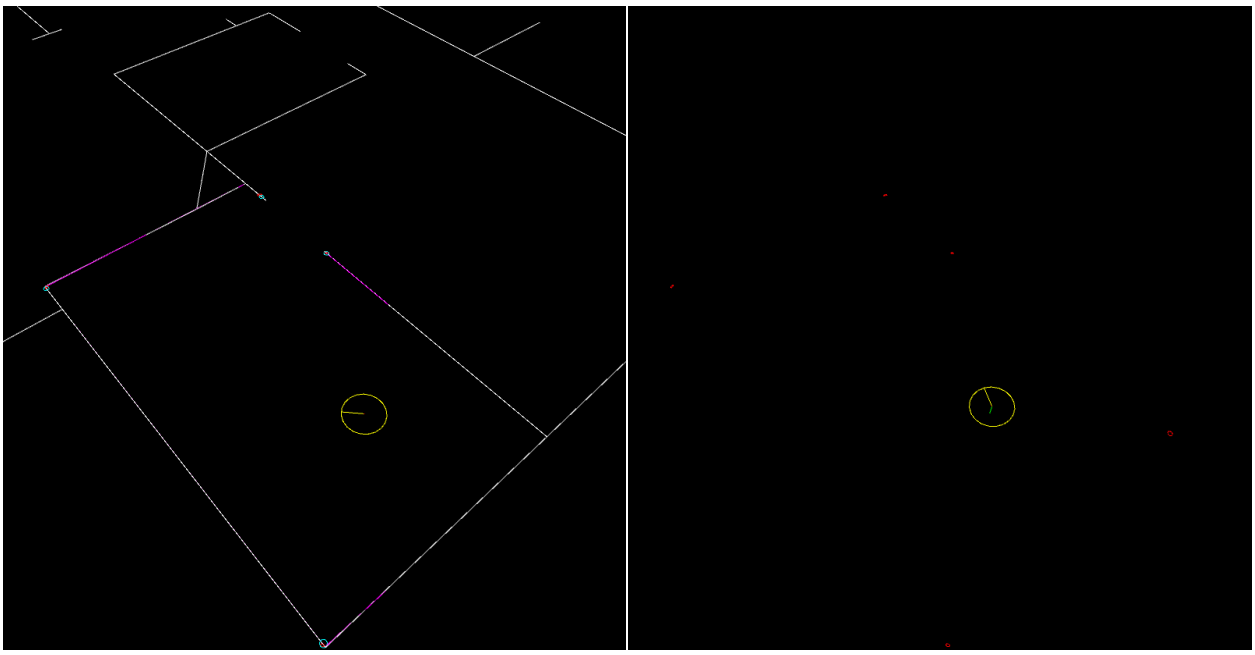

Figure 5: Landmarks from slam state. Red ellipses represent estimate and co-variance of landmark states

4.4 Path Planning

Obstacle Map To move the robot from a start pose to goal pose without colliding with obstacles, they need to be kept track of. The pipeline of obstacle and landmark detection in Figure 2 only outputs obstacles but does not keep track of them. For that we use a simple data structure called Hit grid. We imagine a rectangular boundary on the size of the environment or the subset of environment of interest. This can be an conservative upper bound that covers the region of interest well enough. Then we divide that rectangle into a lot

of small cells essentially creating a grid on the region of interest. Each cell has a counter which counts number of laser end it received and the number of inferred obstacle measurements passed through it. The number of hits it received is proportional to the probability of that cell containing an obstacle. As we receive more and more measurements this information is collected and kept track of over time essentially creating an obstacle map. A subtlety here is that we have to take the extent of the agent into account. As our robot is essentially a circle, for every cell that receives a hit we increase the counter of cells around it under a distance of the agents radius. This way we can just plan for the center of the agent in the next steps. This is illustrated in Figure 6.

