RRT-SLAM for Motion Planning and comparison of results with RRT*

Markose Jacob
Robotics Engineering
A. James Clark School of Engineering
University of Maryland,
College Park, USA
markj11@umd.edu

Govind Ajith Kumar
Robotics Engineering
A. James Clark School of Engineering
University of Maryland,
College Park, USA
govindak@umd.edu

Rajeshwar NS
Robotics Engineering
A. James Clark School of Engineering
University of Maryland,
College Park, USA
rns@umd.edu

***Abstract*—In this paper the authors are addressing the sub-problem of mapping and localisation that arises during path planning for robots when exploring an unknown environment. Authors take into account sensing, localisation and mapping uncertainties in the path planning sub-problem. The robot used is Kobuki iClebo which is non-holonomic and equipped with a laser range sensor. Authors use Simultaneous Localization and Mapping (SLAM) algorithm to create a map of the unknown environment and then use both Rapidly Exploring Randomized Tree (RRT) and Rapidly Exploring Randomized Tree Star (RRT*) to find the path from one point to another within the map and compare both the results. The simulated SLAM explicitly accounts for sensor, localization and mapping uncertainty in the planning stage. The path selected by the robot is carefully selected after taking into account all collision possibilities. The preliminary results shows how SLAM can be used with both RRT and RRT*.**

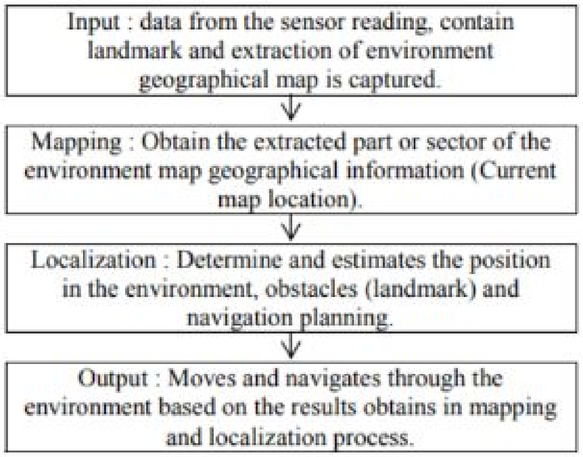
***Index Terms*—Kobuki iClebo, SLAM, RRT, RRT* , ROS**

I. Introduction

Exploration for autonomous robots has been an active topic in research for many years now. This process mainly consists of three iterative steps.1) plan for the robot's next movement to further explore its environment. 2) execute the plan 3) update the map and the position of the robot. Stage 1 consists of 2 sub problems, a) where to go b) how to get there. [1]

This is the problem addressed in this paper. The authors have a robot to exit in a static virtual environment, which is unknown to the robot. The robot is tasked with localisation and mapping as well as navigation through the path, by avoiding obstacles. The path that has been carved out is a result of the path planning that has been done through the algorithm widely known as Rapidly exploring Random Tree.

The authors have used Kobuki iClebo and the robot. Kobuki iClebo can only move in the direction of its orientation, this makes it a non-holonomic robot. The robot is capable of rotating on place using the two wheels which are connected to the motors. The robot is equipped with an on-board laser range sensor and moves in a static indoor environment. In order for the robot to plan its next configuration it needs to its own location in the map and the environment in which the bot is, which are continually updated in stage 3 as mentioned earlier. The authors tackled this problem by making use of a variant of Simultaneous Localization and Mapping (SLAM) known as FastSLAM, a particle based algorithm to update and generate a map of the environment and also localise the bot inside the map. SLAM problem involves a moving vehicle attempting to recover a spatial map of its environment, while simultaneously estimating its own pose (location and orientation) relative to the map. [2] SLAM technique can be very useful when the task is to navigate robots through unknown environments with no GPS connection. Some of the common applications of SLAM are underwater, underground and planetary exploration.


Input : data from the sensor reading, contain landmark and extraction of environment geographical map is captured.

Mapping : Obtain the extracted part or sector of the environment map geographical information (Current map location).

Localization : Determine and estimates the position in the environment, obstacles (landmark) and navigation planning.

Output : Moves and navigates through the environment based on the results obtains in mapping and localization process.

Fig. 1. General steps carried out by an autonomous robot [8]

In this project the authors are first comparing Rapidly-exploring Random Tree (RRT) algorithm with Rapidly-exploring Random Tree Star (RRT*) algorithm. The authors decided to go with these two as these have become widely popular in path planning problems over the last decade. The RRT algorithm is based on random sampling of a configuration space. The name itself gives a good idea of what it does. The sampled configurations are connected to a tree structure in which the resulting path can be found. The algorithm can be divided into three main parts: selection of a vertex for expansion, expansion and terminating condition. [3] An RRT algorithm can efficiently find a valid path. The performance of the RRT algorithm can be poor in environments containing narrow passages and also the path might not be optimal or smooth.

