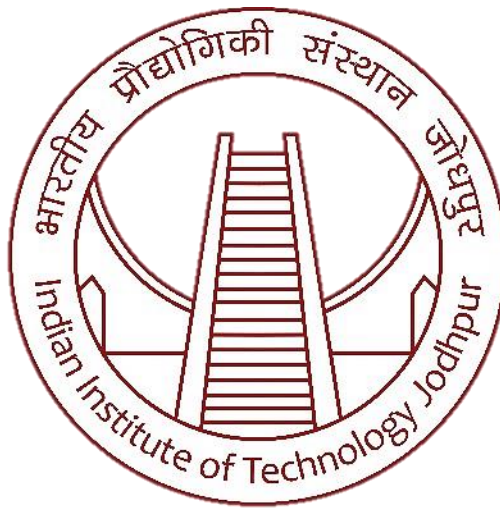


CSL 7650
AUTONOMOUS SYSTEMS



ASSIGNMENT-3
PHASE -2

Submitted by: -
ARASHDEEP SINGH
M23IRM003

Main aim of the Project:-

Robot should be capable of autonomous navigation.

Robot should be capable of obstacle detection.

Robot should be capable of path planning.

Robots should be capable of interacting with the environment.

Platform Used

Ubuntu, ROS Neotic, Gazebo, Rviz

Strategies available

1. Active SLAM with Deep Reinforcement Learning (DRL):

- Active SLAM combines path planning with SLAM to enhance autonomous navigation in complex environments [1].
- In this approach, fully convolutional residual networks are used to recognize obstacles and generate depth images.
- The robot's avoidance obstacle path is planned using the Dueling DQN (Deep Q-Network) algorithm.
- Simultaneously, a 2D map of the environment is built based on Fast SLAM.
- Experiments demonstrate successful obstacle identification, avoidance, and autonomous navigation in complex environments.

2. Building a Map from 3D LiDAR Data Using SLAM:

- The goal is to process 3D LiDAR data from a sensor mounted on a vehicle to build a map and progressively estimate the vehicle's trajectory.
- Maps created this way can facilitate path planning for vehicle navigation or be used for localization.
- This example demonstrates how to build a 2D occupancy map from 3D LiDAR data using SLAM algorithms [2].

3. SLAM-Assisted Coverage Path Planning for LiDAR Mapping Systems:

- Classic offline Coverage Path Planning (CPP) can be adapted for online SLAM by making two modifications [3]:
 - Convex decomposition of the polygonal coverage area allows arbitrary initial point selection while tracing the shortest coverage path.

- A novel approach stitches together different cells within the polygonal area to form a continuous coverage path.

4. Using an Inertial Navigation Sensor (INS):

- In addition to 3D LiDAR data, an INS can be used to help build the map during SLAM [4] .
- Read the point cloud data from the image files using the helperReadPointCloudFromFile function.
- Combine the LiDAR data and INS readings to build the map and estimate the vehicle's trajectory.

Strategy Chosen

Building a Map from 3D LiDAR Data Using SLAM

After getting the map, these steps will be followed:-

1. Define Your Goals:

- Before implementing path planning, clarify your objectives. Are you aiming for the shortest path, avoiding obstacles, or optimizing for energy efficiency?
- Consider factors like robot speed, environment complexity, and safety requirements.

2. Choose a Path Planning Algorithm:

There are several algorithms you can use for path planning:

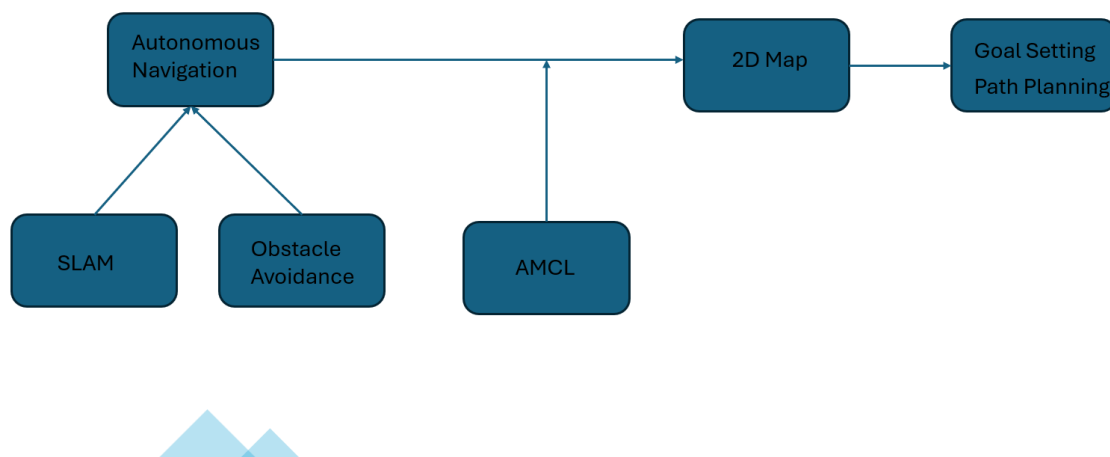
- **A-Star (A*) Algorithm:**
 - A heuristic search algorithm that finds the shortest path between two points.
 - It works well for grid-based maps and can handle obstacles.
 - You'll need to define a heuristic (distance estimate) to guide the search.
- **Dijkstra's Algorithm:**
 - Finds the shortest path in a weighted graph.
 - It explores all possible paths, so it's not the most efficient for large maps.
- **RRT (Rapidly Exploring Random Trees):**
 - Useful for continuous spaces.
 - Randomly samples points and grows a tree to explore the space.
 - Can handle complex environments.
- **PRM (Probabilistic Roadmap):**
 - Builds a roadmap of the environment by sampling random points.
 - Connects these points to create a graph.

