Project 3: An Autonomous VIO-based Quadcopter

1 System Overview and Key Logistic

In the previous projects, we covered the topic from estimation, planning and control.

In Project 1, we were provided with the true state of the quadrotor, including its position, velocity, orientation, and angular velocity, without the involvement of sensing systems or state estimation processes. The main focus was on developing a motion planner to generate feasible trajectories from a known map, and designing a controller capable of accurately tracking these trajectories.

In Project 2, we focused on algorithms for state estimation. For example, we used a complementary filter to estimate the attitude of the quadrotor based on IMU data. Additionally, we estimated the relative pose(attitude and position) from stereo images using the RANSAC method. By knowing both the pros and cons of IMU and stereo vision, we built a VIO system using an Error State Kalman Filter to fuse the IMU and stereo image data, which gives more accurate state estimation under noisy measurement conditions.

By combining the above two projects, we can build a quadrotor system able to self-plan control with onboard state estimation, and allow it to localize itself in the environment. This is similar to how a human uses a brain and sensory organs to perceive and navigate the world. However, we still need to assume that a complete map of the environment is known beforehand, so that a global planner can be constructed; otherwise, it would be impossible to plan a full trajectory.

To be more precise, the pipeline is as follows: we first plan a reference trajectory offline. Then, during the simulation, the VIO algorithm runs by collecting IMU data and stereo camera features (using markers) to estimate the best state in real time. An SE(3) controller is then used to generate motor commands based on the current estimated state and the desired state. Following is the detail of the implemented approach.

1.1 State estimation

We run a quaternion-based error state Kalman filter that fuses high-rate IMU with periodic stereo observations. It fuses IMU measurements (accelerometer and gyroscope data) for high-frequency state propagation and stereo-camera observations for low-frequency updates. Specifically, IMU data provide continuous estimates of the quadrotor's motion, while stereo image pairs give spatial constraints on the position and orientation.

1.2 Trajectory Planing

Our pipeline for trajectory planning is straightforward. Initially, we perform a graph search from the start to the goal position using the A* algorithm on a voxel-based occupancy map, with user-defined resolution and obstacle margins. After obtaining the raw path, we employ the DP algorithm to convert it into sparse waypoints, to best preserve the original trajectory shape. Lastly, we formulate and solve a minimum-jerk trajectory optimization as a quadratic programming problem and yield a smooth, dynamically feasible trajectory.

1.3 Controller

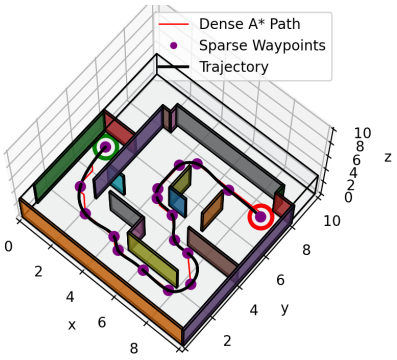
The controller employs an SE(3)-based geometric control approach, which manages both position tracking in three-dimensional space $\mathbb{R}^3$ and orientation alignment on $SO(3)$. The controller calculates the desired thrust direction to match the quadrotor's actual thrust vector with the commanded acceleration from trajectory planning. Simultaneously, it computes the required torque inputs to manage orientation according to the formula formed by the attitude error.

2 Code Changes Since Projects 1 and 2, and Explanation and Results of the Changes

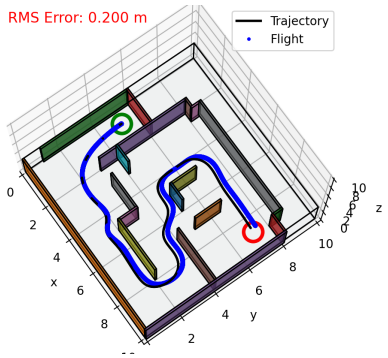
- **PID controller and parameters:** First, we introduce the integral term to handle potential steady-state errors. Since the quadrotor's state is estimated rather than directly measured, there is inevitably some variation between the estimated state and the true state. This estimation error introduces uncertainties into the feedback loop. Therefore, our controller must be capable of compensating for these inaccuracies to maintain stability and performance. To adapt to the dynamics under noisy state feedback, we carefully tune the PID parameters to prevent the quadrotor from **colliding with obstacles** and to best reduce the **tracking error**.

