Reconnaissance Robot

EECE 5550: Mobile Robotics Final project
Prof. Michael Everett

Team:
1. Abhishek Uddaraju
2. Atharva Jamsandekar
3. Yijian Huang
4. Jiatong Wu
5. Bhanu Prasad AJ

Table of Contents:

Overview

The Reconnaissance Bot is a Navigation stack based on frontier exploration of an unknown environment.

The package is able to explore the environment after integration with a SLAM algorithm that provides the map and transform tree.

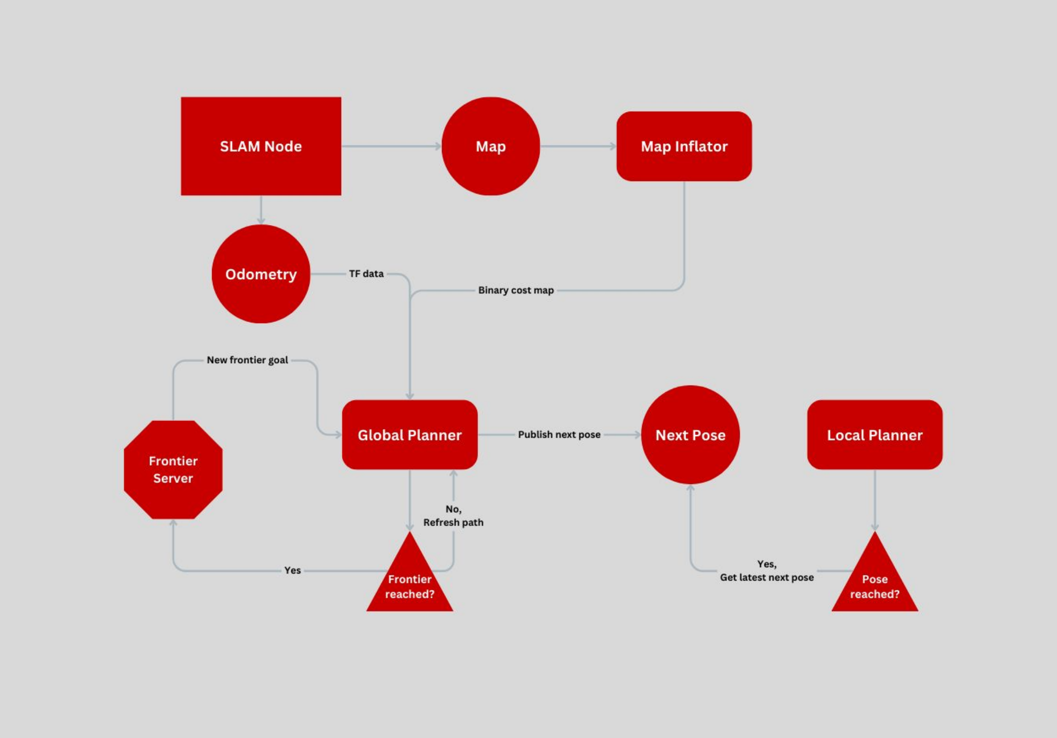
It has the additional functionality of identifying points of interest in the environment.

Applications:

1) Unsafe disaster-struck Environment Survey
2) Reconnaissance of Military-active regions

The Package has been tested on the Turtebot3 burger platform.

Architecture



Frontier Exploration[1]

Overview:

1. Identify candidates
2. Clustering the candidate points
3. Scoring the clusters
4. Choosing a goal

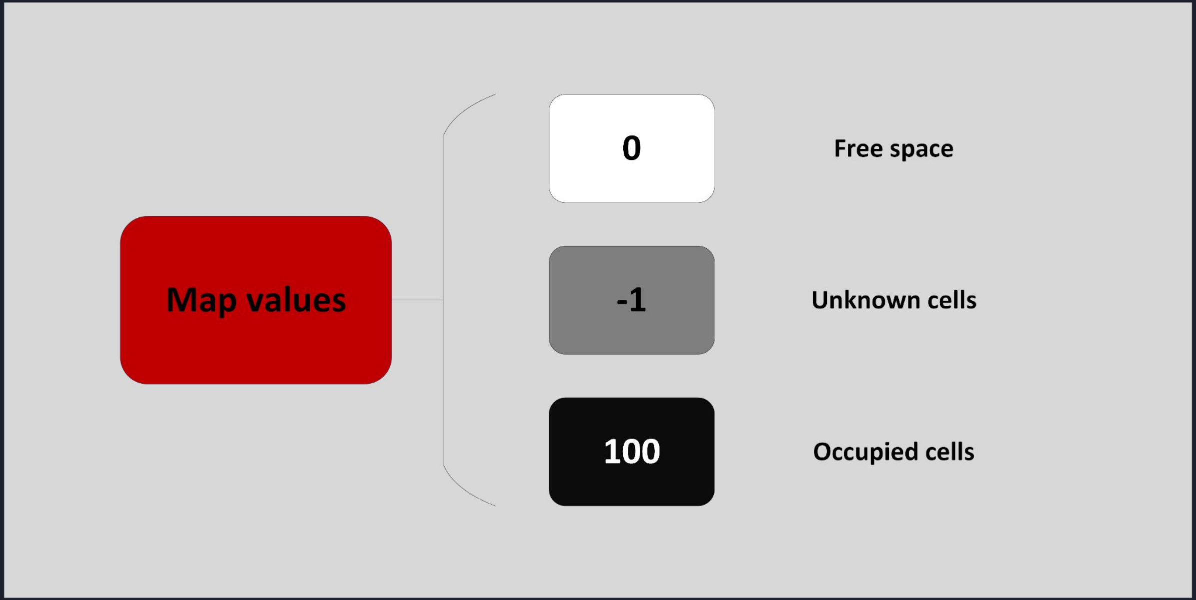
[1]Erkan Uslu, Furkan Çakmak, Muhammet Balcılar, Attila Akinci, Mehmet Fatih Amasyalı, and S. Yavuz, “Implementation of frontier-based exploration algorithm for an autonomous robot,” Sep. 2015, doi: https://doi.org/10.1109/inista.2015.7276723.

Frontier Exploration

Identifying candidates:

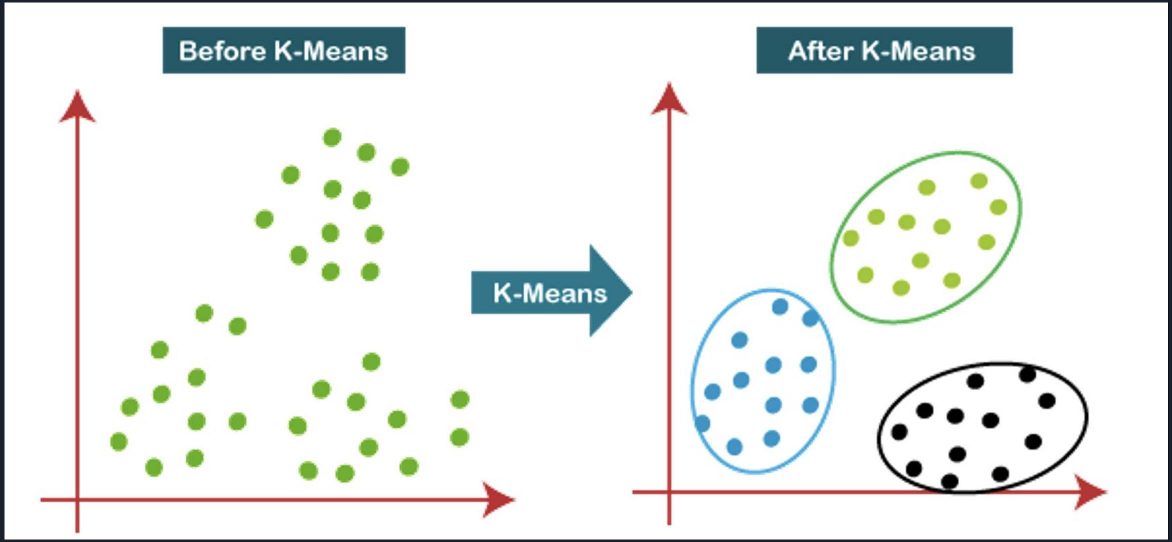
1. Move a kernel through the map
2. If any occupied cells are detected in the neighbourhood, reject the cell
3. The eligible candidates will have a ratio of free cells to unknown cells