- Useful for high-dimensional spaces.
- **Potential Fields:**
 - Assigns attractive and repulsive forces to points in the environment.
 - The robot moves along the gradient of the potential field.
 - Good for real-time obstacle avoidance.

3. Map Representation:

- Convert your 2D map into a format suitable for path planning.
- Common representations include occupancy grids, Voronoi diagrams, or graphs.

Here is flow that I am following for this project :-



Strategy Implementation

Here is the link to see the simulation videos.

https://drive.google.com/drive/folders/1aPPp3mUnslh3fsskbiCS2ZTs8MwLSTWa?usp=drive_link

Before understanding Autonomous navigation, let us understand :-

LiDAR

LiDAR works on the same principles as radar ("radio detection and ranging," a location system that is often used by ships and planes) and sonar ("sonic navigation and ranging," a system that is typically used by submarines). All three technologies emit waves of energy to detect and track objects. The difference is that while radar uses microwaves and sonar uses sound waves, LiDAR uses reflected light, which can measure distance faster, with greater precision and higher resolution than either radar or sonar.[5]

LiDAR Components

A typical LiDAR instrument is made up of several components:

- a laser scanner that emits rapid pulses of near-infrared laser light
- a LiDAR sensor that is used for detecting and collecting the returning light pulses, and
- a processor for calculating the time and distance and for building the resultant data set, called a LiDAR point cloud.

SLAM [6] [7][8]

Simultaneous localization and mapping (SLAM) is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. While this initially appears to be a chicken or the egg problem, there are several algorithms known to solve it in, at least approximately, tractable time for certain environments. Popular approximate solution methods include the particle filter, extended Kalman filter, covariance intersection, and GraphSLAM. SLAM algorithms are based on concepts in computational geometry and computer vision, and are used in robot navigation, robotic mapping and odometry for virtual reality or augmented reality.

In this project LiDAR SLAM(gmapping SLAM) is used. A occupancy grid map will be made.

1. Gmapping:

- **Gmapping** is a specific implementation of SLAM.
- It uses laser scan data (from a laser range-finder) and odometry data (robot's motion information) to build a 2D occupancy grid map.
- The goal is to create a map that represents the environment, similar to a building floorplan.
- The underlying algorithm in Gmapping is called **Rao-Blackwellized Particle Filters** (RBPF).

2. How Does Gmapping Work?

- The robot receives laser scans (measurements of distances to obstacles) and odometry data (how it moved).
- Gmapping processes this data to estimate the robot's position and update the map.
- It uses a particle filter approach, where multiple particles (hypotheses) represent possible robot poses. These particles are updated based on sensor data.
- Over time, the algorithm converges to a more accurate map and robot pose.

Obstacle avoidance is also uses the part of Gmapping SLAM. Like visibility planning is default module in navigation stack for obstacle avoidance.

Avoidance Strategies:

Local Obstacle Avoidance: GMapping considers obstacles detected by the LIDAR (laser range finder) during localization. It avoids collisions by adjusting its path in real-time.

Global Path Planning: When planning a long path, GMapping ensures that the path avoids obstacles. It uses the map information to find a safe route.

Dynamic Obstacles: GMapping can handle dynamic obstacles (e.g., moving people or other robots) by updating the map and adjusting the planned path accordingly.

Here are the parameters of LiDAR SLAM which I used in this project:-

```
map_update_interval: 2.0
maxUrange: 5.0
sigma: 0.05
kernelSize: 1
lstep: 0.05
astep: 0.05
iterations: 3
lsigma: 0.075
ogain: 3.0
lskip: 0
minimumScore: 50
srr: 0.1
srt: 0.2
str: 0.1
stt: 0.2
linearUpdate: 1.0
angularUpdate: 0.2
temporalUpdate: 0.5
resampleThreshold: 0.5
particles: 100
xmin: -10.0
ymin: -10.0
xmax: 10.0
ymax: 10.0
delta: 0.05
llsamplerange: 0.01
llsamplestep: 0.01
```

```
lasamplerange: 0.005
lasamplestep: 0.005
```

Let's break down the parameters and their roles:

1. **map_update_interval**: This parameter specifies the time interval (in seconds) between consecutive map updates. It controls how often the map is updated based on sensor data.
2. **maxUrange**: The maximum range (in meters) of the laser range finder (LIDAR). It defines the maximum distance at which the sensor can detect obstacles.
3. **sigma**: The standard deviation of the noise in the range measurements. It accounts for measurement uncertainty.
4. **kernelSize**: The size of the kernel used for smoothing the laser scans. Larger values result in more robust maps but may reduce accuracy.
5. **lstep** and **astep**: These parameters control the angular and linear step sizes for ray casting during map updates. They affect the granularity of the map.
6. **iterations**: The number of iterations for the scan matching algorithm. More iterations improve accuracy but increase computation time.
7. **lsigma**: The standard deviation of the noise in the laser scan angles. Similar to sigma, it accounts for angular measurement uncertainty.
8. **ogain**: The gain factor for the occupancy grid map. It adjusts the likelihood of cells being occupied or free based on sensor data.
9. **lskip**: The number of laser scans to skip during map updates. Skipping scans can reduce computation time.
10. **minimumScore**: The minimum score required for a laser scan match to be accepted. Higher scores indicate better alignment.
11. **srr**, **srt**, **str**, and **stt**: These parameters model the motion noise in the robot. They affect the particle filter used for localization.
12. **linearUpdate** and **angularUpdate**: The thresholds for updating the particle filter based on robot motion. If the robot moves beyond these thresholds, a resampling step occurs.
13. **temporalUpdate**: The time threshold for temporal updates. If the time since the last update exceeds this value, a resampling step is triggered.
14. **resampleThreshold**: The threshold for resampling particles. If the effective number of particles falls below this value, resampling occurs.
15. **particles**: The number of particles used in the particle filter. More particles improve localization accuracy but increase computation time.
16. **xmin**, **ymin**, **xmax**, and **ymax**: The boundaries of the map in the robot's coordinate system.
17. **delta**: The resolution of the occupancy grid map. It defines the size of each cell in the grid.