  The result of these changes indeed works and can reduce the possibility of collision while also reducing the tracking error. Figure X shows the trajectory after parameter tuning.

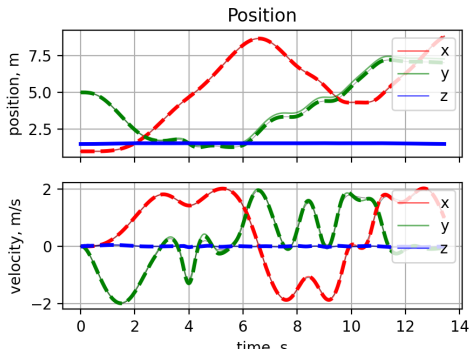
- **The Maximum Velocity of the Quadcopte in Planner:** As the velocity of the quadcopter significantly affects both the tracking error and the quality of state estimation from the VIO system, it must be carefully rtunned. Higher speeds can cause faster growth of integration errors and more accumulated drift. Also, increased velocity reduces the frequency of visual feature observations, making it more difficult for the system to timely correct its estimation errors. The value we finally obtain for $v_{\max}$ is around average 2.8 m/s, And our using this tuned parameter considers both aggressive flight and reduced tracking error.



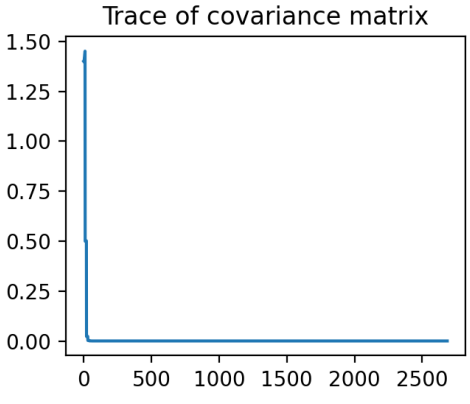
(a) Planned trajectory based on the A* method with selected sparse waypoints



(b) Estimated trajectory(real) vs. reference trajectory, showing small deviations



(c) Estimated (dashed lines) vs. reference (solid lines) position and velocity, reflecting tracking errors. The horizontal axis unit is times(s)



(d) Trace of the covariance matrix, showing increasing accuracy and stability of state estimation. The horizontal axis unit is timestep.

Figure 1: Relevant plots demonstrating the performance of our autonomous vehicle system

Part 2: Local Planner Algorithms

3 Extra credit

3.1 How to implemented this local planner?

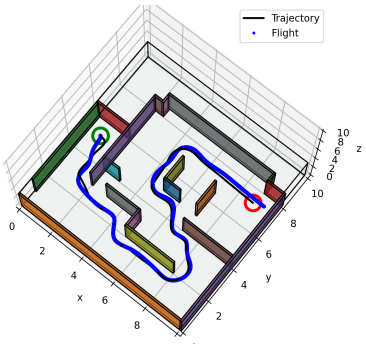
First, I would like to discuss the difference in the algorithms of the global planner and local planner

- **Input Map**: The global planner accesses the full occupancy map once at the beginning, where complete information about the environment is available. In contrast, the local planner only accesses a limited-size local map, which is user-defined and dynamically updated during flight based on the quadrotor's current position.
- **Start/Goal**: The global planner uses a fixed global start and goal pair throughout the entire planning process. In contrast, the local planner repeatedly defines new local start and goal positions in each replanning part. The local goal is typically a cropped point along the straight line toward the final global goal, within a predefined planning horizon. Some tolerance is allowed to ensure the goal lies in free space.
- **Planning Frequency**: The global planner performs the graph search only once offline. The local planner, however, replans dynamically during flight whenever certain conditions are met, such as predicted collisions, exceeding a planning horizon, or elapsed replanning intervals.
- **Flexibility**: The global planner is rigid once the quadrotor deviates from the planned trajectory due to disturbance or estimation error, it cannot adapt. In contrast, the local planner supports adaptive replanning and can handle unexpected changes in the environment or state estimation drift.
- **Robustness**: While the global planner provides a stable, precomputed solution, it is not robust to online disturbances, it only adapt the controller self instead of planning a new trajectory. If the quadrotor deviates beyond a controllable range, it may completely lose track of the path. The local planner, by contrast, is more robust and flexible, whcih capable of replanting on the fly to maintain safety because of the collison checking mechnism.

