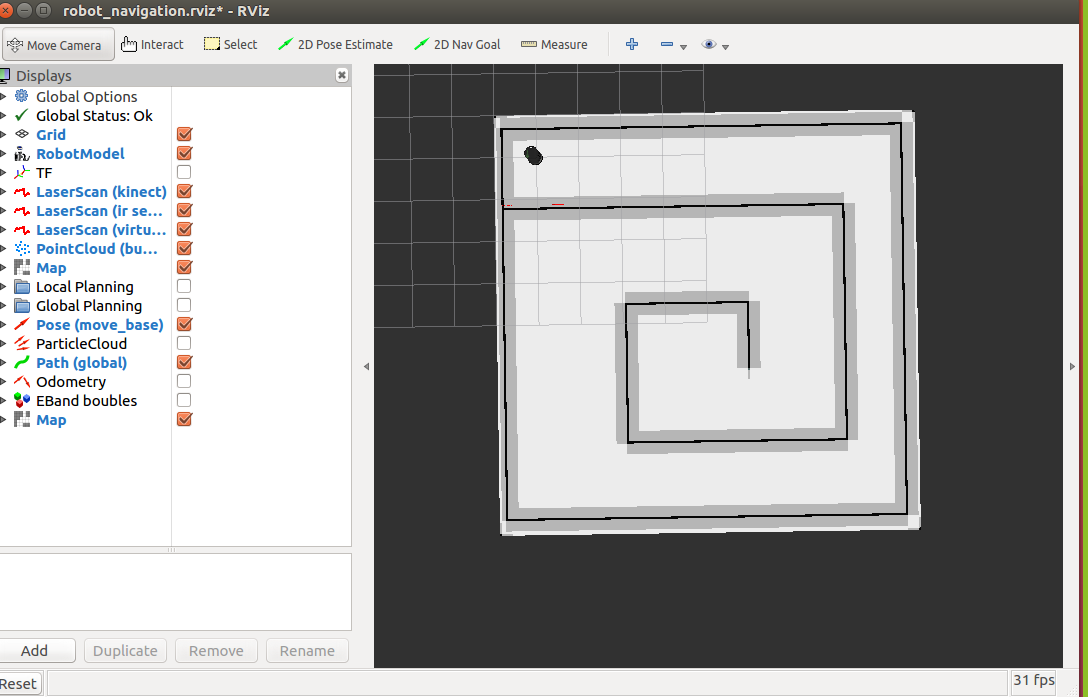
# **Report – Assignment 2**

1. Map inflation:

Since the map can never reflect the where exactly the obstacles are present in the real world it is better to increase the size of the obstacle by some ratio to overcome this issue.

Inflation ratio for the given case is kept constant, but we can create a cost matrix around the obstacles to give more changes of path exploration in the narrow path environment.

For this assignment the inflation ratio used is = size of the robot / resolution, and map is inflated on all side of the obstacles. The inflated map it published in the topic /map\_inf



The greyish area is the inflated area, and the dark black portion is the original obstacle area given in the map.

pesudo code:

1. Convert the occupancy grid data in the matrix format using:

grid[x][y]=self.aug\_map[y\*self.world\_width+x]

1. Inflated the map by adding 100 , from x till inflation ration and from y till inflation ratio
2. Recompute the index position and save it in new\_map and publish the map as rostopic
3. Collision Checker:

For the given x,y value we need to find if the point is on a obstacle in the map. For this using, the previous generated grid matrix and converting the point x and y to grid by dividing it by resolutions.

1. Base Planner:
   1. Used A\* algorithm to implement the search in the discrete state space with deterministic action
   2. Heuristics used for A\* is Manhattan distance
   3. Used heap priority queue to avoid manual operation of reshuffling based on the lowest cost
   4. Pesudo code:

* Initialize openheap to start pos and keep the tracked of the visited node in a dictionary
* Repeat the following process till openheap is empty or goal is reached
* Generate the next possible state based on the list of actions provide (for this case: move Forward, Left and Right)
* Compute the cost as f(n)=g(n)+h(n) , g(n) = Total cost required to reach the node, H(n) estimated cost to reach the goal from the current node
* If the next state generated is not in the visited node list or if it is in visited node and it total cost is more than the current computed cost and also doesn’t falls on obstacles then add its cost: Then add it to the openHeap
  1. Action is deterministic and state is discrete, thus A\* tries to ensure that past exploration and keeps the search spread towards the goal (heuristics) A\* balances the two as it moves from the starting point to the goal
  2. For this planner, inflated map was used
  3. Attached the control files, file format is <map\_name>\_goal\_coordinates , example : task1\_maze0\_0 – used A\* on maze0.png with goal as 3,3

RUN COMMAND : python base\_planner.py - -goal ‘3,3’ – com 0

1. Deterministic Action but continuous state:
   1. Used Hybrid A\* to implement the search to goal and to generate a path from start to goal
   2. Heuristics used for A\* is Euclidean distance
   3. Possibility of the next state is also most 10 \* 10
   4. There is some bug in the code or in the configuration setting, for some maze it can reach the goal and for some it is not able to find the path properly.

Like for maze1 goal position 5,5 – it finds some weird path but if I divide the goals into sub goal it can reach the position. I am not sure why that happens. I have tried with multiple inflation ratio.

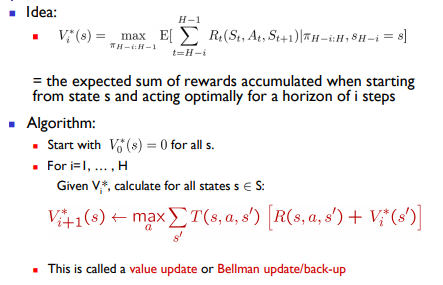
* 1. Pesudo code:
* Like A\* code
* Call: get\_continous\_state to get the next continuous state
* Cost\_matix – Storing the direction of all possible exploration
* Get\_next() – Gets the next position of the candidate position
* Candidate position is calculated using the motion\_predit fuction
* Hybrid A\* included the motion dynamics of the robot

RUN COMMAND : python hybrid\_planner.py - -goal ‘3,3’ – com 0

Graphical user interface, application

Description automatically generated

1. Probabilistic action and discrete state
   1. Used the value iteration function to get the path from given start to end
   2. Rewards are classified as – 100 for the goal position in the grid, -10 for the obstacle in the grid and for each step (rest all the value in the grid world) is set to the -0.001 (very minimum value)
   3. Value iteration function run till the new and the old value function, whose maximum difference is less than given delta value
   4. For certain maze, gamma values were changed to reach the goal position control files attached are generated from a fixed gamma position.



RUN COMMAND: python mdp.py - -goal ‘3,3’ – com 0

Results:

1. Maze0 – All planner executed the position
2. Maze1 – A\* is not working properly
3. Maze2 - A\* is not working properly, MDP with another gamma value
4. Maze3- All the goal position are executed by the planner
5. Com1- A\* only runs for goal 2,10; Rest both the planner finds the path
6. Hybrid A\* I tried with lesser next state Possibility (similar to that of A\*) it runs for all the cases but for the given one it fails in most of the cases.