**Motivation:**

The principle system requirements that impacts the path planning team are:

Req 1: The system shall perform a search for targets in an unknown environment

1.2: The system shall navigate unknown environment, avoiding collisions with obstacles

Req 4: The system shall be autonomous

These suggest important derived requirements:

1. We are operating in an unknown environment, therefore, re-planning will be necessary as obstacles are found
2. Therefore, computational load must be a minimum such that the time required for re-calculation of the path is on the order of milliseconds

**Proposed solution:**

While the overall search pattern has not yet been finalized, with no prior idea of where a target is the entire space will need to be searched. Therefore, it is likely we will employ a simple back and forth swath search. In order to navigate around local obstacles however, a path planning algorithm becomes necessary.

In order to achieve necessary planning time, a 2D path will be calculated in lieu of a 3D one.

Two algorithms are being evaluated, A\* and PRM.

**Algorithm Details:**

**A\*/Voronoi**

A\* is a widely used algorithm which uses weighted cost functions to determine which node to choose.

Cost:

Where g(n) is the known cost of getting from initial to node n and h(n) is an estimate of cost to get from n to goal node (shortest path).

Inputs:

A\* uses as list of nodes defining locations that are able to be visited. In the 2D case, these nodes are listed in terms of their x and y coordinates.

Outputs:

A\* will generate a path. This path is comprised of a list of nodes which will guide the vehicle from start to goal nodes.

The following Figure 1, demonstrates the A\* approach given a grid where each traversable node is defined MxN coordinates. The red circles are obstacles.

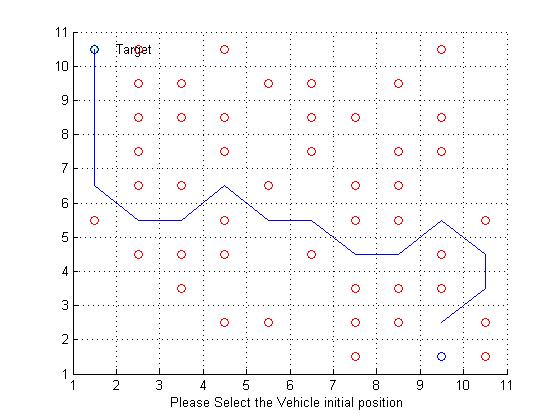


Figure 1. A\* Search Method Example provided by Mathworks File Share

Obstacle avoidance:

A raw A\* approach will yield a grid spacing that must be defined such that a diagonal path next to an obstacle must be greater than the radius of the vehicle in use. Therefore, each node must be the length of the diameter of the vehicle more some margin. This may or may not be a problem in our application.

What is likely a problem for us would be that the traversal of open space would be characterized by successive small steps across the grid which would yield a start stop motion that is undesirable. This could be smoothed out in higher level software.

An alternative to raw grid data that we are exploring employs the Voronoi algorithm to generate points which create paths equidistant between nodes. Figure 2 provides an example where the \*’s represent obstacles. The vertices of these polygons are outputs of the Voronoi algorithm which are then fed into the A\* search algorithm which generates the path found in Figure 3.



Figure 2. Voronoi generated paths

Next, A\* was implemented in Matlab. A\* plans the path of nodes which will be navigated to reach the goal. In the following, Figure 2, the origin is represented as a circle, the goal as X. The paths with dashed lines (that aren’t red) have been eliminated due to their proximity to obstacles.

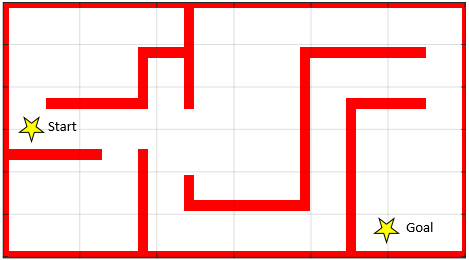


Figure 3. Voronoi/A\*

**Probabilistic Roadmap (PRM)** - Based on <http://www.cs.uu.nl/docs/vakken/mpap/papers/4.pdf>

A PRM planner consists of a learning and query phase. During the learning phase, a roadmap is created by sampling possible robot configurations at random with some probabilistic distribution in an obstacle-free space to compute a route from a starting point to a goal point. The sample points, or ‘nodes’, are connected via pathways of a configurable maximum length and any connections that intersect obstacle points are discarded. The query phase begins by attempting to find a path from the start and end points to nodes on the roadmap. Once the start and end points are connected to nodes on the roadmap, the planner seeks to connect those nodes based on the shortest path between them without colliding with obstacles.