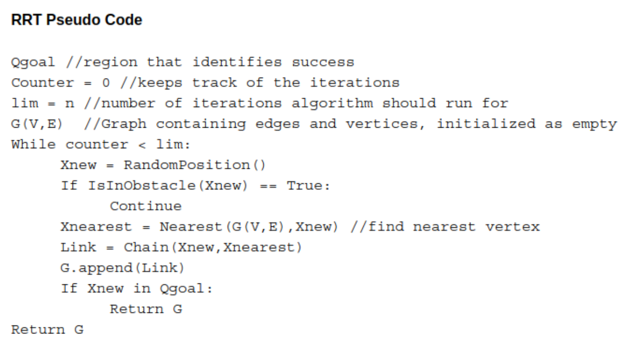
It is for this reason that they decided to implement RRT* and compare the results for the two. RRT* gives a smoother path because of the rewiring and finding best parent step. The down side to RRT* is that it takes much longer time compared to RRT to find the path from start to goal location. The comparison between the two algorithm was done on a map created by the authors. The map contains obstacles, images of the map can be seen later in this paper.

After this, the authors placed the robot in an unknown environment and navigate the robot to explore the area and generate a map using SLAM. Once the map is generated using SLAM algorithm then the next step is to find a collision free path to move the robot between any two points within the map. For this purpose the authors are using Rapidly-exploring Random Tree (RRT) algorithm.

II. Path Planning and SLAM

A. *Rapidly-exploring Random Tree (RRT)*

RRT is a data structure and path planning algorithm that randomly explores the working area filled with obstacles to find a valid path from start node to goal node. RRT is designed for efficiently searching paths in nonconvex high-dimensional spaces. RRTs are constructed incrementally by expanding the tree to a randomly-sampled point in the configuration space while satisfying given constraints, e.g., incorporating obstacles or dynamic constraints (nonholonomic or kinodynamic constraints). An RRT algorithm can efficiently find a valid path but it's not optimal and also the path found may not be smooth or could be longer. The different steps in RRT algorithm are:



RRT Pseudo Code

```
Qgoal //region that identifies success
Counter = 0 //keeps track of the iterations
lim = n //number of iterations algorithm should run for
G(V,E)  //Graph containing edges and vertices, initialized as empty
While counter < lim:
      Xnew = RandomPosition()
      If IsInObstacle(Xnew) == True:
            Continue
      Xnearest = Nearest(G(V,E),Xnew) //find nearest vertex
      Link = Chain(Xnew,Xnearest)
      G.append(Link)
      If Xnew in Qgoal:
            Return G
Return G
```

Fig. 2. RRT Pseudo Code

a) Initialization: First, the tree is initialized to have its root as an initial position of the robot which is the user provided start-point. Then the step size (delta) has to be decided. Having a higher step size makes exploration faster but also reduces the chances covering narrow paths and hence might not converge when there are many obstacles. So, a delta is given when there are many obstacles and less delta when there are few.
b) Random state: A random position X_{new} is generated in the configuration space using random sampling. Also this sample is checked if within an obstacle space using collision detection techniques.
c) Nearest neighbour: Based on a distance metric, the nearest vertex $X_{nearest}$ in the tree is selected that is close to X_{new}.
d) New state: The NEWSTATE selects a new configuration that is away from $X_{nearest}$ by an incremental distance (delta) toward the direction of X_{rand}. The new vertex and the edge is added to the tree and this process repeats until the goal radius is reached. The new node (x,y) is selected based on the following distance metric:

$$x = x_0 \pm l\sqrt{\frac{1}{1+m^2}} \tag{1}$$

$$y = y_0 \pm ml\sqrt{\frac{1}{1+m^2}} \tag{2}$$

Where m = slope of the line joining Xnew and Xrand and L is the branch size.

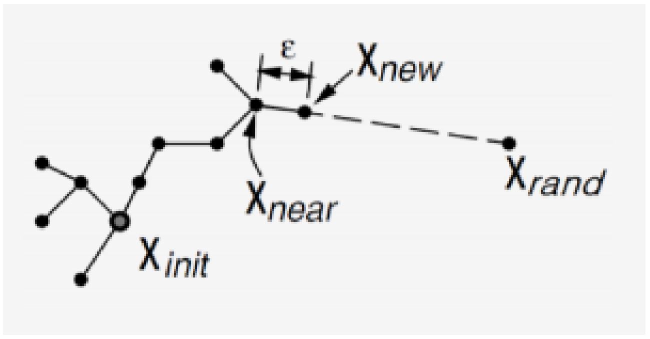


Fig. 3. Mechanism of Tree expansion

B. *Rapidly-exploring Random Tree Star (RRT*)*

RRT* is very similar to RRT. The path found using RRT* is shorter and smoother when compared to RRT. The reason for this is the additional two steps which are performed after the first few steps of RRT.
a) Parent node: Before adding X_{new} to the tree, an additional step of finding the best parent is performed. The next step is to connect X_{new} to the tree, in order to do this a vicinity is chosen around X_{new} and checked for explored nodes. The node in this region with least cost to come is selected as the parent. The cost is the sum of all the distance from the root to the parent node.
b) Rewire: Once the new node has been added to the tree, the nodes in the vicinity of the new node are checked to see if the cost to come for any of these nodes can be reduced by passing through X_{new}. If so then the nodes are rewired and this makes the final path more optimal compared to RRT*.