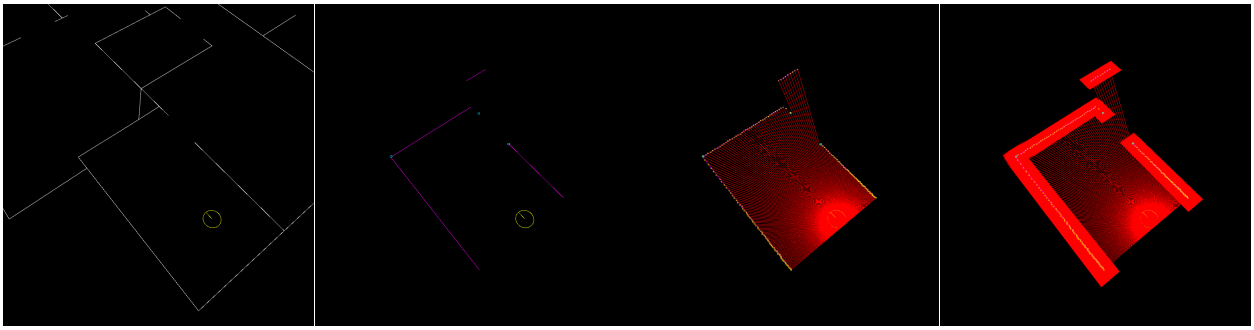

Figure 6: Layout, Detected obstacles and landmarks, Lasers and Hit grid respectfully. The red regions in fourth sub-figure indicate obstacles with radius of robot taken into account.

Path Planning/Replanning The hit grid itself can be seen as an 8-connected graph with centers of cells being vertices and each vertex connected to at most eight vertices on its sides and diagonals. Therefore given an start cell and goal cell, we can use a graph search method to find a path. We use A* for the search and distance from center of a cell to center of goal cell as an admissible heuristic. This significantly decreases the search time compared to other search methods like BFS, UCS while still yielding optimal paths. But A* works for a static known environment, whereas we have an unknown environment. We use the essence of D*-Lite algorithm [6] which is to replan our path once we detect an obstacle it. This simple addition deals with changing obstacle map over time. This is illustrated in Figure 7. A subtlety here is that our robot can not move perpendicular to its orientation. Therefore we at every vertex on the path we first orient the robot to next vertex and then translate it. This control strategy is simple and works well for our purpose.

Trajectory Optimization Although the previous setting deals with changes in obstacle map, the paths produced are only optimal in the discretized 8-connected graph space. For example in the first sub-figure of Figure 8 the robot can directly go towards the opening on the right instead of following all the vertices in between. Further more by following all vertices we have to orient the robot many times unnecessarily. This can be pruned using a simple rule. After reaching a vertex on the path we choose the furthest vertex on the path for which the line segment connecting from current cell to that vertex has no obstacles on it as our next local goal. This guarantees no collision and significantly reduces the number of maneuvers. A subtlety here is that we only keep track of obstacles per hit grid cell. So we

