Trajectory Generation and Control of a Quadrotor

MEAM 620 Project 1, Phase 3

Due: Monday, February 17th, 11:59 PM

1 Introduction

It is time to put everything together! In this phase, you will need to autonomously control a simulated quadrotor through a 3D environment with obstacles. It is essentially an integration of everything that you have done in phase 1 and 2, plus a few improvements to make your quadrotor fly better. Files can be downloaded from the course website.

2 Quadrotor, Map, etc.

The simulated quadrotor is assumed to be a cylinder with radius of 0.15 m and height of 0.1 m. Other properties of the quadrotor are identical to Phase 2. Map definition is identical to Phase 1.

3 Trajectory Generation

You may already notice that in Phase 2, although your quadrotor was asked to precisely track a given trajectory, it was never able to do so. For cases such as sharp turns in the `diamond` trajectory, it is likely that your quadrotor will have some overshoot when making the turn (depending on the how fast the quadrotor is flying). Unfortunately, the optimal path output from your implementation of Dijkstra or A* usually contains many sharp turns due to the voxel grid based discretization of the environment. To improve the performance, you may apply trajectory smoothing techniques to convert sharp turns into smooth trajectories that the robot can track. One example will be the 5^{th} order polynomial fitting covered in Chapter 3 [PC 11] [1] . You will have to determine the start and end points of the polynomial curve as well as all other boundary conditions.

Note that you will need to decide on a velocity profile to turn the path from Dijkstra or A* into a trajectory. The speed of the robot does not need to be a constant.

Complete the implementation of `trajectory_generator.m`. `trajectory_generator(...)` takes a path from Dijkstra or A* and convert it into a trajectory as a function of time. You are allowed to use the map for trajectory generation. You are essentially modifying the trajectory files in phase 2 (`circle.m` or `diamond.m`) such that they are able to accept externally specified paths. You should pre-compute as much quantities as possible and avoid fitting polynomial curves in every call of the function.

4 Collisions

Your quadrotor should fly as fast as possible. However, a real quadrotor is not allowed to collide with anything (video). Therefore, we have zero tolerance towards collision – if you collide, you crash, you get zero for that test.

[1] P. Corke, Robotics, Vision and Control: Fundamental Algorithms in Matlab, Springer Tracts in Advanced Robotics, 2011

After phase 2, you should already have an idea of how well your controller works. Additionally, trajectory smoothing may also deviate the actual trajectory from the planned path. Therefore, you should make good use of the `margin` parameter and set your speed carefully. Please be aware that the robot is assumed to be a cylinder. You should make sure that no part of the robot collides with any obstacles.

However, we guarantee that for all the testing maps, with the specified start and goal locations, there will always be openings that allow cylinder with a radius of 0.5 m and height of 0.5 m to pass through.

5 Coding Requirements

First, you should generate your optimal path following the same procedure as Phase 1. Then, the path will be converted to a trajectory and tracked by the quadrotor. An example sequence is:

```
map = load_map('maps/map1.txt', 0.1, 1.0, 0.2);
start = [0.0  -4.9 0.2];
stop  = [8.0  18.0 3.0];
path  = dijkstra(map, start, stop, true);
init_script;
trajectory = test_trajectory(start, stop, map, path, vis);
```

`trajectory = test_trajectory(...)` will return the actual trajectory executed by the robot. See comments in the code for details. You are free to add visualization code to this file to see intermediate results, however, we will not use your version when testing your code. Make sure that you can run your code with the original version of `runsim.m`. Performance evaluation will be based on the output `trajectory`, which contains 100 Hz samples of the actual trajectory of the quadrotor. You are free to reuse any or all of your Phase 1 and 2 code. Your Phase 3 implementation should be self-contained. Here are important coding requirements for Phase 3:

1. You are not allowed to change input and output formats of any functions.
2. Put your phase 1 code (`dijkstra.m`, etc.) into `phase1` folder
3. Put your phase 2 code (`controller.m`, etc.) into `phase2` folder
4. You may need to slightly change your Phase 1 and/or Phase 2 implementation such that it works within the current code base.
5. Complete the implementation of `trajectory_generator.m`.
6. You may need to change `init_script.m` to initialize your trajectory generator.
7. Please read through `test_trajectory.m` carefully and make sure that you can run your code with the original file.
8. An example testing script is given in `runsim.m`, we will use the same sequence to test your code against approximately 5 different maps.

6 Grading

Your grade will be determined by:

1. How long does the quadrotor take to reach the goal (excluding planning time).
2. Comparison between the length of the actual trajectory of quadrotor and the length of the shortest path (path from Dijkstra or A*).

Remember, if you collide, you get zero for that map, so do not go too crazy.

6.1 Submission

When you are finished you may submit your code via turnin. The project name for this assignment is titled "`proj1phase3`" so the command to submit should be
`turnin -c meam620 -p proj1phase3 -v *`
Your turnin submission should contain:

1. A README detailing anything we should be aware of.
2. All necessary files such that we can just run the automated testing script `runsim` to see your results.

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.

7 Extension to Multiple Robots

The goal of the project is to consider the problem of finding collision-free paths for multiple aerial robots, develop one or more algorithms to solve the problem, and demonstrate the algorithm on a simulator. Note that the simulation supports simulating multiple robots. Of course the performance depends on your laptop and its CPU. You will receive extra credit depending on the quality of your work. If your work is original, we can realize the algorithms on the experimental testbed and produce a conference paper.

The project is open ended and you might consider pursuing one or more options. Here we provide a few suggestions that might help guide your thinking.

1. Small team: planning in the joint state space

The methods you have already developed can work in the cartesian product of the state spaces of the robots. For m robots (points or spheres) in $\mathbb{R}^n$, the joint state space is $\mathbb{R}^n \times \mathbb{R}^n \times \ldots \times \mathbb{R}^n$. It may be possible to solve this problem for m robots in n dimensions by discretizing the space and using a graph search algorithm for small m and n as shown in Fig. 1(a). What is the largest problem you can solve? How does this problem scale with n and m? Are there ways to simplify the problem so you can solve the robot planning problems independently? Use similar environments to the ones you were given in earlier phases of this project and show results using a simulator.

2. Large team: planning trajectories for identical robots in free space

When we consider larger teams as shown in Fig. 1(b), the curse of dimensionality makes it impractical to solve the motion planning robots. The joint state space is prohibitively large (verify this), and the memory requirements for cell decomposition techniques are excessive (how excessive?). On the other hand, if you assume all robots are identical, it is possible to solve the planning problem by also considering the goal assignment (who goes to what goal) as shown in [1]. You might consider developing an algorithm that solves the Concurrent Assignment and Planning of Trajectories (CAPT) problem for a large team of robots using a centralized approach. Is such an approach complete? Will the algorithm in [1] always give you solutions? How do you solve the assignment problem? How does this problem scale with n and m?

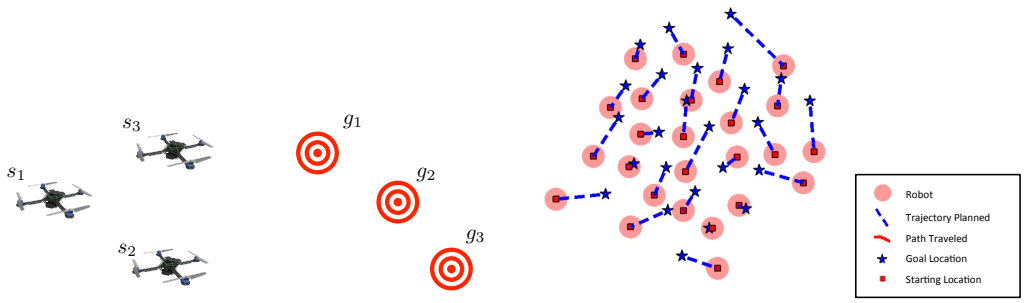

Figure 1: The multi robot motion planning problem: (a) small group (left); (b) medium sized group (right).

3. Planning suboptimal trajectories for teams of robots in cluttered environments

If the robots are not identical and the group size is large, the above approaches will not work. You might instead try to pursue approaches to solve the decoupled subproblems which can be reduced to finding single robot motion plans. Of course the results will be suboptimal. How suboptimal are the results? Can you use simulations with many representative environments and explore this question? Are there tricks you can use to improve the results?

Deliverable

You can pursue any avenue you find interesting. You can even try multiple avenues of investigation depending on how you choose you spend your spring break. You must turn in a paper that includes figures, plots, references and tables as needed (not to exceed 10 pages) with a youtube video (less than 2 minutes) that addresses the following questions:

1. What: The problem you tried to solve?
2. Why: Why is the problem important? Why should we all care?
3. How: How did you go about solving the problem?
4. Key Results

Note that this part of the project is due on March 14, 2014. However, if you want the help of the teaching staff, you have to get an earlier start on the project since our availability is not guaranteed from March 8-14.

References

[1] Matthew Turpin, Nathan Michael, and Vijay Kumar. Capt: Concurrent assignment and planning of trajectories for multiple robots. *The International Journal of Robotics Research*, 33(1):98–112, 2014.