Frontier Exploration

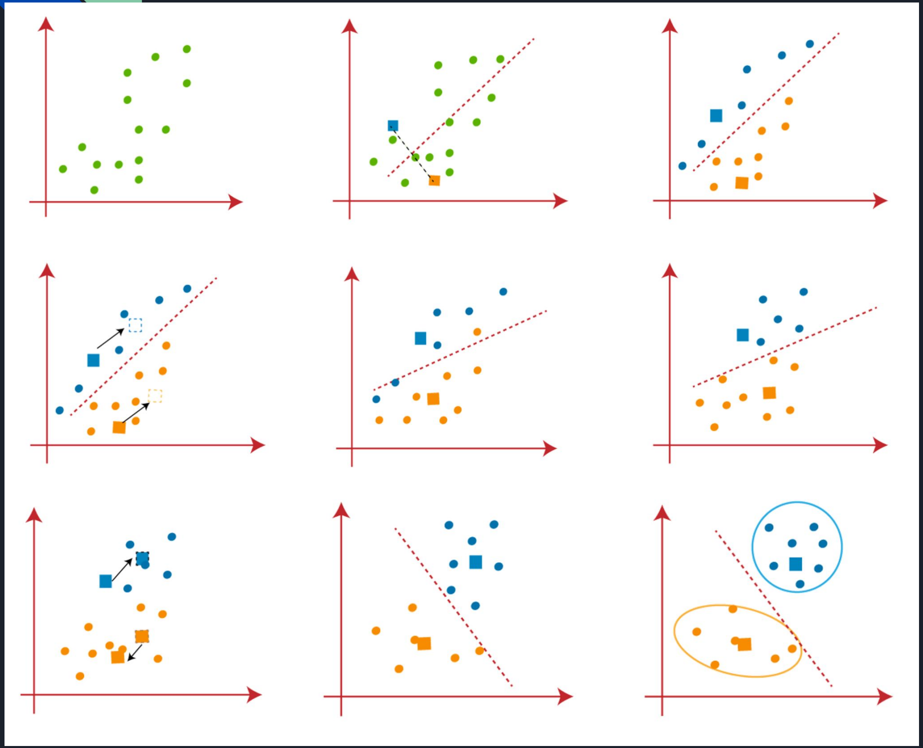


Frontier Exploration — K Means Clustering

- Unsupervised learning algorithm
- Centroid - based algorithm
- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data points to its closest k-center.
    - Those data points which are near to the particular k-center, create a cluster.



Frontier Exploration — K Means Clustering



Step 1: Select the number K to decide the numbers of clusters. (K=8)

Step 2: Select random K points in the grid world.

Step 3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step 4: Calculate the variance and place a new centroid of each cluster.

Step 5: Repeat ‘Step 3’, which re-assign each data point to the new closest centroid of each cluster.

Step 6: I any reassignment occurs, then go to ‘Step 4’, else FINISH.

Frontier Exploration

Scoring candidates:

1. Euclidean distance to the centroid
2. Mass of the Cluster

Choosing candidates:

1. Randomly choosing one candidate from the chosen cluster as the goal

Frontier Exploration

Challenges:

1. Performing boolean operations through nested for loops is expensive
2. Took more than 16 seconds to run through a 384 x 384 map!