C. *RRT V/S RRT**

RRT* grows a tree incrementally, much in the same way RRT does. What makes RRT* different is the two added procedures that look at the path cost. The finding best parent

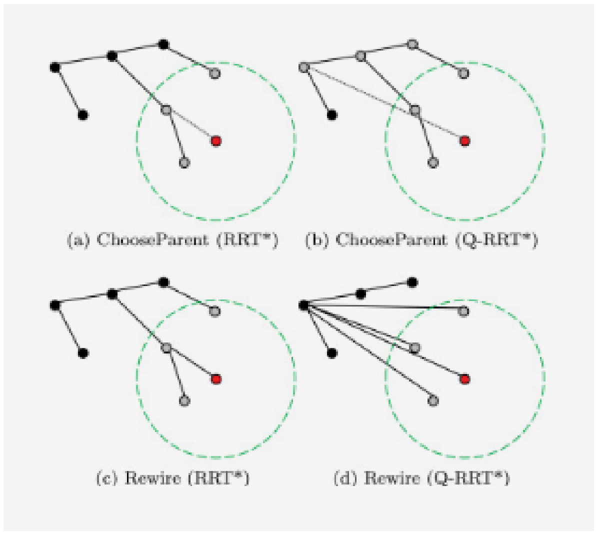
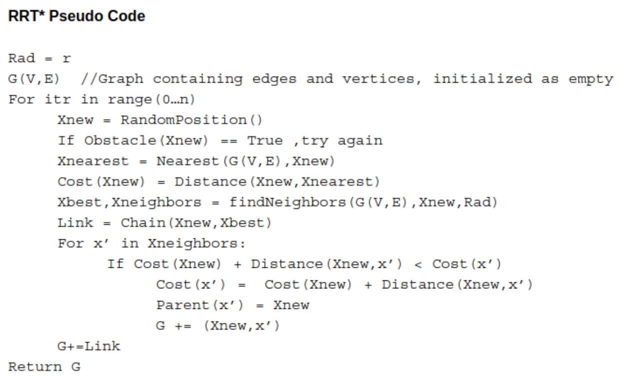


Fig. 4. Mechanish of RRT*



RRT* Pseudo Code

```
Rad = r
G(V,E)  //Graph containing edges and vertices, initialized as empty
For itr in range(0…n)
      Xnew = RandomPosition()
      If Obstacle(Xnew) == True ,try again
      Xnearest = Nearest(G(V,E),Xnew)
      Cost(Xnew) = Distance(Xnew,Xnearest)
      Xbest,Xneighbors = findNeighbors(G(V,E),Xnew,Rad)
      Link = Chain(Xnew,Xbest)
      For x' in Xneighbors:
            If Cost(Xnew) + Distance(Xnew,x') < Cost(x')
                  Cost(x') =  Cost(Xnew) + Distance(Xnew,x')
                  Parent(x') = Xnew
                  G += (Xnew,x')
      G+=Link
Return G
```

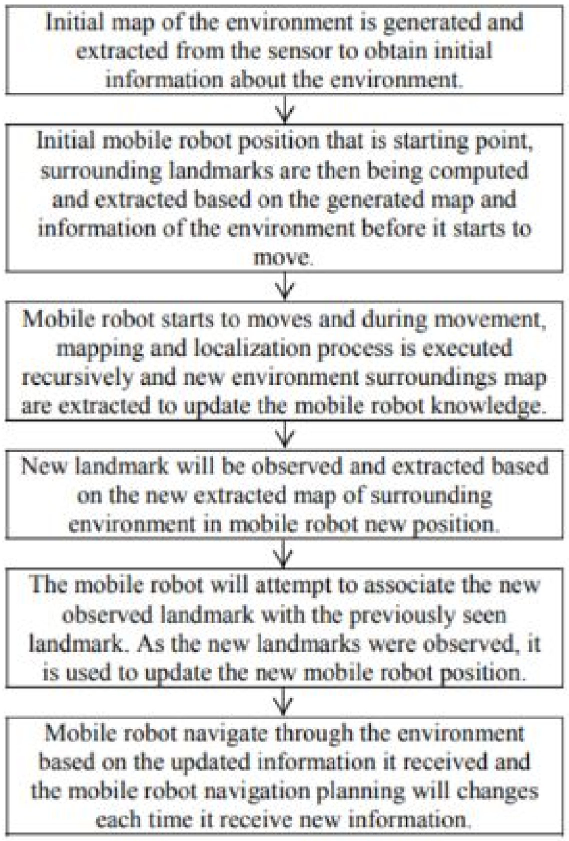
Fig. 5. RRT* Pseudo Code

procedure and the rewire procedure. With these new procedures, RRT* has been proven to be asymptotically optimal. However, the rate of convergence to find the optimal path is very slow compared to the traditional RRT since it explores the whole state space. Though there are various methods like RRT* Smart and informed RRT* which improves the convergence rate, a simple RRT takes less time and is less computationally expensive when used along with FastSlam.

D. Simultaneous Localization and Mapping (SLAM)

SLAM is a widely used concept that is an acronym for simultaneous localization of the robot and mapping of the environment. This concept was initially introduced in [7]. The concept is used when a robot is spawned in an environment that it has little to no knowledge about. In the earlier days, SLAM was known as EKF-SLAM. This is because the SLAM algorithm that was introduced used extended kalman filters for the solutions. The process of SLAM can be condensed into a few steps as shown in Fig 6.