Now Comes the question how robot localizes itself in map. It is done using AMCL.

Adaptive Monte Carlo Localization (AMCL)

1. What is AMCL?

- **Adaptive Monte Carlo Localization (AMCL)** is a probabilistic localization algorithm designed to handle non-uniformly distributed uncertainties in the environment.
- It extends the basic MCL algorithm by dynamically adjusting the number of particles based on KL-distance (Kullback-Leibler divergence) to ensure that the particle distribution converges to the true distribution of the robot state based on all past sensor and motion measurements with high probability [9].

2. How Does AMCL Work?

- **Particle Filter:** Like MCL, AMCL uses a particle filter to estimate the robot's pose (position and orientation) relative to a known map.
- **Laser Scans and Maps:** AMCL takes laser scans and a laser-based map as inputs.
- **Dynamic Particle Adjustment:**
 - Initially, AMCL initializes its particle filter according to the provided parameters.
 - As the robot moves and receives new sensor data, AMCL dynamically adjusts the number of particles.
 - The goal is to maintain a representative set of particles that accurately reflects the robot's true pose.
- **KL-Distance Criterion:**
 - KL-distance measures the difference between two probability distributions.
 - AMCL uses KL-distance to assess how well the current particle distribution matches the true distribution.
 - If the KL-distance exceeds a threshold, AMCL resamples particles to improve convergence.
- **Pose Estimates:**
 - AMCL provides pose estimates (including covariance) based on the maintained particle cloud.
 - The estimated pose represents the robot's position within the known map.

Other than that, a default navigation stack by ROS is also used.

The **ROS Navigation Stack** is a powerful set of software packages designed to help mobile robots move from one location to another reliably. Let's dive into the details:

1. Conceptual Overview:

- The Navigation Stack is fairly straightforward in concept. It takes input from **odometry** (robot motion data) and **sensor streams** (such as laser scans) and produces **velocity commands** to send to a mobile robot base.
- In essence, it helps a robot navigate autonomously by planning safe paths and controlling its movement.

2. Requirements and Constraints:

Hardware Requirements:

- The Navigation Stack is designed for **differential drive** and **holonomic wheeled robots**.
- It assumes that the robot's mobile base can be controlled using desired velocity commands (linear and angular velocities).
- A **planar laser** mounted on the robot is essential for map building and localization.
- While it works on robots of arbitrary shapes and sizes, it performs best on robots that are nearly square or circular.

3. Typical Setup and Configuration:

- **Transform Configuration:** Set up the robot's coordinate transforms (tf tree) to ensure accurate sensor data alignment.
- **Publishing Sensor Streams:** Publish sensor data (e.g., laser scans) to appropriate ROS topics.
- **Odometry Information:** Provide odometry data (motion information) to the Navigation Stack.
- **Building a Map:** Use the planar laser to create a map of the environment.
- **Costmap Configuration:**
 - The Navigation Stack uses **costmaps** (representations of obstacles and free space) for global and local planning.
 - Configure costmaps based on robot dynamics and environment characteristics.
- **Base Controller:** Implement a controller that translates velocity commands into actual robot motion.
- **Global and Local Path Planning:** The Navigation Stack computes paths from the robot's current position to a goal location, considering obstacles and constraints.

Algorithm used by Navigation Stack

Dijkstra's algorithm

The algorithm maintains a set of visited vertices and a set of unvisited vertices. It starts at the source vertex and iteratively selects the unvisited vertex with the smallest tentative distance from the source. It then visits the neighbors of this vertex and updates their tentative distances if a shorter path is found. This process continues until the destination vertex is reached, or all reachable vertices have been visited.[12]

Here are some parameters that are controlled for reaching to goal.

TrajectoryPlannerROS:

```
# Robot Configuration Parameters
max_vel_x: 0.18
min_vel_x: 0.08

max_vel_theta: 1.0
min_vel_theta: -1.0
min_in_place_vel_theta: 1.0
```

```
acc_lim_x: 1.0
acc_lim_y: 0.0
acc_lim_theta: 0.6

# Goal Tolerance Parameters
xy_goal_tolerance: 0.10
yaw_goal_tolerance: 0.05

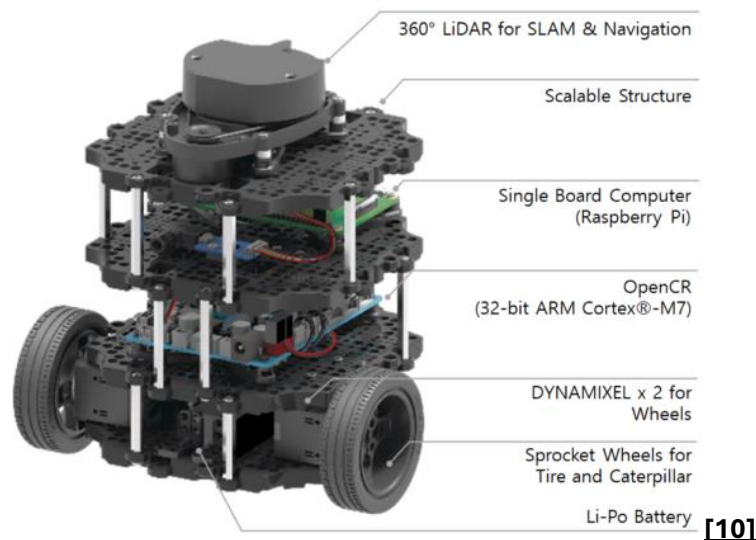
# Differential-drive robot configuration
holonomic_robot: false

# Forward Simulation Parameters
sim_time: 0.8
vx_samples: 18
vtheta_samples: 20
sim_granularity: 0.05
```

Robot Used and Environment

I have used Turtle bot 3 robot.