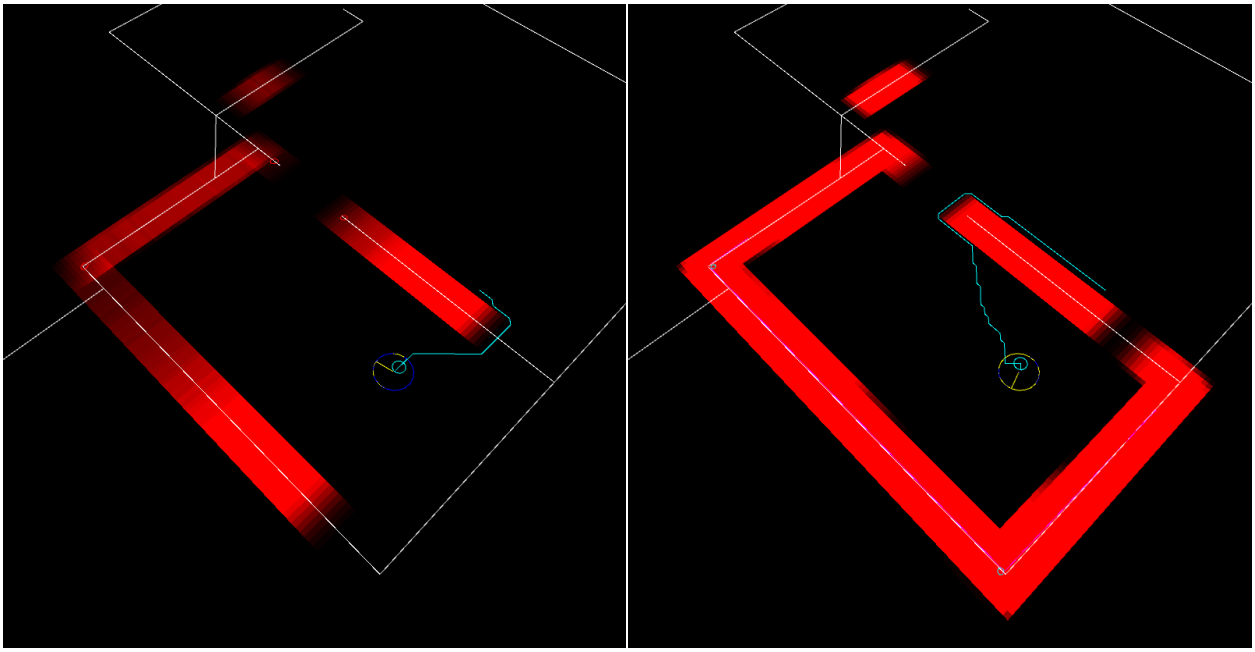

Figure 7: Path planning/re-planning. The goal is just outside the right wall. Cyan shows the planned path. On the left is the planned initially path. On the right is the re-planned path, when obstacle is detected in the original path.

approximate the line segment connecting current cell and a vertex on path using a set of hit grid cells that intersect with it. Therefore it finally boils down to a line rendering problem on a pixelated space, which is well-known and a fundamental problem in computer graphics. We use the well known Bresenham algorithm [3] for this. The second and third sub-figures of Figure 7 illustrates the optimized trajectory (blue) over planner trajectory (cyan).

4.5 Exploration

To explore all of the region of interest first we keep track of seen region. For this we use a separate instance of hit grid called sense grid. A sense grid cell is seen if a laser has intersected it at least once. The Bresenham algorithm is used to detect cell line intersection. This is illustrated in Figure 9.

5 Simulations

We use the parameters described in Table 5 for our simulator. We did not assume a limit on maximum linear and angular accelerations i.e. a new control is applied instantaneously in the robot. The error in control is multiplicative and error in laser scanner is additive. We assume zero uncertainty in robot pose at the start of simulation. We tested our robot in simple rectangle, simple block and apartment scenes. In all of the scenes we gave the robot a goal to reach. We used discontinuity thresholds of 100, 200, 100 respectively for the scenes. We used conservative offset while building obstacle maps to decrease the probability of