The three main features in SLAM are Mapping, Localization and Navigation. Mapping involves the robot learning its



Initial map of the environment is generated and extracted from the sensor to obtain initial information about the environment.

Initial mobile robot position that is starting point, surrounding landmarks are then being computed and extracted based on the generated map and information of the environment before it starts to move.

Mobile robot starts to moves and during movement, mapping and localization process is executed recursively and new environment surroundings map are extracted to update the mobile robot knowledge.

New landmark will be observed and extracted based on the new extracted map of surrounding environment in mobile robot new position.

The mobile robot will attempt to associate the new observed landmark with the previously seen landmark. As the new landmarks were observed, it is used to update the new mobile robot position.

Mobile robot navigate through the environment based on the updated information it received and the mobile robot navigation planning will changes each time it receive new information.

Fig. 6. Steps followed in SLAM [8]

surroundings before taking any action. The map is generated by the robot based on the laser range sensors that have been fit to it. This data is crucial in localizing the robot. This task of mapping is succeeded by Localization. The concept of localization gives the robot the ability to estimate the distances to its surroundings and mark all the obstacles, which it should avoid. The last major step in a SLAM problem is the very task of navigation, from any desired start to the end goal. A point to be noted here is that, even during navigation, localization and mapping does not stop. They have to account for the ever-changing position of the robot and a new path has to be calculated as and when the new point is arrived on. The navigation can be based on a path planning algorithm that can be implemented by the user. When explored, the robot will have all the traversed paths in memory and it can traverse itself back to the starting point [8]. While, various forms of SLAM exist, in this project, the authors have implemented FastSLAM as discussed below. Most of the state-of-art SLAM algorithms calculate variants of the following posterior distribution:

$$p(\theta, s^t | z^t, u^t, n^t) \tag{3}$$

where $z^t = z_1, z_2....z_t$ is a sequence of measurements (e.g., range and bearing to nearby landmarks), $u^t = u_1, ...u_t$ is a sequence of robot controls (eg.velocities for the wheels). Authors assume that the loss of generality that only a single landmark is observed at each time t.The variables $n^t = n_1, ..., n_t$ are data association variables where each n_t specifies the identity of landmark at time t. Initially, authors assume n^t is known.[9]

E. FastSLAM

The paper uses a special type of localization and mapping algorithm called FastSLAM. This type of SLAM integrates particle filters and extended Kalman Filters [4]. The core idea of FastSLAM is to exploit dependencies between the different dimensions of the state space $(x_{0:t}, \theta_1, ..., \theta_N)$. FastSLAM is based on an important observation that the posterior can be factored:

$$p(\theta, s^t|z^t, u^t, n^t)\prod_n p(\theta_n|s^t, z^t, u^t, n^t) \quad (4)$$

This factorization is exact and universal in SLAM problems.It states that if one knew the path of the vehicle,the landmark positions could be estimated independently of each other. In FastSLAM each particle has attached its own map,consisting of N extended Kalman filters. Formally, the m-th particle S_t^m contains a path $s^[t, [m]$ along with Gaussian N landmark estimates, described by the mean $\mu_{n,t}^m$ and covariance $\Sigma_{n,t}^m$ Each update in FastSLAM begins with sampling new poses based on the most recent motion command u_t:

$$s_t^m \ p(s_t|s_{t-1}^m, u_t) \quad (5)$$

Here the distribution only uses motion command u_t but ignores measurment z_t. Next, FastSLAM updates estimation of landmarks according to the following posterior. This posterior takes measurement z_t into consideration:

$$\text{p}(\theta_{nt}|s^{t,[m]}, n^t, z^t) = \eta p(z_t|\theta_{nt}, s_t^{[m]}, n_t)p(\theta_{nt}|s^{t-1,[m]}, n^{t-1}, z^{t-1}) \quad (6)$$

Here η is a constant. This posterior is the normalized product of two Gaussians above. Thus to make the result Gaussian, EKF is used.[11] In final step, FastSLAM corrects for the fact that the pose sample s_t^m has been generated without consideration of the recent measurement. This is done so by resampling the particles [10]. The probability for the m-th particle to be sampled is given by the variable $w_t^[m]$ called as the importance factor:

$$w_t^{[m]} = \eta\int p(z_t|\theta_{nt}, s_t^{[m]}, n_t)p(\theta_{nt}|s^{t-1,[m]}, z^{t-1}, n^{t-1})d\theta_{nt} \quad (7)$$

The resampling operation can be implemented in O(M log N) time using trees, where M is the number of samples and N is the number of landmarks in the map. FastSLAM has been extended to SLAM with unknown data associations. if the data association is unknown,each particle m in FastSLAM makes its own local data association decision $\tilde{n_t}^m$ by maximizing the likelihood. The formula for finding the most likely data association maximizes the resulting importance weight:

$$\tilde{n_t}^{[m]} = argmax\ w_t^{[m]}(n_t) \quad (8)$$

Here $w_t^{[m]}(n_t)$ makes the dependence of the factor $w_t^{[m]}$ on the variable n_t explicit. A key characteristic of FastSLAM is that each particle makes its own local data association. In contrast, EKF techniques must commit to a single data association hypothesis for the entire filter Therefore each sample is defined as a path hypothesis. And for each path hypothesis an individual map is computed every time. In the robot used in the simulation presented in this paper, this algorithm takes the laser scan data along with the odometry and builds an occupancy grid. Since the robot is moving along a two dimensional configuration space, the occupancy grid is a two-dimensional grid. In the Robot Operating System software, the FastSLAM is carried out using the gmapping package [5]. Hence, for the purposes of this simulation the authors have adhered to this method as the preferred choice to conduct SLAM [6] The gmapping package subscribes to /odom and /scan topics to get the data and solves the path on Rviz. It is already wrapped for ROS in the ROS gmapping package. A high level view of gmapping can be seen below [6]