TurtleBot3 Burger



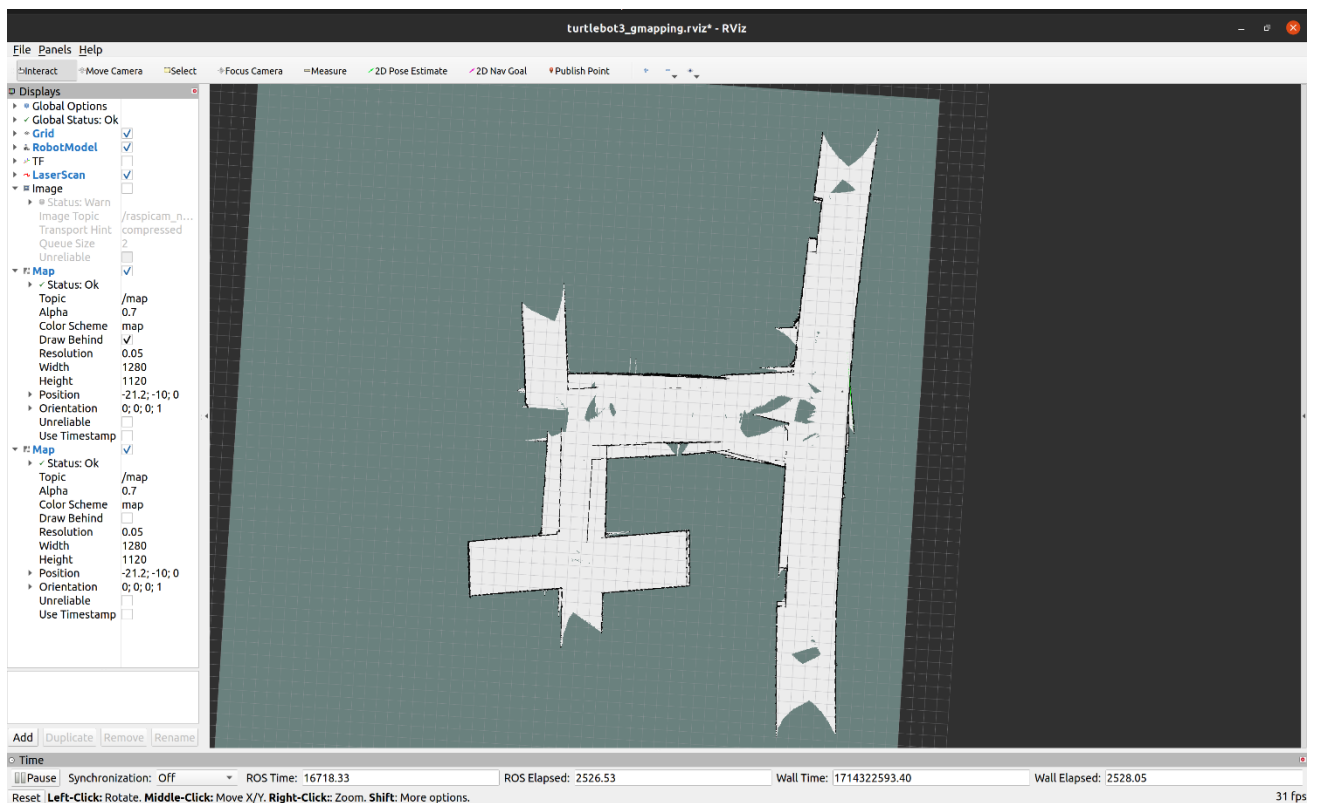
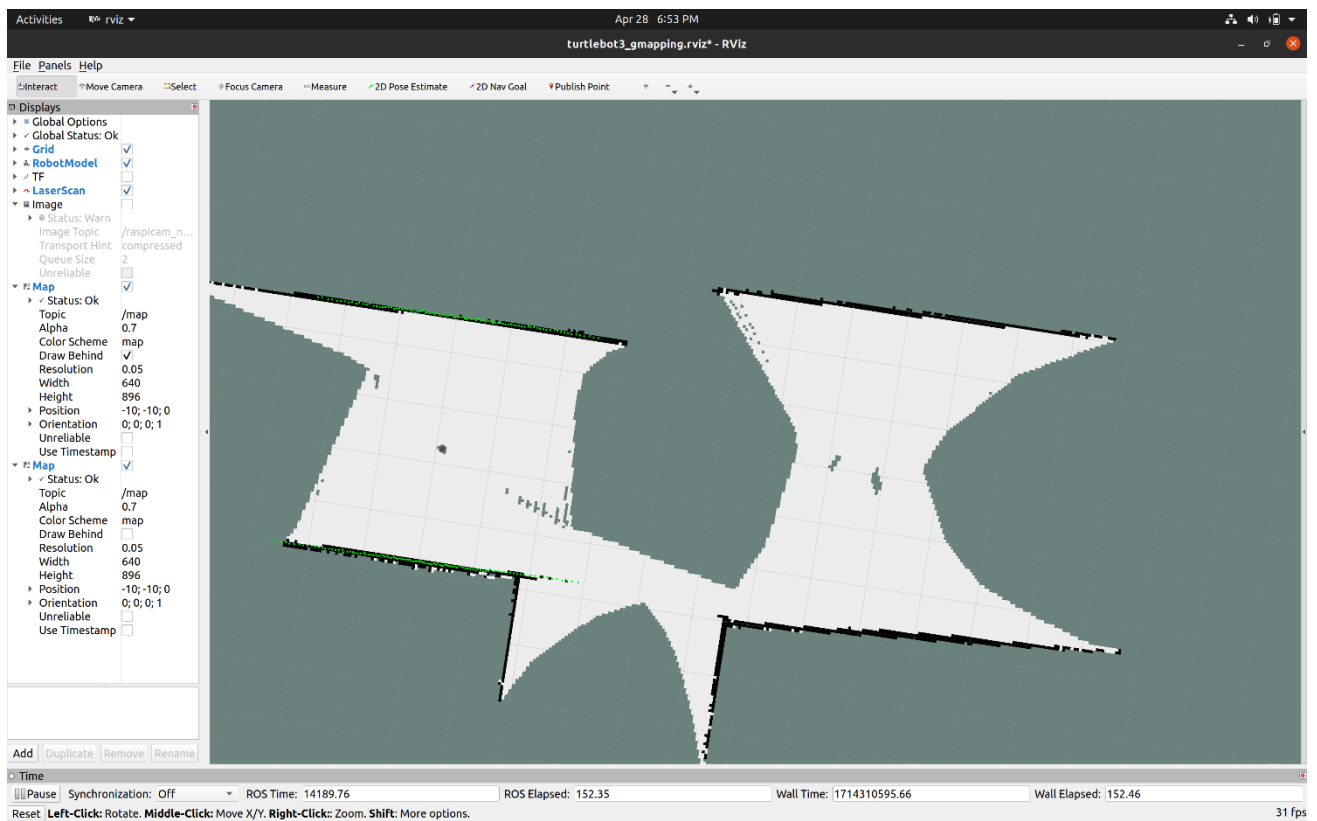
The implementation includes many tasks, and many files need to be configured. Here, I am attaching the link for the google drive. You can see all the files by referring to this link.