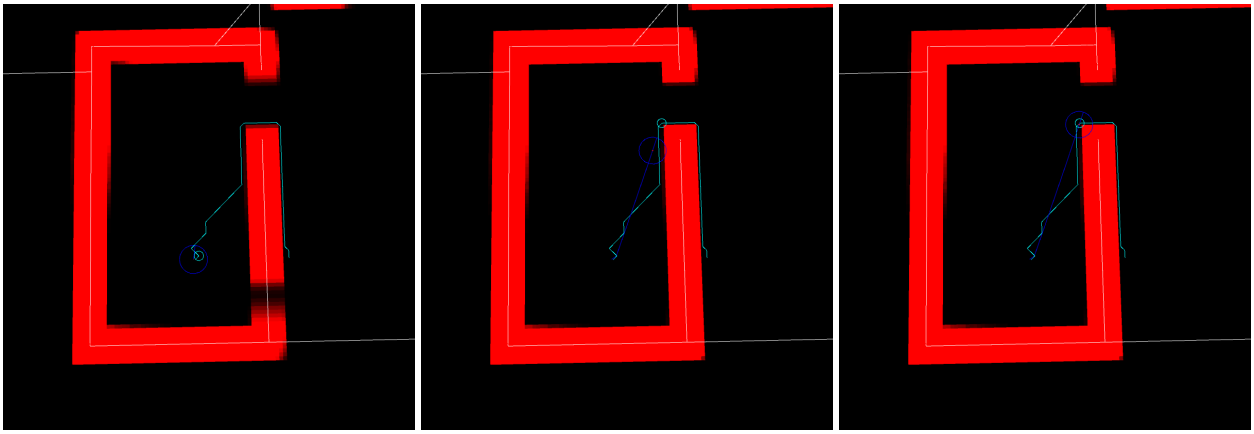

Figure 8: Trajectory optimization. The cyan path is the planned trajectory. The cyan circle is invariantly the vertex the robot is directly going towards. The blue path is the optimized trajectory.

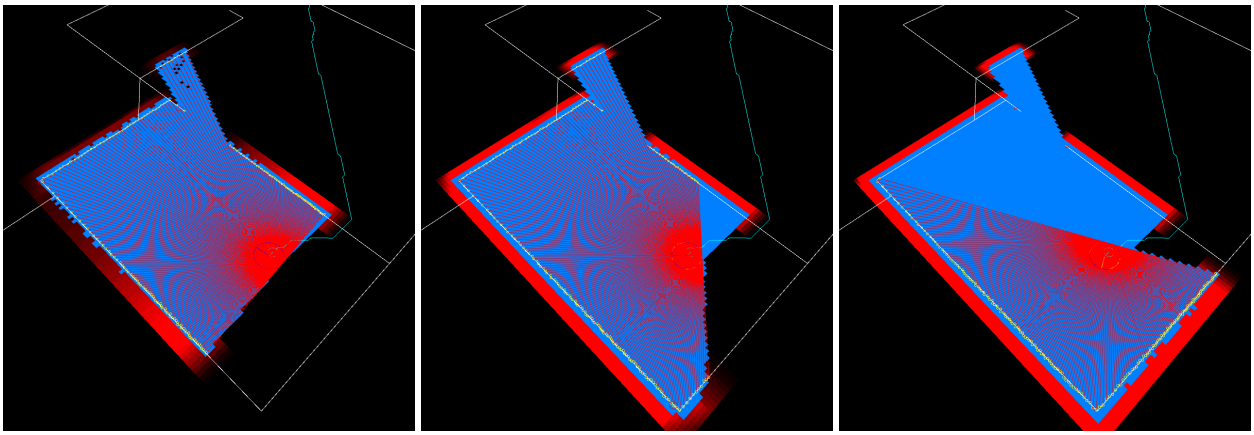

Figure 9: Keeping track of seen region. Blue area represents seen region.

Parameter	Value
Robot dt	0.01
Robot loop duration	16.67 ms
Control update	1 per 1 loop
Measurement	1 per 10 loops
Linear control error limit	0.1
Angular control error limit	0.1
Angle range of laser scanner	$-\pi$ to $+\pi$
Number of lasers	181
Max laser distance	500
Laser distance error limit	1
Laser angle error limit	0.05
RANSAC threshold	4
RANSAC iterations	1000
Minimum inliers for line	8

Table 1: Simulator parameters

collision. We compared obstacle and landmark detection with and without iterative endpoint partitioning and with and without least squares fitting.

6 Results and analysis

The robot was able to reach the goals in both simple rectangle and apartment scenes without collision while updating an obstacle map, seen region map and detecting landmarks and correcting its pose. Some moments of simulation are in Figure 10.