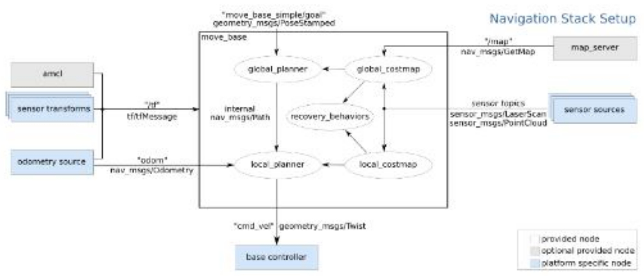


Fig. 7. High level view of gmapping

The algorithm employed by gmapping creates a grid-based map from the laser and also the pose data that has been collected by the robot by subscribing to /odom topic. Of course, it also has a subscription to /map topic for all the transformations as well as depiction of paths for visualization. A map that has been generated can be saved by using the mapsaver function that belongs to mapserver. The map-saver then saves two files of the map, a .yaml and a .pgm file. A screenshot of the image formed by the .pgm can be seen in Fig 8.

III. SIMULATION

The idea was set to be tested using Python and ROS. The simulation was tested on Python to check for proof of concept,

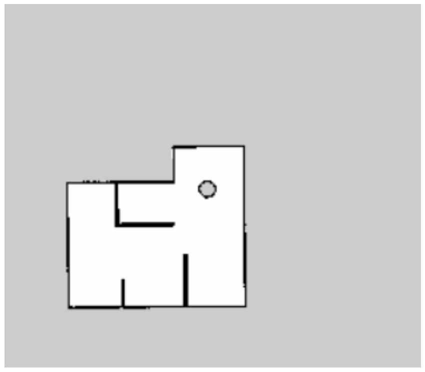


Fig. 8. Map received through gmapping

before implementation using SLAM and ROS on Gazebo and RViz.

A. Python Based Simulation

The python code was written by the authors and the code was written to perform RRT on a 2D obstacle space. The world file was designed by the authors and is seen to have sufficient walls and an object placed as well. The obstacle space was solved using RRT on python. The authors also compared the result with a python simulation of RRT*. The results can be seen below and the difference can be observed here. The empty world file used for the simulation can be seen in Fig 9.

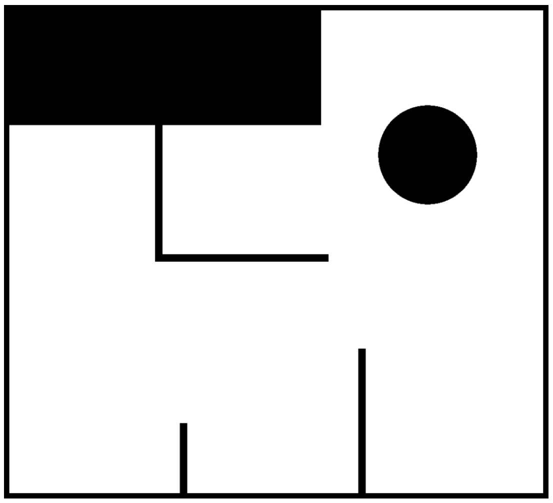


Fig. 9. Python Configuration space

The parameters were assigned by the authors as well. These included the delta, minimum distance to add, goal radius, number of nodes and all others that might be relevant for the simulation. The same thing was designed in Gazebo as well and the robot was spawned in the new maze as shown in Fig 11.

B. Robotic Operating System based Simulation (ROS)

In ROS, the authors use the Kobuki robot to navigate and solve the environment space. The robot spawns itself in the world arena created by the authors. In the simulation, the robot is equipped with a LIDAR, which can scan the environment and return a map file. The map is saved in .pgm and .yaml format, in order to later access the occupancy grid.The robot has various elements with its own coordinate axes, and it is important to set up the tf frames on ROS, so that the simulation carries out as intended. The tf frames of some of the major parts of the robot are shown below.