Frontier Exploration

Solution:

1. JAX to the rescue!
   a. Used the built-in convolution function that uses FFT to speed up the process
2. Ran it in a parallel pipeline to get next frontier before the robot reaches current goal

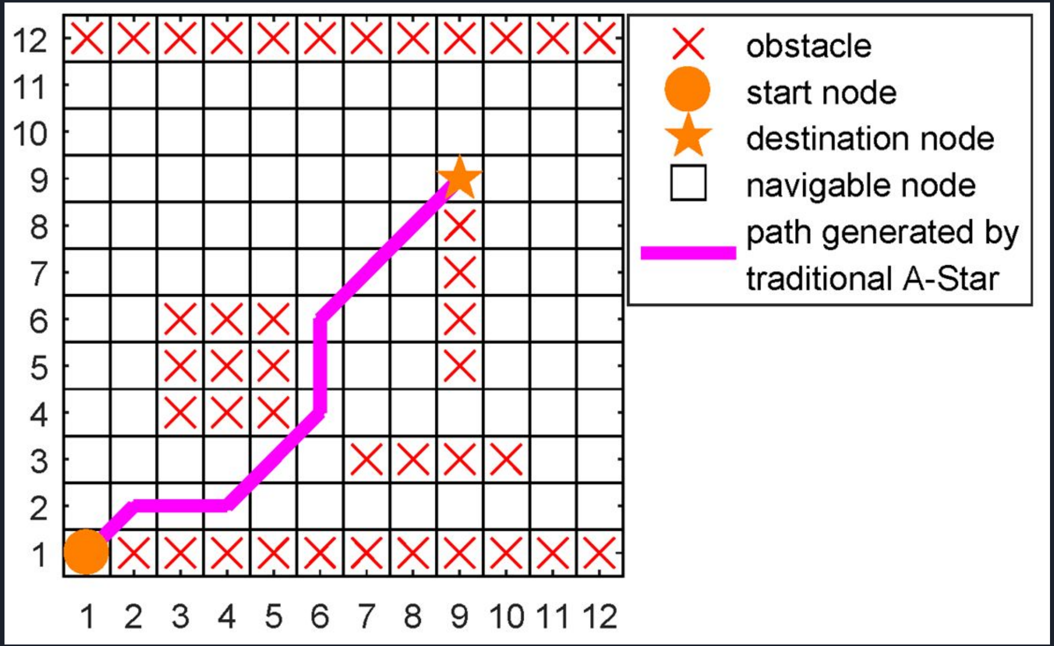
Global Planning: A*

- A kind of path searching algorithm
- An extension of Dijkstra's algorithm
- Node to node
- Completeness, optimality, and optimal efficiency

- Input: map(grid), start position, goal
- Output: path(list of position)

The value of grids:

100 means obstacle
0 means clear grids
- 1 means unknown grids

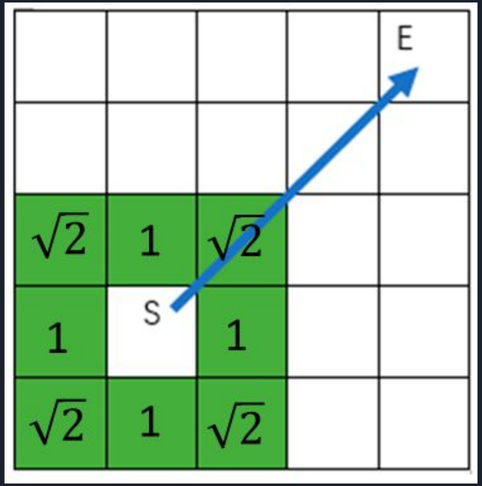


Complexity : $O(\log h^*(x))$

where h* is the optimal heuristic, the exact cost to get from x to the goal.

Global Planning: A*

How it works?



Move cost : distance from start point

Heuristic function: Hypotenuse of triangle
C=√(a^2+b^2)

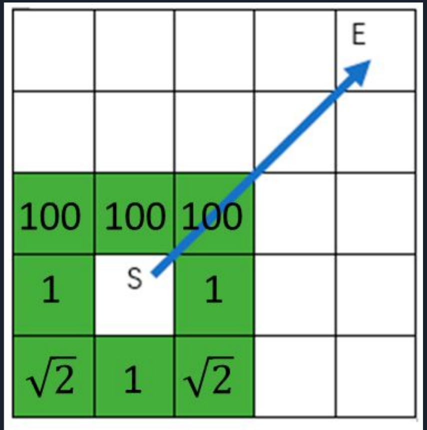
Cost sum : heuristic function + move cost