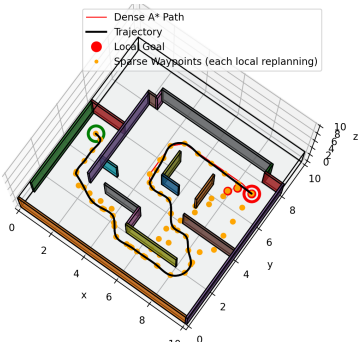
Our implementation basically follows the instructions provided in the PDF. Notably, to prevent the local goal from lying inside an obstacle, which could lead to failure in finding a feasible path from the start. We use a goal tolerance mechanism in our graph search. This allows the planner to terminate early and accept a nearby collision-free point within a specified radius as the effective goal.

We implemented a local replanning pipeline where the quadrotor incrementally navigates toward the global goal. At each planning cycle, we construct a local occupancy map centered at the current estimated position and select a cropped local goal along the global direction. We then perform A* search within the local map, generate sparse waypoints, and fit a smooth cubic spline for trajectory tracking (changes made here compare to the previous part). During flight, if the estimated trajectory is predicted to collide with obstacles due to estimation drift or environmental changes, we trigger replanning from the current state. This local planning framework enables online adaptability and continue these steps until it reach the goal.

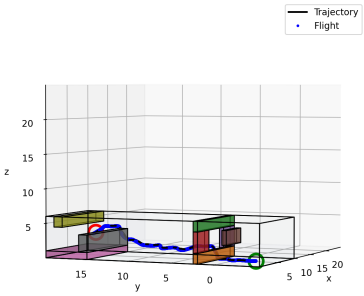
3.2 How is the performance of your system compared to the vanilla implementation/simulator?



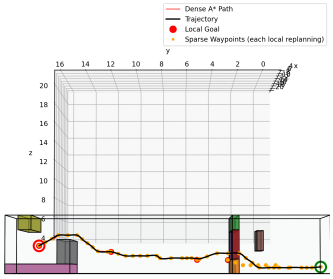
(a) Estimated trajectory(real) vs. reference trajectory, showing small deviations



(b) Planned trajectory



(c) Estimated trajectory(real) vs. reference trajectory in map 2



(d) Planned trajectory

Figure 2: Performance of our local planner

The figure above illustrates the flight trajectories generated by the local planner in two different environments, with the flight speed kept consistent with the drone shown in Fig. 1. The yellow dots represent all sparse waypoints generated during the replanning process, clearly demonstrating the system's ability to dynamically check the obstcales during flight. When the drone predicts a potential collision ahead, the replanning mechanism is triggered to perform a new local path search from the current position and generate an updated trajectory. This behavior is particularly evident in Fig 2.(d), where the initially planned path was predicted to result in a collision within the next 3 seconds. As a result, the replanning process was activated, and a new local goal and trajectory were computed, leading the drone to adjust its direction accordingly. Overall I think the performance of new system compared to the vanilla implementation is much better in the safety level, but obviously need more time and computation resources.

3.3 Are there failure cases in your implementation?

Yes, our local planner can fail in several scenarios (many are potential, but in our tests, failure occurred only at high speeds). First, if the cropped local goal lies outside the local map or within an obstacle , even though we have a tolerance mechanism to ensure the goal is not located in such regions replanning cannot proceed. Second, if no feasible path exists between the current position and the local goal, A* will fail, and consequently, the local planner will also fail.Third, when the drone is near the global goal, minor estimation drift may cause local goal selection to fail; however, this issue is relatively uncommon.Finally, at high speeds, the drone may collide before replanning can be triggered in time.

4 Reference and readme.text

There are no collaborators in this homework and the materials used are shown below:

1. Lecture slides

2. 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.