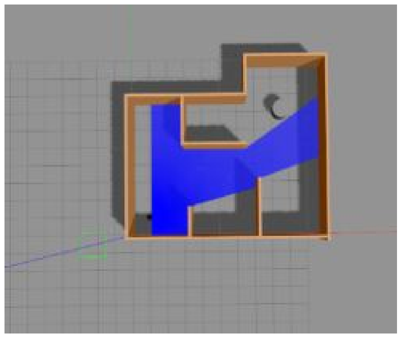
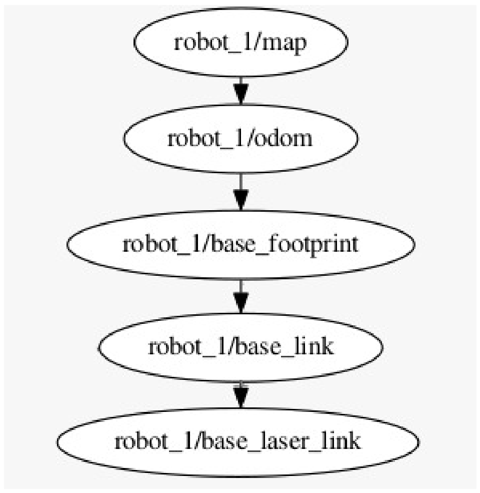


Fig. 10. Gazebo world environment



$$\begin{tikzcd}
\text{robot\_1/map} \arrow[d] \\
\text{robot\_1/odom} \arrow[d] \\
\text{robot\_1/base\_footprint} \arrow[d] \\
\text{robot\_1/base\_link} \arrow[d] \\
\text{robot\_1/base\_laser\_link}
\end{tikzcd}$$

Fig. 11. tf Frames from ROS for the Robot

C. Robot

A kobuki iClebo robot is implemented in the simulations performed here. The robot is only used in Gazebo and RViz, and a practical implementation was not performed for the same. The robot is accompanied with sensors, lidars and actuators. These can provide data with sufficient accuracy and precision , hence can be used for navigation and path planning. However, the robot is in itself not functional and has to be built upon its shell. This robot is widely used and is a low cost option, especially implemented to learn ROS. This robot serves as the base for the turtlebot robot, which has a platform built on the kobuki iClebo.

Parameter	Value
map_update_interval	2.0
maxUrange	50.0
maxRange	50.0
sigma	0.05
kernalSize	1
lstep	0.05
astep	0.05
iterations	5
Lsigma	0.075
ogain	3.0
linearUpdate	0.01
angularUpdate	0.01
temporalUpdate	0.1
particles	30
xmin	-5.0
ymin	-5.0
xmax	5.0
ymax	5.0
delta	0.1

TABLE I
GMAPPING PARAMETERS FOR SCANNING THE ENVIRONMENT

Fig. 12. kobuki iClebo

D. Exploration of the map

Here you can see various areas of the map that have been highlighted differently. The colored area depicts the local cost map. The local planner is very important to help with the navigation of the robot along a small section of the global path that was planned with the global planner. In order to complete this task, the robot has to be subscribed to /scan and /odom topics. Taking these velocity, the robot is assigned an appropriate velocity. On the turtlebot, this is done by publishing to the /cmd_vel topic.

The next colour that can be seen is by the global planner. The global planner here uses RRT to solve the path. Lastly, the third region is the area that has been scanned using FastSLAM

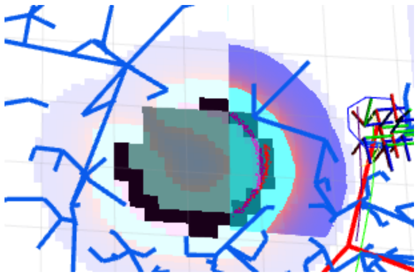


Fig. 13. Cost maps generated by the LIDAR

by the readings from the LIDAR. This denotes the areas which are not covered by any obstacle and free for the robot to traverse. The black borders denote the region of obstacles and untraversable paths, such as walls and out-of-bound areas. All of this can be seen clearly in the image above. Notice the vivid pink and purple, which the detection of the object that is near the robot. Notice the black, which is the definitve border of the object. Additionally, the light pink hues, denote the global planners. The white space around, is the path which the robot can traverse on. Finally, the green hue at the center of the image, is the untraversable region which the robot cannot reach at any given moment throughout its path.

E. Steps taken by the robot

The robot has to learn about its position for it to start solving and generate the costmap. In gmapping on ROS, this was aided by the Adaptive Monte Carlo Localization algorithm, also known as AMCL [5]. This algorithm employs a particle filter to get the localization data. Hence, the robot is spawned as per the data provided by the user in the launch file. This is followed by setting a goal position for the robot to calculate the final goal position. The goal is selected by the 2D navigation tool on RViz. This data is sent to the global planner. The global path is broken down to many small local costmaps which is solved by the local planner. As mentioned above, the local planner gets the necessary data from /scan and /odom and executes this task.

IV. Results

The results of the simulations done on Python for RRT and RRT* can be seen in Fig 15, Fig 16, Fig 17, Fig 18 and Fig 19. First, the authors have done the simulation for path planning through RRT.