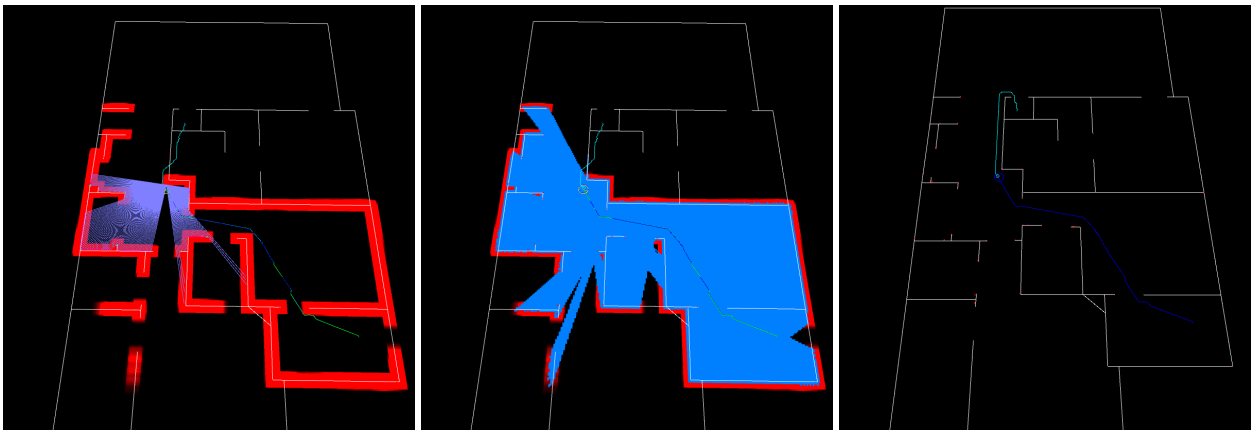

Figure 10: Keeping track of seen region. Blue area represents seen region.

The iterative end point method made the obstacle landmark detection pipeline much more robust as illustrated in Figure 11.

The least squares fitting made the obstacle detection much more stable. The periodic bad landmark culling proved efficient at removing bad measurements periodically as illustrated

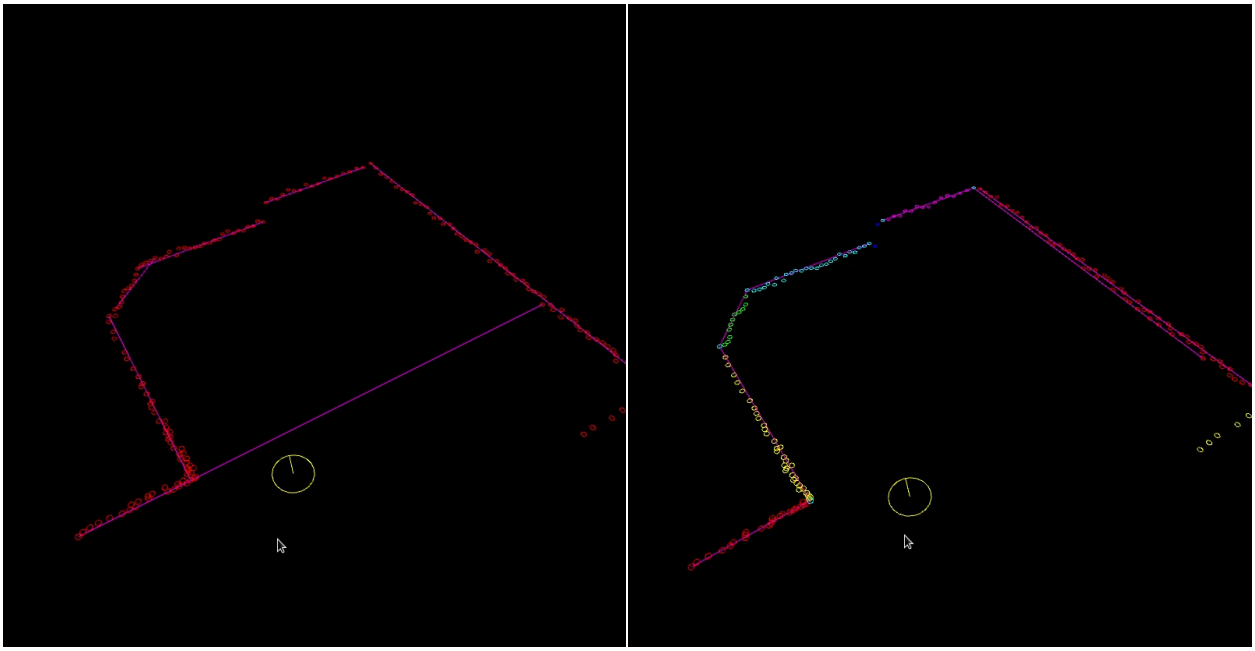

Figure 11: Left: RANSAC on first level partions, Right: RANSAC on second level partitions using IEP. Note the false positive that occurred in first sub-figure because of the point on right being included in the line of left.

