CMSC 828T
Vision, Planning and Control in Aerial Robotics
Project 1 Phase 1 (P1Ph1): Graph-based Planning
Due on: 11:59:59PM on September 7th, 2017

Prof. Yiannis Aloimonos, Nitin J. Sanket,
Kanishka Ganguly, Snehesh Shrestha

August 30, 2017

1 Programming

1.1 Dijkstra's Algorithm

For this assignment, you will write a planner which can generate paths from a start position to an end position in a known 3D environment. Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks in Google maps. In this assignment, we will have a much simpler version of the map. You will need MATLAB software to write, test, and run your code. MATLAB can be obtained from www.terpware.umd.edu using your student login. We will be using MATLAB 2017a for testing your submissions.

1.1.1 The Map

Complete the implementation of `load_map.m` and `collide.m` that are provided to you.

The `load_map()` takes the filepath to an environment file and resolutions at which to discretize the environment and returns a representation of the environment for your planner. Here's an example the environment file format:

```
# An example environment
# boundary xmin ymin zmin xmax ymax zmax
# block xmin ymin zmin xmax ymax zmax r g b
boundary 0 0 0 45 35 6
block 1.0 1.0 1.0 3.0 3.0 3.0 0 255 0
block 20.0 10.0 0.0 21.0 20.0 6.0 0 0 255
```

- The format uses # for comments.
- The rectangular boundary extents for this environment are (x0, y0, z0) = (0, 0, 0) (lower left) and (x1, y1, z1) = (45, 35, 6) (upper right).
- A block line also specifies the rectangular boundary, and the blocks' color as a RGB triple with values from 0-255. All values are whitespace delimited and all numerical values are represented as a float.
- All distances (values provided) are in meters.
- Blocks can overlap.
- NOTE: Because this assignment is in MATLAB - somewhat limiting the speed of your code - we will always coarsely discretize the z-dimension.

There will be sample files provided to you for testing. This will be available in a folder called `sample_maps`.

1.1.2 Dijkstra

Implement Dijkstra's algorithm. You can refer to Wikipedia pseudo-code as well. Complete the files `dijkstra.m` and `plot_path.m` provided to you. Things to note:

- In addition to returning the path, `dijkstra()` should return the total number of states that were visited.
- `plot_path()` should create a figure which displays the environment as well as the planned path.
- 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 in that regard. For those of you know how to use C and MEX, please **do not use** those for this assignment.
- Your algorithm will have a **maximum run time of 30 seconds** for each test case. You are provided with some test maps in the folder called `sample_maps`.

```
map = load_map('map0.txt', 0.1, 2.0, 0.3);
tic;
path = dijkstra(map, start, stop);
toc;
```

All the submissions will be tested using the same system configuration. Don't worry about the timing before you complete the first draft of the algorithm. For this assignment, we have provided enough time for you to be able to meet the requirements. If you are not able to meet the required time, then you can look for ways to make your code faster.

1.2 A* Algorithm

A* is another powerful algorithm for finding a directed path between multiple points on a graph. This algorithm finds widespread use in path-planning and is efficient in both performance and accuracy.

Modify `dijkstra()` so that when the `astar` parameter is passed as `1`, it uses distance to the goal as a heuristic to guide the search. No other changes should be necessary. Remember that any heuristic must be admissible and valid; if not, the algorithm is no longer guaranteed to return the shortest path.

Your algorithm will need to have a **maximum run time of 30 seconds** for each test case. You are provided with some test maps in the folder called `sample_maps`.

```
map = load_map('map0.txt', 0.1, 2.0, 0.3);
tic;
path = dijkstra(map, start, stop, 1);
toc;
```

2 Submission Guidelines

You will be provided with a folder named `code` in which you will have files and directories organized as:

```
code
├── sample_maps directory
│   └── sample_map1.txt
├── load_map.m
├── collide.m
├── dijkstra.m
├── plot_path.m
└── ANY OTHER FILES OR DIRECTORIES
```

Any other files and folders you create is your choice, and will not factor into your grading. Please note that you must use relative paths if you refer to files inside other folders. Do not use `cd` statements, since that will guarantee your code will fail. **DO NOT** make any changes to the files already inside the `code` folder provided to you. You will only be required to zip the folder and submit that.
Your code only goes in between the specified comments in each function provided to you. NOWHERE ELSE will your code be accepted or graded.
Submit your assignment named according the format `code.zip` onto the website. **Please compress it as .zip compressed file format. Other formats will not be acceptable.** If your submission does not comply with the guidelines, you'll be given **ZERO** credit since we will be using an auto-grading system.

2.1 Auto-grading System

A few points to note regarding the auto-grading system:

- We will be following a strict set of guidelines regarding code upload and grading, since it is an automated system.
- Do not rename files unless specifically requested to do so. Do not move files around in the folder provided to you.
- We will be using your UMD login to automatically sort and name your submissions, so do not add any details to the submitted folder.
- Since this is a new system, there might be errors and issues regarding submissions and grading. We will accommodate such issues as they come up. Please use Piazza to bring up issues with your submission.
- DO NOT use a mobile browser to upload your code. Use a laptop or a desktop browser for submissions.
- The grading will be done periodically, and scores will be updated accordingly.
- You are allowed only 5 submissions per project and per phase, so be mindful of that. **TEST your code locally before submitting**.

DISCLAIMER: We will change submission policy as needed, since we are testing the submission server and there might be issues.

3 Collaboration Policy

You can discuss with any number of people. But the solution you turn in MUST be your own. Plagiarism is strictly prohibited. Plagiarism checker will be used to check your submission. Please make sure to **cite** any references from papers, websites, or any other student's work you might have referred.