For top, cost = 1+5 =6
For top right, cost = √2 + 3 √2 = 4 √2

Global Planning: A*

Avoid Obstacles

Move cost : give obstacle a large cost like 100

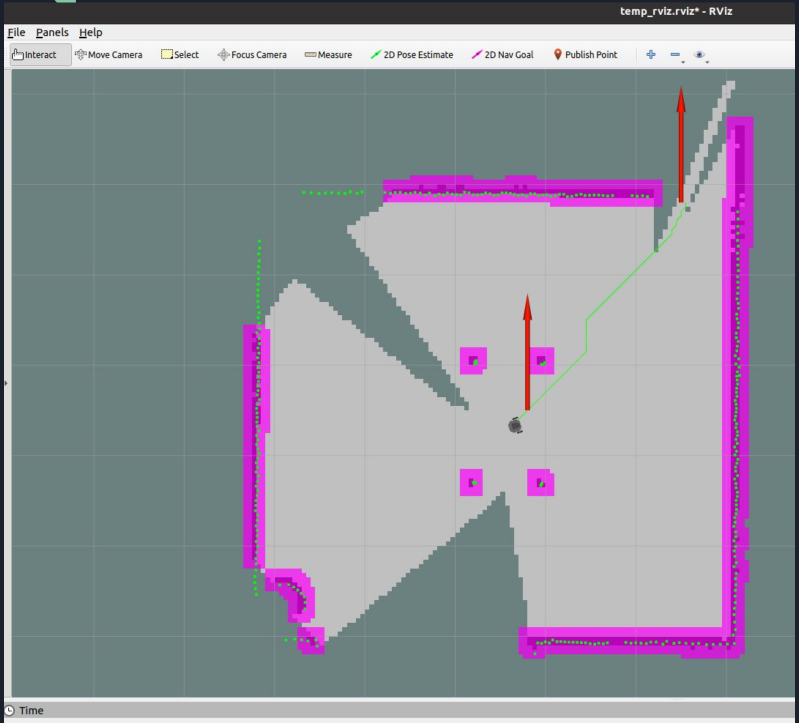


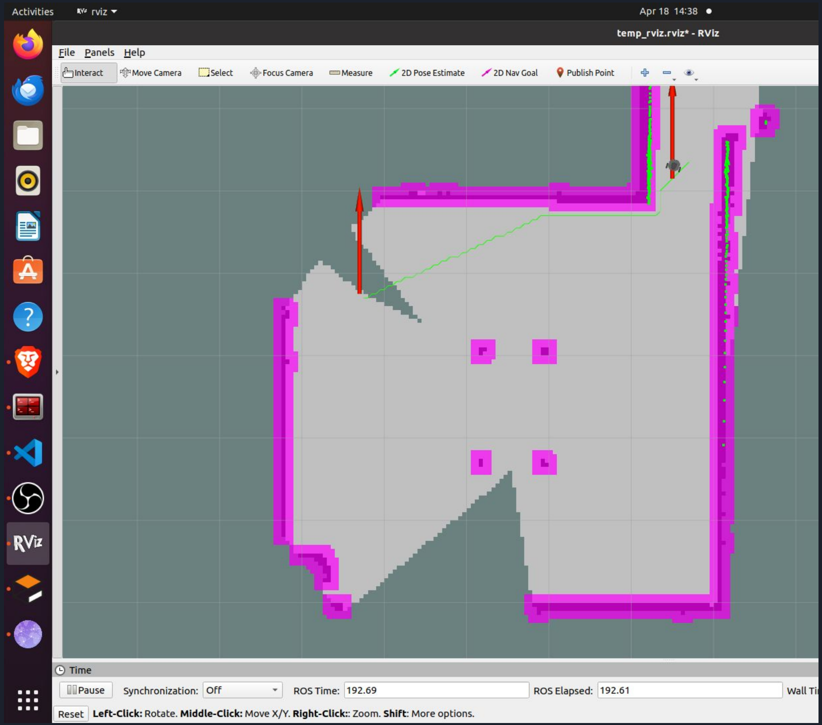
Drawback:

Space complexity
Stores all generated nodes in memory
But for our grid it is not big map