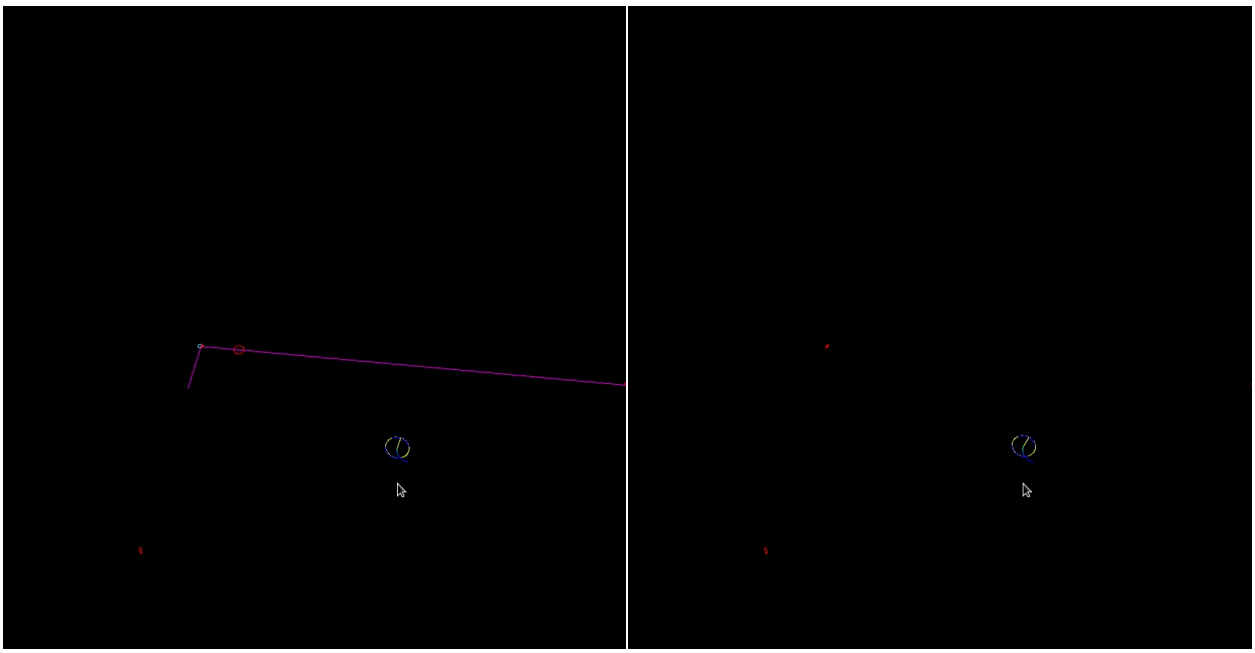

Figure 12: Left: A bad landmark is detected, Right: Bad landmark is culled based on the check that number of updates it received was less than a threshold.

in Figure 12.

The hit grid worked well for detecting and keeping track of obstacles, path planning/re-planning. The trajectory optimization decreased the time of travel significantly. The sense grid worked well for keeping track of seen region. All the simulations were close to 60 frame per second most of time indicating the real-time performance of the pipeline.

The most problematic scene was the simple block, due to its lack of landmarks as per our definition. There was significant drift of the robot which resulted in wrong updates of obstacle map and finally the goal position was estimated to be inside an obstacle. This serves as a control and proves that the robot in fact uses the landmarks in landmark rich apartment scene to improves its estimate regularly. In order to handle such scenes with sparse landmarks more features of environment such as parallel walls can be leveraged. But such context specific navigation was not a goal of our project.

