3D Path Planning

MEAM 620 Project 1, Phase 2

Due: Tuesday, Feb 19, at 11:59 PM

In this assignment you will begin building a trajectory generator for a quadrotor by implement two graph search algorithms: dijkstra's algorithm and a popular variant called A*. The goal of this phase is to compute resolution optimal paths in a 3D environment from a start position to a goal position that avoid colliding with obstacles.

Map Files

The cuboid environments used for this assignment are defined in text files and example of one such file is shown here:

```
# An example environment
# boundary xmin ymin zmin xmax ymax zmax
# block xmin ymin zmin xmax ymax zmax r g b
boundary 0.0 0.0 0.0 20.0 5.0 6.0
block 3.10 0.0 2.10 3.90 5.0 6.0 255.0 0.0 0.0
block 9.10 0.0 2.10 9.90 5.0 6.0 255.0 0.0 0.0
```

where `#` is used for comments and the boundary of the environment as well as block obstacles are defined in terms of their minimum (lower-left-near) and maximum (upper-right-far) corners stored in a row vector as `[x_min y_min z_min x_max y_max z_max]`. The block definition also includes a triplet of RGB color values used in the visualization.

In order to find paths with a graph search algorithm it is necessary to first represent the environment as a graph. To do this we discritize space into a regular grid and attach graph nodes to each voxel. The choice of grid resolution is especially important and impacts both the runtime of the algorithm as well as its ability to find a path. Because this assignment is in Matlab, somewhat limiting the speed of your code, you will always want to coarsely discretize the z-dimension. Another consideration is safety, which becomes critical when considering real world applications. Since quadrotors are not infinitesimal points but rather physical objects that take up space, it is important to ensure that generated paths avoid all obstacles in the environment with some margin. The appropriate choice of margin is a balance between safety and path length. Choose a margin too small and your quad may strike an obstacle and crash. Choose a margin too big and the goal may become inaccessible.

Properly formatted maps may be loaded using the provided `load_map` function which takes the name of the textfile, grid resolution, and obstacle margin as inputs and returns a struct with map properties and a 3D matrix representing the voxel grid. Voxels with value 0 enclose free space while voxels with value 1 intersect obstacles. The function inflates each obstacle by a distance of margin in each direction before discritizing and marking the intercepted voxels as occupied. For indexing purposes, the origin of each voxel is considered to be the lower-left-front corner (the minimum corner of the voxel's bounding box). Another function called `plot_path` can be used for visualizing maps and paths. More detail can be found in the header descriptions of both files.

NOTE: a number of helper functions for indexing and visualizing 3D maps discritized into voxel grids are described in Section 4.

1 Dijkstra

Your main task is to implement Dijkstra's algorithm as described in class. A thorough description of the inputs and expected outputs can be found in the header of the `dijkstra.m` included in the project packet. Things to note:

- In addition to returning the path, `dijkstra()` should return the total number of states that were visited; see the description in the header of `dijkstra`
- One of the main considerations of any planner is how fast it is: part of your grade will be determined by how quickly your planner runs. The Matlab profiler will help you considerably. Also see this Mathworks page. For those of you who know how to program in C/C++ and use MEX, do **not** use them for this assignment. You are also **not** allowed to use any external libraries or implementations on this assignment.

2 A-Star

A popular variant of Dijkstra's algorithm is the A* algorithm. Since both algorithms are very similar, modify your completed `dijkstra()` function so that when the astar parameter is `true`, it uses distance to the goal as a heuristic to guide the search. Remember that any heuristic must be admissible and valid; if not, the algorithm is no longer guaranteed to return the shortest path.

3 Grading

For this assignment your grade will be determined by automated testing. Your planner must find optimal paths through a variety of environments and do so quickly. In addition, you must consider cases where there is no valid path or when there are no obstacles in the environment (i.e. `map.occgrid is empty`). Again, a thorough description of the expected output of `dijkstra` can be found in the description in the header of the function. Consider making your own maps and testing your algorithms on simple cases for which it is easy to reason the correct answer. Once again, we have provided a function called `plot_path` that can be used for displaying loaded maps as well as plotting resulting paths.

4 Helper Functions

The following helper function aid in indexing and visualizing 3D maps stored as voxel grids:

- `xyz = sub2pos(map, ijk)` : converts an $n \times 3$ matrix of subscripts (each row is the i,j,k subscripts of a map voxel) to an $n \times 3$ matrix of points (each row contains the x,y,z coordinates of a point), where `map.occgrid(i, j, k)` corresponds to the voxel containing point xyz [1]
- `ijk = pos2sub(map, xyz)` : does the inverse of `sub2pos`, taking metric positions xyz ($n \times 3$) and returning corresponding subscripts ($n \times 3$) ijk [1]
- `xyz = ind2pos(map, ind)` : converts an $n \times 1$ vector of linear indices of voxels in the map to an $n \times 3$ matrix of corresponding points xyz, where `map.occgrid(ind)` corresponds to the voxel containing point xyz [1]
- `ind = pos2ind(map, xyz)` : does the inverse of `ind2pos`, taking points xyz and returning the linear index of the corresponding voxels inside the map [1]
- `plot_path(map, path)` : takes a map and creates a 3D visualization and (optionally) plots a path through the environment

[1] these functions make the assumption that all points, indices, and subscripts are within bounds of the map; inputs must be checked for validity before calling

5 Submission

When you are finished, upload your code to Eniac and submit via `turnin`. The project name for this assignment is titled "`proj1phase2`" so the command to submit should be
"`turnin -c meam620 -p proj1phase2 -v *`", where `*` selects all files in the current directory for submission; be sure that the command is executed in a directory that only contains the files you want to submit. Your turnin submission should contain:

1. A `README` file detailing anything we should be aware of.
2. Files `dijkstra.m`, `load_map.m`, `plot_path.m`, as well as any other Matlab files you need to run your code.

Shortly after submitting you should receive an e-mail from `meam620@seas.upenn.edu` stating that your submission has been received. You can check on the status of your submission at https://alliance.seas.upenn.edu/~meam620/monitor/. Once the automated tests finish running, you should receive another e-mail containing your test results. This report will only tell you whether you passed or failed tests; it will not tell you what the test inputs were or why your code failed. Your code will be given at most 10 minutes to complete all the tests. There is no limit on the number of times you can submit your code.

Please do not attempt to determine what the automated tests are or otherwise try to game the automated test system. Any attempt to do so will be considered cheating, resulting in a 0 on this assignment and possible disciplinary action.