Notice how the path is just longer, hence there are regions where the two points it connects overlaps an obstacle, hence these parameters exceed the necessary requirements. This is not a correct solution, even though the path is reached because the path crosses through the obstacle. Hence, the programme has to be altered by keeping this change in mind.Reducing the number of nodes should also be done carefully, because if it's too small the programme will terminate and RRT will not be able to find the goal before the node count is reached. Hence, these parameters have to be calculated carefully before they

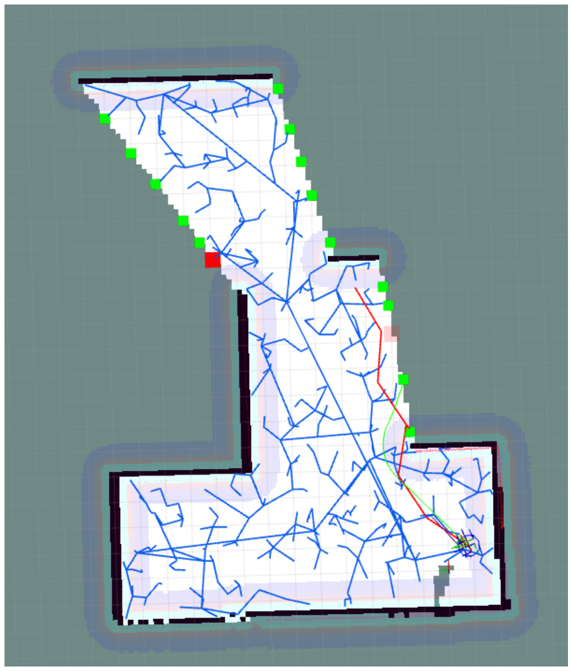


Fig. 14. RViz environment

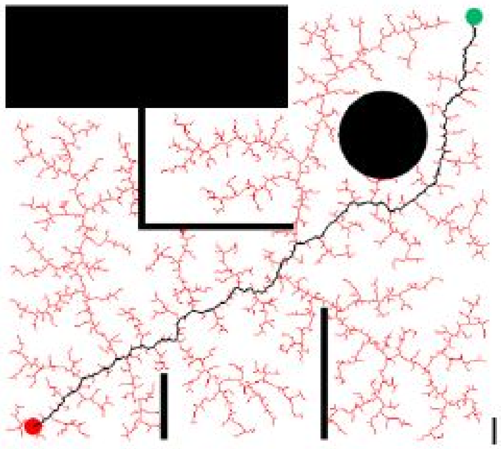


Fig. 15. Parameters (delta = 8, minimum distance to add = 1.0, number of nodes = 10000)

are used for the purposes of this experiment and simulation. This can be seen in the Fig 17.

Following this, the authors have extended the concept to RRT*. RRT* gives a shorter path, however the computation is much higher. The results with various parameters can be seen in Fig 18 and Fig 19.

Just like RRT, the RRT* can't also have high delta value because the path then goes through the obstacles. Hence these values have to be tuned by the author. The number of nodes also have to be very high because RRT* generates much more nodes as seen above, as compared to the basic RRT algorithm.

In ROS, it can be seen that the robot successfully traverses the path from the start to the goal using RRT algorithm. The path is similar to the one predicted by the python algorithm. The local and global costmap is also visible using the RRT Map tool. The robot is seen to solve the path initially without

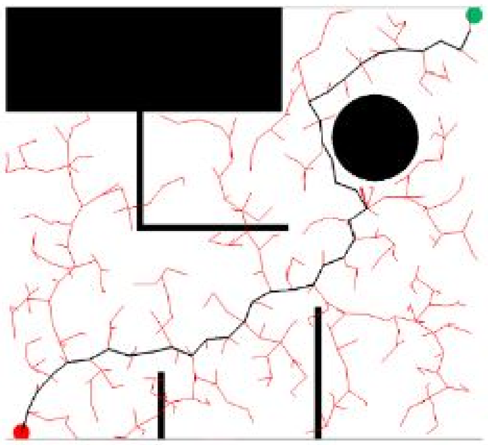


Fig. 16. Parameters (delta = 50, minimum distance to add = 1.0, number of nodes = 10000)

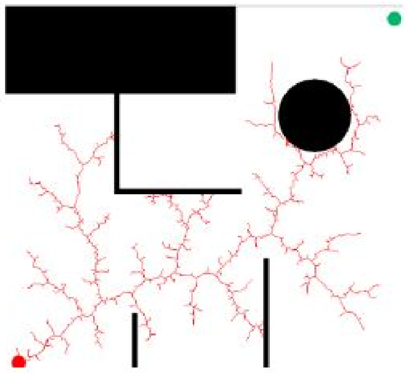


Fig. 17. Parameters (delta = 8, minimum distance to add = 1.0, number of nodes = 2500)

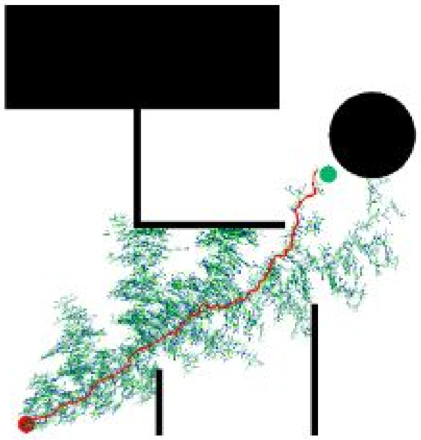


Fig. 18. Parameters (delta = 12, minimum distance to add = 1.0, number of nodes = 20000)

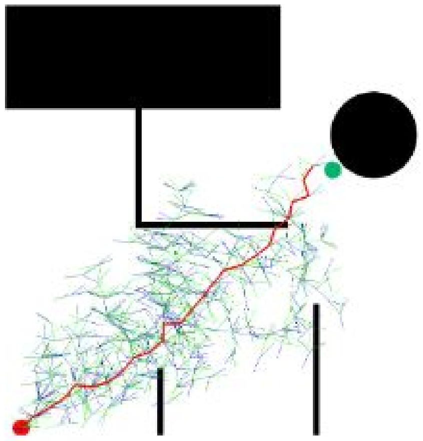


Fig. 19. Parameters (delta = 40, minimum distance to add = 1.0, number of nodes = 20000)

any obstacles and then relies on the data from the LIDAR, to detect obstacles and go around it. The path the planner calculates can be seen in green here on the RViz maps.