Furthermore we observed that high values of control, especially ω drastically affects the obstacle and landmark detection due to the low rate of measurement compared to control updates. Slow movements with and frequent revisiting of landmarks with less uncertainty improved the estimates of both robot pose and landmarks with high uncertainty. It is important especially at the start when the uncertainty is at its minimum, to move slowly and make many updates of immediately visible landmarks for an effective and sustained estimation.

7 Conclusion

Our initial goal was, given a differential drive robot with a laser scanner in an static unknown 2D environment, the goal is to

1. Create a map of the obstacles in the environment leaving no area unexplored.
2. Simultaneously localize and map the landmarks in the environment.
3. Not collide with any obstacles while doing so.

We managed to achieve all of our goals except the active unseen area exploration due to time constraints. After keeping track of seen region, by randomly sampling goal positions in the largest unseen areas one can explore all of the accessible region of interest. This is a good direction for further refinements.

The most challenging parts were

1. To define and distinguish what a landmark is and what an obstacle is.
2. To extract obstacle and landmarks from laser scans in a robust and stable manner.
3. To cull bad landmarks from SLAM state.
4. Keep track of obstacles, plan and replan paths to goals taking agent of extent into account.
5. Finding intersections of a line and hit grid cells.
6. Making optimizations along the way to keep the system real-time.

However this approach makes some assumptions. Mainly we assume that obstacles are only polygons and environment has well-spread landmarks. If either of the assumption breaks our system cannot guarantee minimized collision which is well illustrated in our simple block demo. If the resolution of hit grid is low there can be a lot of false positives cutting off agent

from regions which are in reality accessible. If the region of interest is very large then the size of hit grid can become intractable where one might need to use sparse implementations.

8 Work division

Yashasvi Sriram Patkuri IEP, RANSAC, LS integration, Hit Grid, Planning/Replanning, Trajectory Optimization

Prashanth Kurella Implemented measurement partitioning and line segment extraction using IEP and RANSAC

Mandakinee Singh Patel Landmark detection, Bresenham implementation, Sense Grid

Abhinav Mehta Implemented the script for uncertainty calculation and Gaussian fitting over variable noise distribution.

References

[1] Å. Björck. *Numerical Methods for Least Squares Problems*. 1996.

[2] Geovany Borges and Marie-José Aldon. "Line Extraction in 2D Range Images for Mobile Robotics". In: *Journal of Intelligent and Robotic Systems* 40 (July 2004), pp. 267–297. DOI: 10.1023/B:JINT.0000038945.55712.65.

[3] J. E. Bresenham. "Algorithm for Computer Control of a Digital Plotter". In: *IBM Syst. J.* 4.1 (Mar. 1965), pp. 25–30. ISSN: 0018-8670. DOI: 10.1147/sj.41.0025. URL: https://doi.org/10.1147/sj.41.0025.

[4] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110.

[5] Martin A. Fischler and Robert C. Bolles. "Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography". In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: https://doi.org/10.1145/358669.358692.

[6] Sven Koenig and Maxim Likhachev. "D*lite". In: *Eighteenth National Conference on Artificial Intelligence*. Edmonton, Alberta, Canada: American Association for Artificial Intelligence, 2002, pp. 476–483. ISBN: 0262511290.

[7] Pedro Núñez et al. "Feature extraction from laser scan data based on curvature estimation for mobile robotics". In: vol. 2006. June 2006, pp. 1167–1172. DOI: 10.1109/ROBOT.2006.1641867.

[8] C. Runge. *Ueber die numerische Auflsung von Differentialgleichungen*. June 1895. DOI: 10.1007/bf01446807. URL: https://doi.org/10.1007/bf01446807.

[9] Sebastian Thrun and John J. Leonard. "Simultaneous Localization and Mapping". In: *Springer Handbook of Robotics*. Ed. by Bruno Siciliano and Oussama Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 871–889. ISBN: 978-3-540-30301-5. DOI: 10.1007/978-3-540-30301-5_38. URL: https://doi.org/10.1007/978-3-540-30301-5_38.

[10] Shin Ting, Mercedes, and Rocío Gonzales Márquez. *A non-self-intersection Douglas-Peucker Algorithm*.