Motion Planning

1. Introduction

Motion planning is an essential part of an autonomous robot. This can involve planning the movement of various joints of a robotic arm to be able to reach an object, planning a sequence of waypoints to travel through to traverse an obstacle filled environment, planning the movements of robotic legs to walk on rough terrain. Humans and other animals are able to do this intuitively - we don't really need to maintain an explicit list of points in our head when walking through a crowded area.

For robots, there are various techniques and algorithms we can use to generate trajectories and plans of motion. This document will focus primarily on how to plan a path for moving through an environment. More specifically, given an accurate map of the surroundings and a goal point, how can we generate a sequence of locations (a path) which when traversed will safely lead us to the goal.

To get a glimpse of what path planning involves check out this video.

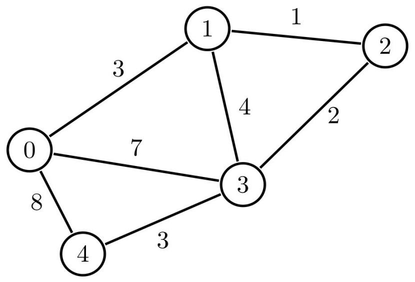
2. Graph Based Algorithm

A graph is basically a set of vertices and edges that connect the vertices. This is a heavily explored and researched topic in Mathematics and CS there are many algorithms developed to find paths within graphs. This is very useful for path planning since we can often model our surroundings using graphs and then find a path through it. Before doing any type of operations on graphs, we need a way to represent graphs in code. Some of the most common ways of doing this are adjacency lists and adjacency matrices. Follow this video guide for a basic introduction to graphs and their representations in code.

2.1 Dijkstra’s Algorithm

Djikstra’s algorithm is a graph based algorithm used to find the shortest distance between two vertices of a graph. The distance between two vertices of the graph is defined as the sum total of the weights of all the edges between them lying in the path followed. For eg. The distances along the four different paths *0 -> 1 -> 3, 0 -> 3, 0 -> 4 -> 3* and *0 -> 1 -> 2 -> 3* are from vertice *0* to vertice *3* in the graph shown below are 7

(3+4), 7 (7) ,11 (8+3) and 6 (3+1+2) respectively. Out of these 6 is the shortest possible distance between these two nodes.



To get some kind of intuition about how the algorithm works, check out these videos (1,2,3). The pseudo code for the algorithm can be written as follows -

```
Dijkstra(Graph, root_node):
{
     1.   Initialize the vertex set N
     2.   For each vertex v in Graph:
          2.1  dist[v] ← INFINITY
          2.2  prev[v] ← UNDEFINED
          2.3  append vertex v to N

     3.   dist[root_node] ← 0

     4.   While N is not empty:
          4.1  m ← vertex in N with min dist[m]
          4.2  remove m from N

          4.3  for each neighbour p of m:
               4.3.1     new_dist ← dist[m] + len(p,m)
               4.3.2     if  new_dist < dist[p]:
                         4.3.2.1   dist[p] ← new_dist
                         4.3.2.2   prev[p] ← m

     5. return dist[],prev[]
}
```

3. Sampling Based Algorithms

In this class of algorithm, we first sample random points in the environment, check if the points are valid (i.e. not inside any obstacle) and then construct a graph out of these points from which we can generate a path.

3.1 Rapidly Exploring Random Trees (RRT)

This is one of the simplest types of sampling based algorithms and has a lot of variants which make certain modifications to improve performance. The algorithm can be summarised a follows -

```
RRTPlan(obstacles, start, goal, expand_dist, max_iterations)
{
      1.  Initialize graph G with start
      2.  For max_iterations do:
          2.1.  Sample p_random from environment
          2.2.  Find nearest point p_near in G to p_random
          2.3.  Set p_new to a point at a distance expand_dist from
                p_near    and on the line joining p_near and p_random

          2.4.  If p_near ←→ p_new is valid (not intersecting any
                obstacles):
              2.4.1.  Add p_new as a vertex to G
              2.4.2.  Add p_near ←→ p_new as an edge to G

          2.5.  If goal can be connected to any vertex in G using a
                valid edge:
              2.5.1.  Add goal and the edge to G
              2.5.2.  Return G

      3.  Return G
}
```

Lets walk through this. At the very start, your graph is initialized with only the **start** point as vertex. Through the graph expansion procedure we will be building this graph until the **goal** point is reachable. In the expansion loop, we first sample a random point **p_random** in the environment. We then find the closed point **p_near** and expand it in the direction of **p_random** to produce **p_new**. If the straight line from **p_near** and **p_new** does not intersect any obstacles, we add **p_new** to the graph.

To get a better idea of what this looks like, check out this video.

3.2 Other algorithms

Sampling points to explore the environment is a very broad concept and has a ton of variations. Variants of RRT include RRT*, Informed-RRT* and EG-RRT just to name a few. The other popular class of sampling based path planning algorithms is Probabilistic Road Map (PRM). This is a very active area of research.

Further Resources

1. This lectures series by Modern Robotics
2. Lectues 50 - 66 (Section 2.0) of this series
3. This talk on Self Driving Cars by Sertac Karaman
4. Book for reference: *Planning Algorithms,* Steven LaValle. [link]
5. Open Motion Planning Library [link]