When facing a far goal, it may calculate a path with a long time.
This may cause robot stop or crash

Global Planning: A*





Local Planning: MPPI

TurtleBot Limits:

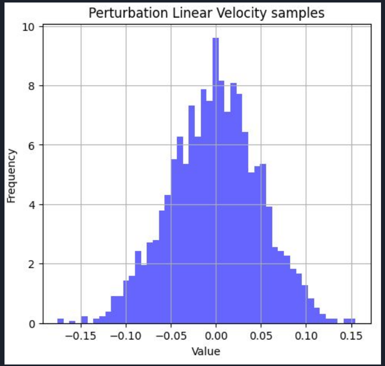
- Linear velocity Limits (m/s) : [-0.26, 0.26]
- Angular Velocity Limits (rad/s): [-2.84, 2.84]

MPPI Parameters:

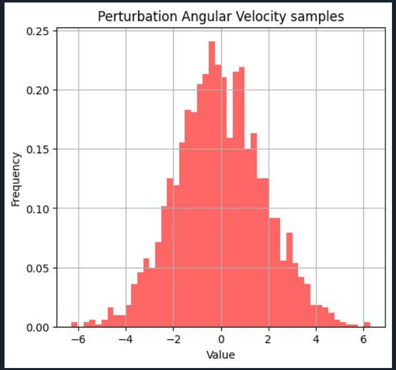
- Number of Rollouts = 100
- Number of Steps = 20
- Nominal Velocity = 0.8
- Nominal Angular Velocity = [0, pi]

Maximum Frequency of the Local planner: 12 Hz

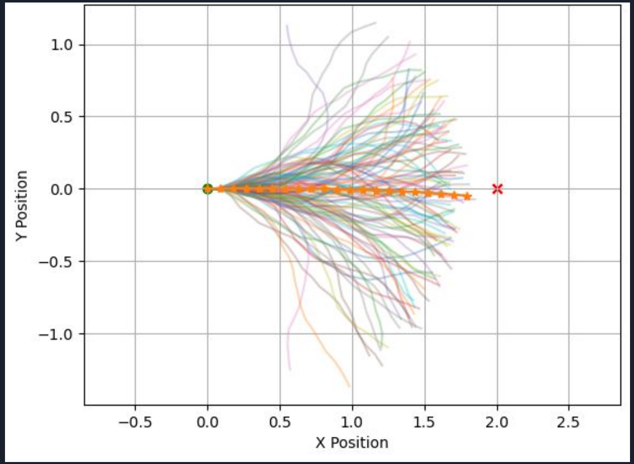
We are running it at 10 Hz frequency, with freshly updated map.