All videos of simulations are also included in Google Drive.

<https://drive.google.com/drive/folders/1ohCWMKV1A4odGJG-IP2Cig52LwsrmW0X?usp=sharing>

Here is which folder includes what:-

- A* Path Planning includes details about A* algorithm.
- AMCL includes files about localization algorithm.
- Commands include all the commands that are required to run this project.
- Autonomous Navigation and Obstacle avoidance include simulation videos.
- G Mapping Slam include parameter file and all other files under turtle3_slam.
- Turtle bot includes details about robot.
- Environment includes configuration files about environment.
- Documentation includes reports and other necessary documentation.



Screenshot of Rviz when the robot is moving and the map is being generated.

```
/home/arash/catkin_ws/src/turtlebot3/turtlebot3_slam/launch/turtlebot3_slam.launch http://localhost:11311
/home/arash/catkin_ws/src/turtlebot3/turtlebot3_slam/launch/turtlebot3_slam.launch http://localhost:11311 187x54
m_count 25
Average Scan Matching Score=136.798
neffs= 100
Registering Scans:Done
update frame 78
update ld=2.83973e-06 ad=4.17942e-06
Laser Pose= 6.00549 21.4113 -0.0998594
m_count 26
Average Scan Matching Score=136.795
neffs= 100
Registering Scans:Done
update frame 81
update ld=2.82933e-06 ad=4.16401e-06
Laser Pose= 6.00549 21.4113 -0.0998635
m_count 27
Average Scan Matching Score=136.764
neffs= 100
Registering Scans:Done
update frame 84
update ld=2.81915e-06 ad=4.14894e-06
Laser Pose= 6.00548 21.4113 -0.0998677
m_count 28
Average Scan Matching Score=136.791
neffs= 100
Registering Scans:Done
update frame 87
update ld=2.80922e-06 ad=4.13423e-06
Laser Pose= 6.00548 21.4113 -0.0998718
m_count 29
Average Scan Matching Score=136.783
neffs= 100
Registering Scans:Done
update frame 90
update ld=2.79951e-06 ad=4.11983e-06
Laser Pose= 6.00548 21.4113 -0.0998759
m_count 30
Average Scan Matching Score=136.818
neffs= 100
Registering Scans:Done
update frame 93
update ld=2.78998e-06 ad=4.10571e-06
Laser Pose= 6.00548 21.4113 -0.09988
m_count 31
Average Scan Matching Score=136.809
neffs= 100
Registering Scans:Done
update frame 96
update ld=2.78065e-06 ad=4.09187e-06
Laser Pose= 6.00547 21.4113 -0.0998841
m_count 32
Average Scan Matching Score=136.803
neffs= 100
Registering Scans:Done
```

In this screenshot one can see scanning is being done and various parameters are being updated.

n count: This parameter keeps track of the number of scans that have been taken by the robot.

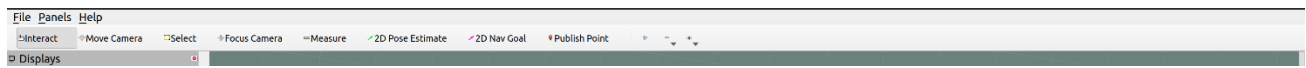
Average Scan Matching Score: This parameter is the measure of how well the most recent scan matches the robot's map.

neff: This parameter is likely the effective number of particles that are being used by the SLAM algorithm. In particle filters, a set of weighted particles is used to represent the robot's belief about its location. The effective number of particles is a measure of how diverse the set of particles is.