In RViz, the costmaps were calculated and the robot solved the environment space.Notice how the color gradient can be seen in the image below. This is followed by a region of slightly less vivid colours. The vivid colours denote the area taken care of by the local planner.

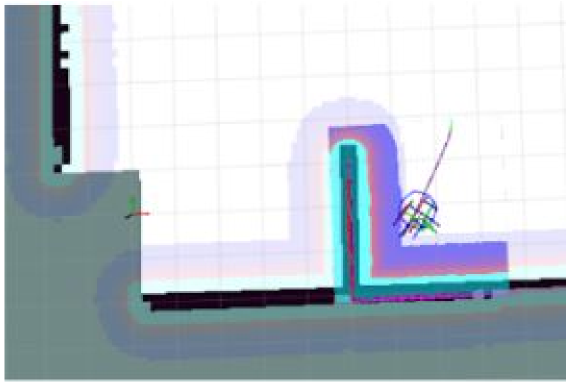


Fig. 20. Costmaps calculated by the robot and LIDAR data

In the image below, notice the path calculated by the robot when the goal is defined by the user. This can be seen as a green line which bends around the walls. The robot takes into account the LIDAR data published to the /scan topic to solve this configuration space. Notice the red and blue paths that has been calculated. Everytime the robot moves, it updates itself in the location and this can be seen live on RViz and the red path keeps changing, which is the robots planner working itself out to see what the best path is, for the robot to reach from the initial to the final point, thereby completing the task.

V. Conclusion

The robot's trajectory that was calculated using Python and RRT was implemented. The path was also visualized using

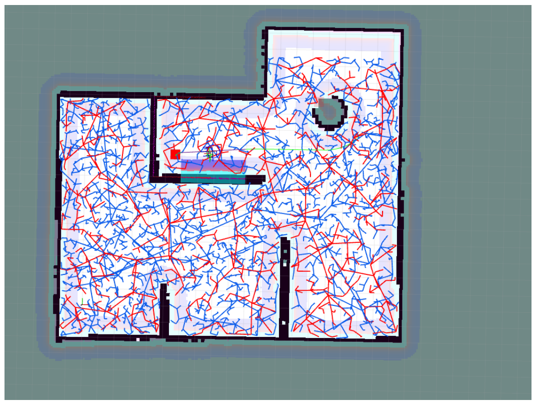


Fig. 21. Final Paths calculated by the Robot on RVIZ

Gazebo and RViz, using ROS kinetic. This has verified the results predicted by the path planning algorithms that was initially simulated by the authors. The advantage was also noted between RRT and RRT*, the latter of which gave the shortest path, even though it had more computational expenses.

References

[1] Huang, Y. and Gupta, K., 2008, September. RRT-SLAM for motion planning with motion and map uncertainty for robot exploration. In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 1077-1082). IEEE.

[2] Thrun, S., Montemerlo, M., Koller, D., Wegbreit, B., Nieto, J. and Nebot, E., 2004. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. Journal of Machine Learning Research, 4(3), pp.380-407.

[3] Vonásek, V., Faigl, J., Krajník, T. and Přeučil, L., 2009. RRT-path–a guided rapidly exploring random tree. In Robot Motion and Control 2009 (pp. 307-316). Springer, London.

[4] Thrun, S., Montemerlo, M., Koller, D., Wegbreit, B., Nieto, J. and Nebot, E., 2004. Fastslam: An efficient solution to the simultaneous localization and mapping problem with unknown data association. Journal of Machine Learning Research, 4(3), pp.380-407.

[5] Lentin Joseph and Jonathan Cacace. 2018. Mastering ROS for Robotics Programming - Second Edition: Design, build, and simulate complex robots using the Robot Operating System (2nd. ed.). Packt Publishing.

[6] Estler, D., Path Planning and Optimization on SLAM-Based Maps.

[7] J. J. Leonard and H. F. Durrant-Whyte, "Mobile robot localization by tracking geometric beacons," Robotics and Automation, IEEE Transactions on, vol. 7, pp. 376-382, 1991

[8] Khairuddin, A.R., Talib, M.S. and Haron, H., 2015, November. Review on simultaneous localization and mapping (SLAM). In 2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE) (pp. 85-90). IEEE.

[9] Montemerlo, M., Thrun, S., Koller, D. and Wegbreit, B., 2003, August. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In IJCAI (pp. 1151-1156).

[10] Rubin, D.B., 1988. Using the SIR algorithm to simulate posterior distributions. Bayesian statistics, 3, pp.395-402.

[11] Maybeck, P.S., 1979. Stochastic models, estimation and control, volume 1 Academic Press.

.