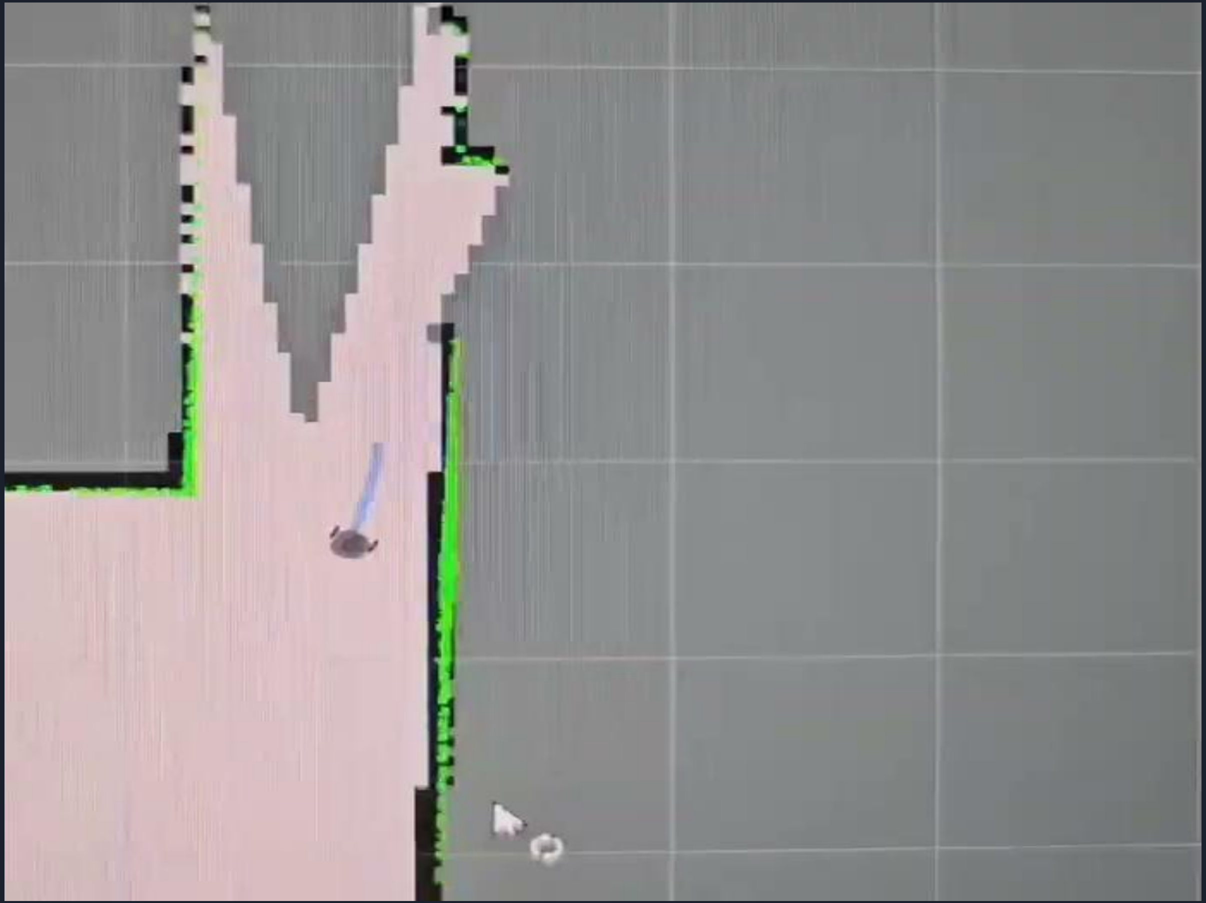
Sigma : 0.05 m/s

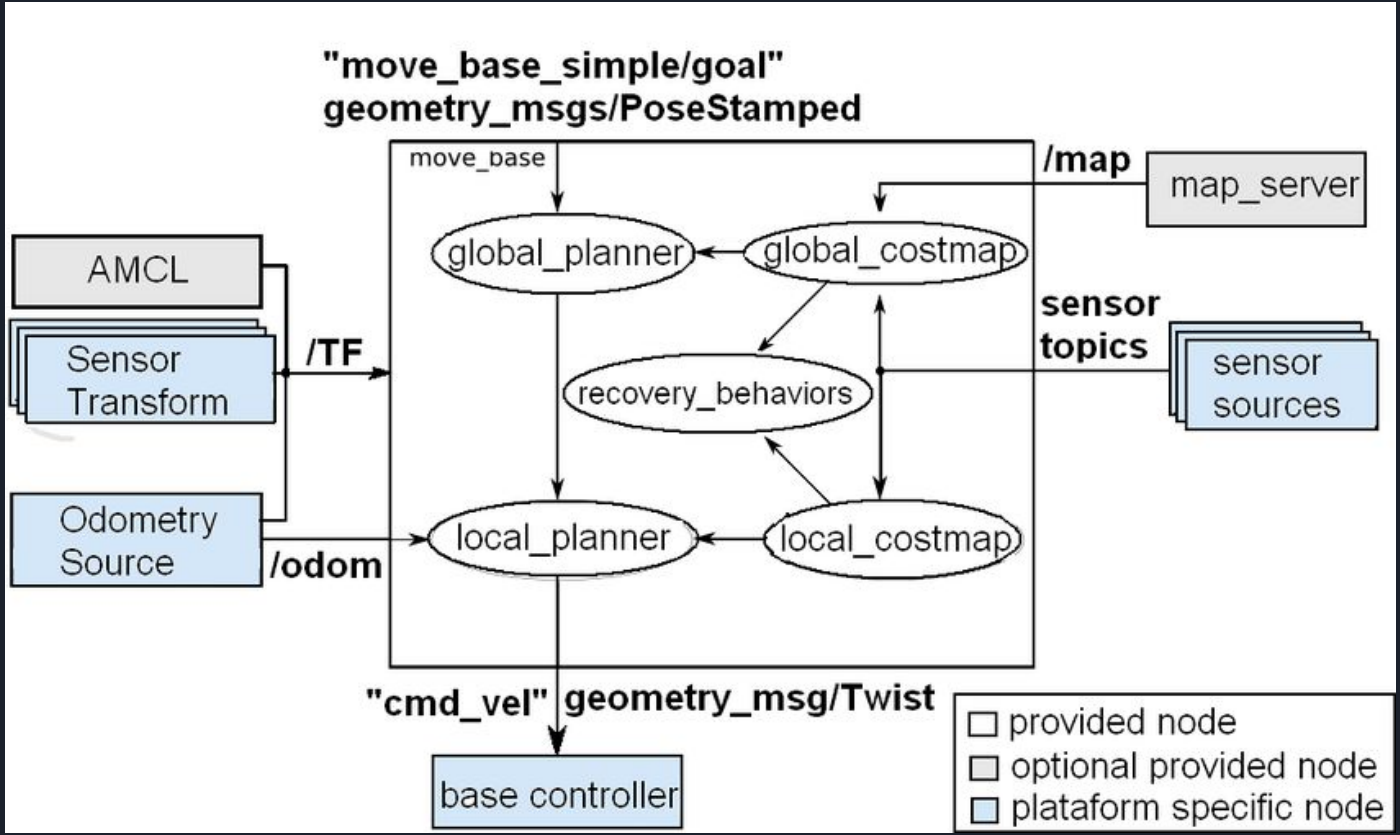


Sigma : 0.3 rad/s



MPPI Demo




"move_base_simple/goal"
geometry_msgs/PoseStamped
move_base
/map
map_server
AMCL
global_planner
global_costmap
sensor
topics
Sensor
Transform
/TF
recovery_behaviors
sensor
sources
Odometry
Source
local_planner
local_costmap
/odom
"cmd_vel"
geometry_msg/Twist
provided node
optional provided node
plataform specific node
base controller

SLAM and April tag detection

SLAM:

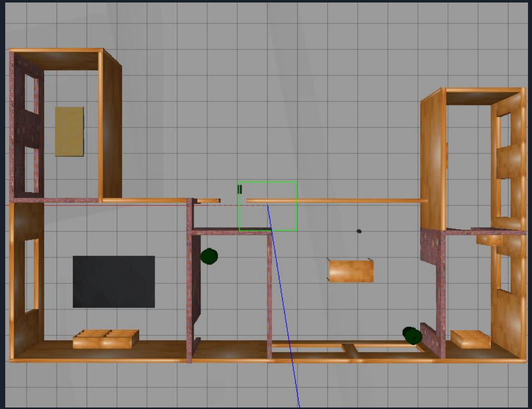
- Gmapping is used for performing Simultaneous Localization and Mapping.
- Part of Turtlebot3_slam ROS package.
- **Will attempt to transform each incoming scan into the odom (odometry) tf frame.**

April tag detection:

/tag_detections: the same information as provided by the /tf topic but as a custom message carrying the tag ID(s), size(s) and geometry_msgs/PoseWithCovarianceStamped pose information (where plural applies for tag bundles). This is always published

Results

Simulation Environment:



Real-world Environment:

Reconnaissance Robot Demo

Gazebo: Turtlebot House

Reconnaissance Robot Demo

Location: Hurtig Hall
Date: 04/17/24

Future Scope & Extensions:

- AprilTag Pose seeking
- Scalable to a Larger Environment
- Victim Detection Vision Pipeline to replace AprilTag Detection
- AprilTag Pose Estimation using Factor Graphs

Github: https://github.com/arjunjyothieswarb/MR_FinalProject/