Laser Poses: This parameter is the location of the robot's laser scanner in the map.

```
/home/arash/catkin_ws/src/turtlebot3/turtlebot3_navigation/launch/move_base.launch http://localhost:11311
/home/arash/catkin_ws/src/turtlebot3/turtlebot3_navigation/launch/move_base.launch http://localhost:11311 187x54
[INFO] [1714310990.314984796, 14583.945000000]: Got new plan
[INFO] [1714310990.516111041, 14584.146000000]: Got new plan
[INFO] [1714310990.715988213, 14584.346000000]: Got new plan
[INFO] [1714310990.915751476, 14584.545000000]: Got new plan
[INFO] [1714310991.115901846, 14584.745000000]: Got new plan
[INFO] [1714310991.316290505, 14584.946000000]: Got new plan
[INFO] [1714310991.515923401, 14585.145000000]: Got new plan
[INFO] [1714310991.715975539, 14585.345000000]: Got new plan
[INFO] [1714310991.915981994, 14585.545000000]: Got new plan
[INFO] [1714310992.116548885, 14585.745000000]: Got new plan
[INFO] [1714310992.316532130, 14585.945000000]: Got new plan
[INFO] [1714310992.516818755, 14586.145000000]: Got new plan
[INFO] [1714310992.717160480, 14586.345000000]: Got new plan
[INFO] [1714310992.916538450, 14586.545000000]: Got new plan
[INFO] [1714310993.117439391, 14586.745000000]: Got new plan
[INFO] [1714310993.317188227, 14586.945000000]: Got new plan
[INFO] [1714310993.517707445, 14587.146000000]: Got new plan
[INFO] [1714310993.717698046, 14587.345000000]: Got new plan
[INFO] [1714310993.917770200, 14587.545000000]: Got new plan
[INFO] [1714310994.117566447, 14587.745000000]: Got new plan
[INFO] [1714310994.317281134, 14587.945000000]: Got new plan
[INFO] [1714310994.517325258, 14588.145000000]: Got new plan
```

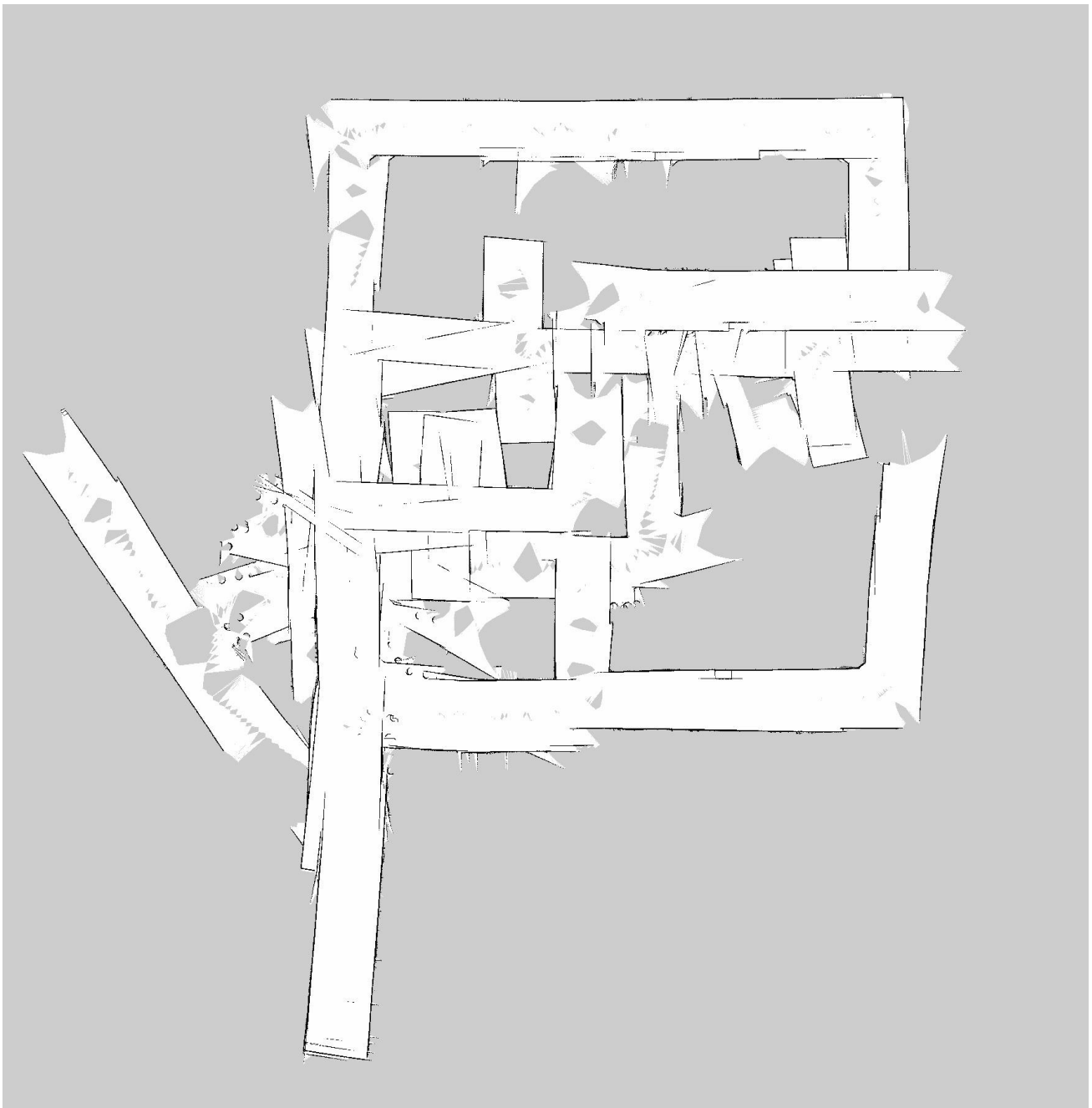
This screenshot shows that robot gets new goal.



2D Pose estimate helps to localize the robot at the current position.

