**Path Planning for Mobile Robot with trolleys**

**in Indoor Environment**

**Introduction**

Path planning is the method of finding the optimal path from one point to another in space. It optimizes the path between source and destination by determining shortest path between them. It is sometimes also termed as motion planning as it helps to decide the motion of any object in an environment. An object can be a robot which is autonomous in nature as it makes use of the path finding algorithm to determine its traversing points in space. Such a robot is referred to as mobile robot. Path planning can also be defined as the process of breaking down a desired path into number of iterative steps to make discrete motions to optimize some entities. Here we address Path Planning in a static environment for non-holonomic car like robot, with trolleys.

**Problem Formulation**

The Path Planning problem can be stated as follows: Given a map ***M****,* find the cost-optimal path from a vehicle’s current location ***agv\_initial*** including the pose*,* to a goal location ***goal*** subject to the constraints given by ***Vehicle and the Trailer Kinematics***. The discrete grid map ***M*** is an occupancy grid of resolution 0.05 obtained from /map\_server node in ROSconsisting of values – [0, -1, 100] where, 0 – free space, -1 – unexplored region and 100 – obstacle.

The Path generated should be optimal (least cost), traversable (considering vehicle constraints), collision free (from static obstacles) and inbound collision free (collision among trolleys and robot).

**Proposed Approach**

***A\**** algorithm, which is limited to discrete state spaces is used for generating a *Cost Map* i.e. a grid, with each cell having a cost to reach the goal. Along with generating a cost map it’s also used to identify whether a goal is ‘Reachable’ or ‘Not Reachable’.

***Hybrid A\**** algorithm is used for generating a continuous path in the region searched by A\*.

The continuous path is a result of the kinematic equations []

Each node *n* of the search graph i.e. the state of the robot is completely defined by,

*robo\_state = (x, y, psi (orientation), g, h, f, v, w, vr, vl, num\_trolley, trolley\_list )*

trolley\_list – is a list of trolley states, of length equal to num\_trolley

*trolley\_state = (x, y, psi, index)*

Parameters considered for path planning here are,

Robot : height, width, wheel radius

Trolley : height, width, distance between trolley and robot

v - linear velocity (considered as constant)

w - angular velocity (augmented for n values in range ‘min angle’ and ‘max angle’)

***Heuristics considered***

A\* is used to search for the start position from the goal in a grid, having the cost of goal equal to zero and the neighboring expansions incremented by unit value. This helps in validating whether a goal is *Reachable* or Not.

If ‘Reachable’ then every cell in the grid is stored with value needed to reach the goal from that cell ( ***h*** ).

***g***  – distance between the start to the current cell ( Euclidean considered)

***heading\_cost*** – cost added based on the magnitude of difference in the *psi* value of current and previous states.

***collision\_cost*** – cost added in case of boundary collision and inbound collision

***forward\_cost*** – cost added by calculating if the future states are collision free

***f*** – total cost (***g + h + heading\_cost + collision\_cost + forward\_cost*** )

On every iteration of calculating next states, the list containing this states is sorted based on ***f*** value.

***Lookup Table***

Here we consider the map to have a set of anchor points (i.e. is predefined set of locations where the robot have to pass from). Since the *Map* is static the search algorithms are used and the paths are generated between these anchor points. The paths are then stored in a table and used for future references, hence reducing the computation for generation of paths on every job.

The table here is a directional weighted graph, with nodes as the Anchor points and weights as the path cost. The path returned would consider the least cost using ***Dijkstra’s Algorithm*** to reach the end node and the orientation of the start node.

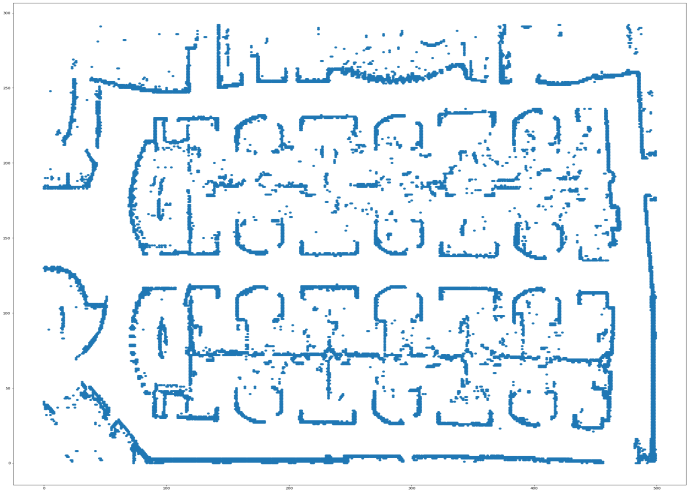
***Implementation Details & Related Diagrams***

The algorithms are implemented using the ***Numpy*** operations in Python (*version- 3.5*) for reducing the computation cost. Framework used for coding is ***Jupyter Notebook/ Google Colab.*** Output is visualized using the ***Matplotlib*** package.

Input Parameters : *map (1*D occupancy gridfrom /map\_server node in ROS), *map\_width, map\_height,*

*robot\_initial (x, y, psi), goal (x, y), num\_trolley*

*Output : Path (List of Robot states)*



Input :Map A\* Searched Map

Output Path:

