An Autonomous VIO-based Quadcopter

MEAM 620 Project 3

April 11, 2024

1 Introduction

The goal of this project is to integrate everything we have learned from this course so far! Let's do a quick review:

- Project 1 tasked us to plan trajectories and track them accurately, while the ground-truth states of the robot (positions and velocities) were given to us.
- Project 2 tasked us to estimate robot's state given noisy sensor measurements, but there was no robot autonomy (planning and control) involved.

Our goal for Project 3 is to integrate the state estimation from Project 2 with the planning and control from Project 1. In other words, you will use your Project 2 code to estimate the state of the robot, and use your Project 1 code to plan and track trajectories in the simulation environment.

Using visual inertial odometry (VIO) in-the-loop has proven to be a reliable approach to implement GPS-denied autonomy stack in academia [1, 2] and industry [3]. Once a quadcopter can accurately estimate its own state and use it for planning and control, the last step towards autonomy is to implement a mapping solution so the quadcopter can sense where the obstacles are using its on-board sensors.

The principle objective for Project 3 is to achieve quadcopter planning and control with on-board state estimation. However, extra credit exercises are included for students who are interested in completing the entire autonomy pipeline.

2 Simulator Implementation

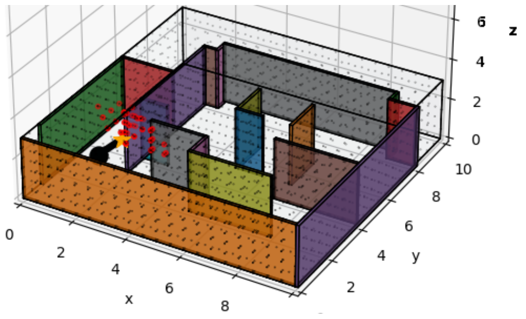
A full simulator implementation of a stereo VIO-based quadcopter has to synthesize camera images from the robot position. This is the preferred approach when we want to test novel VIO algorithms, or when we want to test VIO-based autonomy. The drawback of this approach is that the simulation requires capable hardware to generate photorealistic camera images. Fig. 1a shows an example of such simulation environments.

Since these specialized hardware may not always be readily available, in this project we have generated the features beforehand in `flightsim` by attaching *markers* to the obstacles and boundaries of the simulator (Fig. 1b). The features are then projected into the camera frame of the quadcopter and this is the data that will be provided to you. This approach requires significantly less compute than providing complete frames and removes the need to use feature detection code.

3 Main Task : Trajectory Tracking with State Estimator

The task is the same as Project 1.3's, except that we do not give the controller the ground truth state. Instead, the state comes from your state estimator. It is likely you will need to modify your planner, controller, and/or even the state estimator to make the quadcopter able to track the planned trajectory.

(a)



(b)

Figure 1: **Left panel:** A quadcopter in a Unity3D-based photorealistic simulation environment. **Right panel:** Example environment with sprinkled features in *flightsim* (the simulator used in the class). Features (small black dots) are sprinkled on walls, ceiling, ground of the environment. The robot center is shown as the big black dot, and the camera facing direction is shown as the ray from black dot to orange star (facing up, slightly tilted). The red dots show the features that are currently observable to the camera. Features are considered observable if (1) their distances to the camera are within a predefined threshold, (2) they lie inside the camera field of view, (3) there are no obstacles or walls exist between them and the camera.

We provide a visualization of the markers that the quadcopter is observing at a specific position. This should give you a hint on how to improve the performance of your vehicle planning and control strategies.
Hint: For you to better tune your code, you can play around with the sensor noise parameters in the file `flightsim/sensors/vio_utils.py`. Specifically, for IMU, those parameters define the noise: accelerometer_noise_density, accelerometer_random_walk, gyroscope_noise_density, gyroscope_random_walk; For stereo features, those parameters define the noise: image_u_std_dev, image_v_std_dev, image_measurement_covariance. You are recommended to **multiply those parameters with the same number** (e.g., 0.01 for lower noise level) instead of changing them individually. Note that the autograder will still use the predefined noise parameters.
Hint: An aggressive desired trajectory or PID tuning can increase the state estimation error. Start with lower gains and low velocities and be sure to reach the goal before trying aggressive trajectories.
Hint: We strongly suggest you to create new maps to test your quadcopter performance.

4 Extra credit (EC)

When navigating a quadrotor in unknown environments, you may need to replan your trajectory multiple times in shorter segments as you discover open corridors for flying [4]. After finishing the main task, extra-credit sections will require you to refine your global planner to a local planner with a limited sensor range. You will need to modify your existing code as well as some additional code we provide. You should not need to modify anything in `flightsim` and any backend simulator code will be provided.

As shown in Fig. 2, we provide an example replanning framework in `code/world_traj.py`. You can use or implement your own pipeline. There're some changes for local planning you should take into considerations:

- As the local start has no-zero velocity, it may generate zig-zagging curves on replanning points. You may need to adjust waypoints and time allocation to mitigate it. A feasible way is to directly set the current start as the current state or index position along the trajectory while assuming the local goal is the zero state.

  relative

- When replanning, you have to consider using the absolute or relative time for trajectory representation.

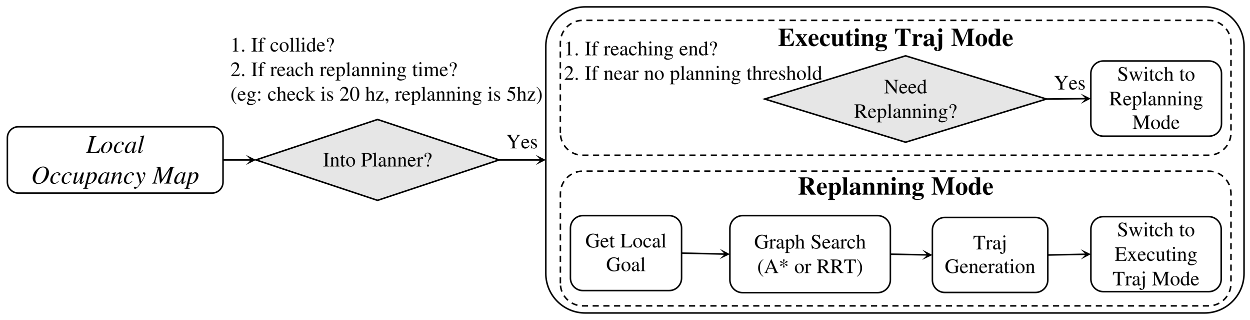


Figure 2: Overview of the replanning framework.

- The method provided to find a local goal is draws a straight line from the current position toward the final goal state, for some fixed distance. This local goal may lie within an obstacle. One way to work around this is to relax the graph search exit condition with some tolerance for a collision-free goal position.

5 Code Organization

The code packet that you are being provided with as part of this assignment is organized in the usual manner. In the top level directory there is a file entitled `setup.py` which you should run in order to install all of the needed packages.

5.1 Main Task

The `proj3` package is divided into 2 subdirectories.

- The `util` directory contains a few maps that you can use for testing your planner, controller and estimator.
- The `code` directory contains a set of code files and sandbox files which constitute the coding assignment. You will need to copy your code from the previous two projects to this `code` directory `graph_search.py`, `occupancy_map.py`, `se3_control.py`, `world_traj.py` and `vio.py`.

5.2 Extra Credit

The `proj3_ec` package has the same subdirectories, but `occupancy_map.py` is in `util` directory that you cannot make changes. With limited sensor range, you can only use `occupancy_map.py` to generate a local map centered at the current position.

6 Coding Requirements

6.1 Main Task

You will be provided with a project packet containing the code you need to complete the assignment. For this phase you will need your graph search algorithm, trajectory planner and controller from Project 1, and state estimator from Project 2. In summary:

1. Copy over your Project 1 `code` directory `graph_search.py`, `occupancy_map.py`, `se3_control.py`, `world_traj.py`.

2. Copy over your Project 2 `vio.py`
3. Now that all your code resides in `proj3/code`. Make sure your `import` commands are adjusted accordingly in `proj3/code/sandbox.py` and `flightsim/sensors/vio_utils.py` (for example, `from proj1_3.code.occupancy_map` becomes `from proj3.code.occupancy_map`).
4. If necessary, improve your the implementation of `se3_control.py` and `world_traj.py`.
5. Use the provided `code/sandbox.py` to aid in tuning and analysis.
6. Test your implementation on a collection of given maps using `util/test.py`

6.2 Extra Credit

In addition to following the same procedures as the main task, you need to fill your Project 1 code in the new `world_traj.py` and `graph_search.py` in `proj3_ec/code`

7 Grading

7.1 Main Task

For the main task, a significant part of the grade will be determined by automated testing. You must find trajectories through six obstacle filled environments which your quadcopter must then quickly and accurately follow without collision. Performance will be measured in terms of the in-simulation flight time from start to goal. For each map, you will earn 11 points for a safe flight and up to an additional 8 points for a fast completion time for a total of 114 points. The time targets for each map will be in your Gradescope report; attaining full marks on every trajectory may be extremely challenging.

You are also required to submit a summary report, as described in Sec. 8.3. The report is worth 36 points.

7.2 Extra Credit

We will be providing an extra credit component that will require you to implement a local planning/re-planning component into your trajectory planner. The code and additional handouts will be released at a later date.

The extra credit component will be worth 30 points, which will contribute to the total combined grade of Project 3 Code and Report.

8 Deliverables

You should submit a report document and your code.

8.1 Code Submission

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.txt` file detailing anything we should be aware of (collaborations, references, etc.).
2. Files `se3_control.py`, `graph_search.py`, `occupancy_map.py`, `world_traj.py`, `vio.py` and 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.

8.2 EC Code Submission

If you attempt an EC section, you may submit this code in the corresponding entry in Gradescope. The autograder will use the same test cases as the standard autograder, with 3x time requirements. Ie. if a map has a time maximum of 15s, the extra credit version will have a time maximum of 45s.

8.3 Report

We ask you to write a 2-page report stating your findings in this project. Submit your report to the separate Gradescope assignment discussing, among other things:

1. What you have changed to your code since project 1 and 2.
2. Why did you need to make those changes.
3. Anything else you would like to discuss.

If you attempt an EC section, you can add +2 pages per EC attempt. We are interested in knowing:

1. How you implemented the EC.
2. How is the performance of your system compared to the vanilla implementation/simulator.
3. Are there failure cases in your implementation?
4. All the bibliography you used to implement the EC.

9 Academic Integrity

Please do not attempt to determine what the automated tests are or otherwise game the automated test system. This is an individual submission and must reflect your own work. You are encouraged to discuss the project with your peers in detail, but you may not share code. **Using your partners' code from Lab. 1.4 is not allowed, and it will be considered plagiarism**. Please acknowledge any assistance in detail in your `readme.txt`.

References

[1] K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Experiments in fast, autonomous, gps-denied quadrotor flight," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7832–7839, 2018.

[2] D. Thakur, Y. Tao, R. Li, A. Zhou, A. Kushleyev, and V. Kumar, "Swarm of inexpensive heterogeneous micro aerial vehicles," in *International Symposium on Experimental Robotics*, pp. 413–423, Springer, 2020.

[3] Skydio, "Skydio 2+." https://www.skydio.com/skydio-2-plus. Autonomous Drones for business, public safety and creative endeavors.

[4] S. Liu, N. Atanasov, K. Mohta, and V. Kumar, "Search-based motion planning for quadrotors using linear quadratic minimum time control," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 2872–2879, IEEE, 2017.