3D Path Planning

MEAM 620 Project 1, Phase 2

January 26, 2022

1 Introduction

In this assignment you will begin building a trajectory generator for a quadrotor by implementing two graph search algorithms: Dijkstra's algorithm and 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 avoids colliding with obstacles. This project is independent of Phase 1, but we will combine our planning and control algorithms in a subsequent phase.

2 Code Packet

2.1 Provided Files

- `flightsim/world.py`

  The environments used for this assignment are defined by **World** objects. As documented in the file header, the world is defined by a dictionary containing the bounds of the world ('extents'), and a list of dictionaries of axis-aligned cuboid obstacles ('blocks'), which have 'extents' and 'colors'. These worlds can be loaded from .json files like those provided in `util/`. We recommend you create more examples for yourself. Please do not submit or modify the World object code or any other files from `flightsim/`.

- `code/occupancy_map.py`

  In order to find paths with a graph search algorithm in a **World**, it is necessary to first represent the environment as a graph. To do this we discretize space into a regular grid and attach graph nodes to each voxel. An **OccupancyMap** object is a voxel grid corresponding to a **World** object. Voxels with value $False$ enclose free space while voxels with value $True$ intersect obstacles.

  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. 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. The provided **OccupancyMap** class is already parameterized by the resolution of the discretization and the margin of inflation.

  You are free to modify `occupancy_map.py`, though it is not necessary to complete the assignment.

- `code/sandbox.py`

  This script loads a `World`, runs your algorithm, and plots the results. It is useful for debugging.

- `util/test.py`

  This script runs tests against all `test_*.json` maps in `util`, including any you have created yourself.

2.2 Tasks

- `graph_search.py`

  Your main task is to implement Dijkstra's algorithm as described in class. A description of the inputs and expected outputs can be found in the header of the provided file.

  A popular variant of Dijkstra's algorithm is the A* algorithm. Since both algorithms are very similar, modify your completed graph search 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 consistent; if not, the algorithm is no longer guaranteed to return the shortest path.

  In addition to outputting the shortest path, you are also required to count and return the total number of expanded nodes. Nodes are counted as 'expanded' when they are removed from the priority queue and we update their neighbors' costs.

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. Consider making your own maps and testing your algorithms on simple cases for which it is easy to reason the correct answer.

4 Submission

4.1 Code Implementation

When you are finished, submit your code via Gradescope which can be accessed via the course Canvas page. Your submission should contain:

1. A `README` file detailing anything we should be aware of (collaborations, references, etc.).
2. Files `graph_search.py`, `occupancy_map.py`, as well as any other Python files you need to run your code.

Shortly after submitting you should receive an e-mail from `no-reply@gradescope.com` stating that your submission has been received. Once the automated tests finish running the results will be available on the Gradescope submission page. 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.

4.2 Summary "Slide"

Submit your single summary "slide" to the separate Gradescope assignment. Create your own test map, and solve this same map with both Dijkstra and A* using at least four different grid resolutions. Present a plot that illustrates for both algorithms how the number of nodes expanded changes as the map resolution changes. In addition, please show an illustration of your map.