The PRM planner requires a map grid where and vehicle position is quantized to grid cells (i.e. vehicle position coordinates are integer values that lie on the grid). The map should differentiate between free space and obstacles by setting array indices that correspond to free space to a value of 0 and obstacles to a value of 1. The algorithm also requires start and goal point coordinates that lie on the map grid. Figure 4 is an example of a map that can be used for the PRM planner.



Goal

Start

Figure 4. PRM initial map containing obstacle locations in red.

The map in Figure 4 is a 300 row x 600 column array where each cell represents an inch. The cells in red (obstacles) contain a value of 1 and whitespace representing open space contains values of 0. The start and goal points are denoted with yellow stars.

The PRM planner generates a random distribution of nodes that lie in free space, connecting each node to its neighbors. The maximum number and distance of the neighbors with relation to a target node are configurable. These nodes and their connections are the basis for the roadmap computed by the planner shown in Figure 5.

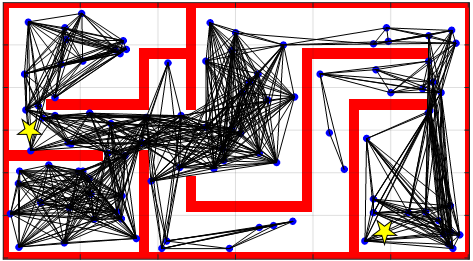


Figure 5. PRM-generated roadmap

Nodes are shown in blue and their connections are shown in black. Note that connections are only made when there are no intersections with mapped obstacles. If new, unmapped, obstacles are found, the planner should be re-executed with an updates map. Once the roadmap is generated, the planner attempts to connect the start and goal points to nodes on the roadmap by finding the shortest feasible path. Once the start and goal nodes are found on the roadmap, the algorithm cycles through the possible vehicle configurations until the shortest paths is found. Figure 6 shows the path calculated by the PRM planner in green.

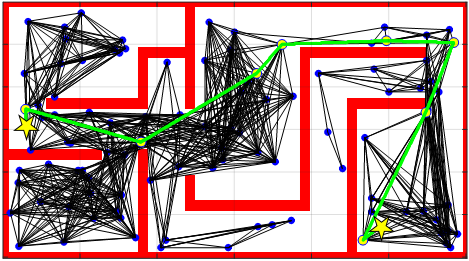


Figure 6. PRM-generated start-to-goal path

The PRM-generated path avoids collisions with obstacles shown on the map and reaches the goal point successfully. For this particular configuration, the PRM planner is able to compute a path in 1.350242 seconds (excluding time to create the map). The time to update an existing map (stored as a binary file) once the location of an obstacle is discovered is negligible. The PRM method is a reliable and robust method that avoids collisions. It is probabilistically complete so the probability of computing a path increases as the number of randomly placed nodes increases.

The disadvantages of the PRM planner are its speed and its reliance on random node placement. In order to increase the probability of finding a path from the start point to the goal point, it is necessary to increase the number of nodes on the roadmap which increase the computing cost. In certain cases, it is possible that the geometry of the space is sufficiently complex that no node connections can be made between the start and goal configurations in which case the space would need to be resampled.

**Coverage Path Planning Solution – Nearest Neighbor (NN) + 2Opt**

Coverage path planning is motivated by the need for the quadcopter to cover sufficient amount of space in a building to find a target. It is important that coverage path is short enough that the vehicle endurance time allows for the path to be completed. Therefore, processing speed and path efficiency are important to the design of the coverage solution. To fulfill the project needs, a Nearest Neighbor (NN) method is used with a 2Opt optimization.

The NN method works by auto-generating an evenly spaced set of goal points in the target space, discarding any of those points that collide with known obstacles. At this point, the calculated roadmap containing nodes and collision-free paths is loaded and each goal point is mapped to its nearest roadmap node. The A\* algorithm can then be used to generate an adjacency matrix which contains information regarding the cost of traversing the roadmap from one node to every other node. Once the costs are known, NN can be used to determine the most efficient way to visit each goal once.

NN works iteratively on each goal point, searching for the nearest goal to the current position based on the A\* cost. Once that goal is found, NN sets the next goal as the closest node to its new position, excluding the set of previously-visited nodes. Once a full path is computed, 2Opt optimization analyzes the path to find any cross-over and fixes them as shown in Figure 7.



Figure 7. 2Opt path optimization example.

The 2Opt is necessary because NN tends to diverge from some points when it only looks at the next closes node. 2Opt will process the path output by NN in order to generate a more efficient path.