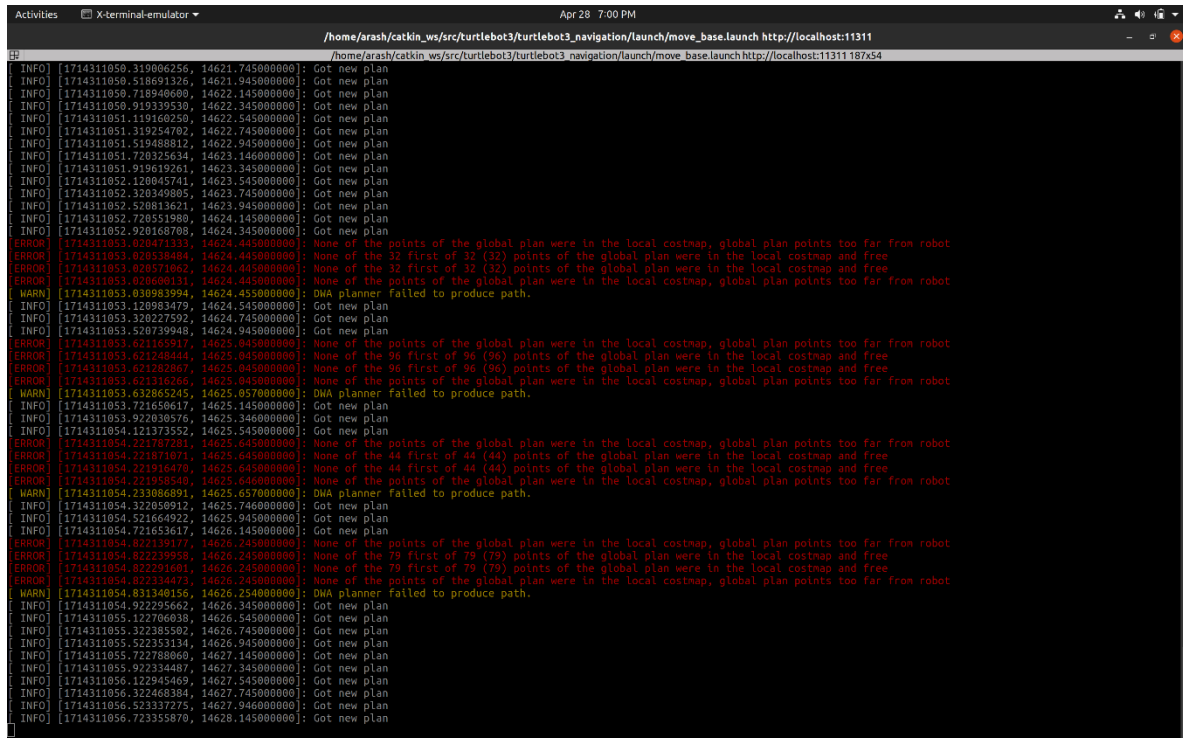
2D Nav Goal used to give goal to robot.

Result



Issues

- Robot is not able to localize itself always. When it crosses the same place after some time, generally, it maps it to the right place in a 2D map. But sometimes, it is not able to map it to the right place. Instead, it maps to a new place. Due to this, it becomes difficult to completely map the entire environment.
- After giving goal I was getting this error.



```
Activities X-terminal-emulator Apr 28 7:00 PM
/home/arash/catkin_ws/src/turtlebot3/turtlebot3_navigation/launch/move_base.launch http://localhost:11311
/home/arash/catkin_ws/src/turtlebot3/turtlebot3_navigation/launch/move_base.launch http://localhost:11311 187x54

[INFO] [1714311050.319006256, 14621.745000000]: Got new plan
[INFO] [1714311050.518691326, 14621.945000000]: Got new plan
[INFO] [1714311050.718940600, 14622.145000000]: Got new plan
[INFO] [1714311050.919339530, 14622.345000000]: Got new plan
[INFO] [1714311051.119160250, 14622.545000000]: Got new plan
[INFO] [1714311051.319254702, 14622.745000000]: Got new plan
[INFO] [1714311051.519488812, 14622.945000000]: Got new plan
[INFO] [1714311051.720325634, 14623.146000000]: Got new plan
[INFO] [1714311051.919619261, 14623.345000000]: Got new plan
[INFO] [1714311052.120845741, 14623.545000000]: Got new plan
[INFO] [1714311052.320349805, 14623.745000000]: Got new plan
[INFO] [1714311052.520813621, 14623.945000000]: Got new plan
[INFO] [1714311052.720551980, 14624.145000000]: Got new plan
[INFO] [1714311052.920168708, 14624.345000000]: Got new plan
[ERROR] [1714311053.020471338, 14624.445000000]: None of the points of the global plan were in the local costmap, global plan points too far from robot
[ERROR] [1714311053.020538484, 14624.445000000]: None of the 32 first of 32 (32) points of the global plan were in the local costmap and free
[ERROR] [1714311053.020571062, 14624.445000000]: None of the 32 first of 32 (32) points of the global plan were in the local costmap and free
[ERROR] [1714311053.020606131, 14624.445000000]: None of the points of the global plan were in the local costmap, global plan points too far from robot
[WARN] [1714311053.030853994, 14624.455000000]: DWA planner failed to produce path.
[INFO] [1714311053.320227592, 14624.745000000]: Got new plan
[INFO] [1714311053.520739948, 14624.945000000]: Got new plan
[ERROR] [1714311053.621165917, 14625.045000000]: None of the points of the global plan were in the local costmap, global plan points too far from robot
[ERROR] [1714311053.621248444, 14625.045000000]: None of the 96 first of 96 (96) points of the global plan were in the local costmap and free
[ERROR] [1714311053.621282807, 14625.045000000]: None of the 96 first of 96 (96) points of the global plan were in the local costmap and free
[ERROR] [1714311053.621316266, 14625.045000000]: None of the points of the global plan were in the local costmap, global plan points too far from robot
[WARN] [1714311053.632865245, 14625.057000000]: DWA planner failed to produce path.
[INFO] [1714311053.721658617, 14625.145000000]: Got new plan
[INFO] [1714311053.922038376, 14625.346000000]: Got new plan
[INFO] [1714311054.121373552, 14625.545000000]: Got new plan
[ERROR] [1714311054.221787201, 14625.645000000]: None of the points of the global plan were in the local costmap, global plan points too far from robot
[ERROR] [1714311054.221871071, 14625.645000000]: None of the 44 first of 44 (44) points of the global plan were in the local costmap and free
[ERROR] [1714311054.221916470, 14625.645000000]: None of the 44 first of 44 (44) points of the global plan were in the local costmap and free
[ERROR] [1714311054.221958540, 14625.646000000]: None of the points of the global plan were in the local costmap, global plan points too far from robot
[WARN] [1714311054.233086891, 14625.657000000]: DWA planner failed to produce path.
[INFO] [1714311054.322050912, 14625.746000000]: Got new plan
[INFO] [1714311054.521664922, 14625.945000000]: Got new plan
[INFO] [1714311054.721452617, 14626.145000000]: Got new plan
[ERROR] [1714311054.822139117, 14626.245000000]: None of the points of the global plan were in the local costmap, global plan points too far from robot
[ERROR] [1714311054.822239958, 14626.245000000]: None of the 79 first of 79 (79) points of the global plan were in the local costmap and free
[ERROR] [1714311054.822291601, 14626.245000000]: None of the 79 first of 79 (79) points of the global plan were in the local costmap and free
[WARN] [1714311054.831348150, 14626.254000000]: DWA planner failed to produce path.
[INFO] [1714311054.922295662, 14626.345000000]: Got new plan
[INFO] [1714311055.122706038, 14626.545000000]: Got new plan
[INFO] [1714311055.322385502, 14626.745000000]: Got new plan
[INFO] [1714311055.522353134, 14626.945000000]: Got new plan
[INFO] [1714311055.722788060, 14627.145000000]: Got new plan
[INFO] [1714311055.922334487, 14627.345000000]: Got new plan
[INFO] [1714311056.122945469, 14627.545000000]: Got new plan
[INFO] [1714311056.322468394, 14627.745000000]: Got new plan
[INFO] [1714311056.523337275, 14627.946000000]: Got new plan
[INFO] [1714311056.723355870, 14628.145000000]: Got new plan
```

I also tried implementing **A* algorithm** for path planning. Here is some details about this algorithm.

A* (pronounced “A-star”) is a graph traversal and pathfinding algorithm. It’s widely used due to its completeness, optimality, and optimal efficiency. Given a weighted graph, a source node, and a goal node, A* finds the shortest path (with respect to the given weights) from the source to the goal.[13][14]

Key Features:

Heuristic-Based: A* uses a heuristic function to guide its search efficiently.

Smart Exploration: It prioritizes paths that seem to lead closer to the goal.

Optimal Solution: A* guarantees finding the shortest path if one exists.

Algorithm Steps:

1. Initialize an open list and a closed list.
2. Put the starting node on the open list.
3. While the open list is not empty:
 - Find the node with the lowest “f” value (sum of “g” and “h”) on the open list.
 - Process that node.
 - Generate successors (neighbors) of the current node.
 - Compute “g” (movement cost) and “h” (estimated cost to reach the goal) for each successor.
 - Update the successor’s “f” value.
 - If a successor is the goal, stop the search.
 - Otherwise, continue exploring successors.

The path can be reconstructed by following parent pointers from the goal node back to the start.

2.Integration with Gmapping SLAM:

Gmapping SLAM:

Gmapping is a popular SLAM algorithm for creating 2D maps using laser scan data.

It estimates the robot’s pose (position and orientation) while mapping the environment.

How A Works with Gmapping*:

Gmapping provides the map of the environment.

A* uses this map as the graph for path planning.

The starting position is the robot’s current pose, and the goal position is the desired destination.

A* computes the shortest path on the map, considering obstacles and free space.

The resulting path guides the robot’s movement.

Input for A with Gmapping:

Map: The occupancy grid map generated by Gmapping.

Start Position: The robot’s current pose.

Goal Position: The desired destination.

Heuristic Function: A* requires a heuristic function (“h”) to estimate the cost from a cell to the goal. Common heuristics include Manhattan distance, diagonal distance, or Euclidean distance.

Conversion:

The map cells become nodes in the graph. The cost between adjacent cells (edges) is based on the map data (obstacles, free space, etc.). The heuristic function estimates the remaining cost from a cell to the goal.

Future aspects

I will try resolve issues related to maps.

References –

1. [Path planning for active SLAM based on deep reinforcement learning under unknown environments | Intelligent Service Robotics \(springer.com\)](#)
2. [Build Occupancy Map from 3-D Lidar Data Using SLAM - MATLAB & Simulink - MathWorks India](#)
3. [1811.04825 \(arxiv.org\)](#)
4. [Build a Map from Lidar Data Using SLAM - MATLAB & Simulink - MathWorks India](#)
5. [What is LiDAR? | IBM](#)
6. [What Is SLAM \(Simultaneous Localization and Mapping\) – MATLAB & Simulink - MATLAB & Simulink \(mathworks.com\)](#)
7. [gmapping - ROS Wiki](#)
8. [Simultaneous localization and mapping - Wikipedia](#)
9. [amcl - ROS Wiki](#)
10. [TurtleBot3 \(robotis.com\)](#)
11. [How to Launch the TurtleBot3 Simulation With ROS – Automatic Addison](#)
12. [What is Dijkstra's Algorithm? | Introduction to Dijkstra's Shortest Path Algorithm - GeeksforGeeks](#)
13. [A - Wikipedia](#)
14. [Introduction to the A* Algorithm \(redblobgames.com\)](#)