MEAM6200 Project 3 Report: An Autonomous VIO-based Quadcopter

Dhyey Shah

I. Introduction and System Overview

The objective of Project 3 was to integrate the trajectory planning and control strategies from Project 1 with the state estimation techniques from Project 2, enabling autonomous navigation of a quadcopter using visual inertial odometry (VIO) in a GPS-denied environment. Project 1 focused on planning and tracking trajectories using ground-truth robot states, while Project 2 involved estimating the robot's state from noisy sensor measurements. In Project 3, the quadcopter relies on VIO-based state estimates to plan and follow collision-free trajectories through obstacle-filled environments, as simulated in the flightsim environment with pre-generated features.

The system leveraged a modular pipeline including a graph-based path planner that we had coded in graph_search.py which generates a coarse path, which is sparsified and smoothed by our code in world_traj.py to produce a dynamically feasible trajectory. The VIO module in vio.py from our Proj_2 estimates the quadcopter's state, and the controller se3_control.py ensures accurate trajectory tracking throughout.

II. Changes in the Code Since Proj_1 & 2

No major modifications were made to my core code files (`graph_search.py`, `occupancy_map.py`, `se3_control.py`, `world_traj.py`, and `vio.py`) from Projects 1 and 2 for Project 3. The implementation from Project 1.3 was sufficiently robust to handle the integration of VIO-based state estimation without requiring changes. The main reason for the smooth working of my pipeline was a reliable trajectory planner in `world_traj.py` which was designed with adaptive mechanisms, such as dynamic margin adjustment (starting from an initial margin of 0.60m and reducing by a step of 0.05 everytime it fails to find a path) and velocity-based time allocation, which ensured compatibility with noisy state estimates. Similarly, the controller in `se3_control.py` used well-tuned gains to maintain stability under estimation errors, and the VIO module from Project 2 was optimized to provide reliable state estimates.

The only adjustments made were to the import statements in `sandbox.py` and `flightsim/sensors/vio_nonprofit.py` to reflect the new project directory structure. These changes were purely logistical and did not affect the algorithmic logic.

III. Why Were These Changes Made

Since the core algorithms were unchanged, the focus was on ensuring seamless integration of the existing codebase into the Project 3 framework. The robustness of the Project 1.3 implementation was critical, as it was designed to:

- The adaptive margin structure was necessary to balance safety and feasibility. A fixed, large margin might prevent finding paths in narrow corridors, while a fixed, small margin risks collisions due to state estimation errors. This mechanism allowed the planner to iteratively relax constraints when no path was found, ensuring a feasible trajectory even in challenging maps. It compensated for potential inaccuracies in the occupancy map caused by noisy VIO estimates.
- The velocity based time allocation for different segments of the path, was needed to tailor the quadcopter's speed to the environment's complexity and the reliability of state estimates. By slowing down in shorter, potentially obstacle-dense segments, the planner reduced the risk of overshooting or deviating from the planned trajectory due to estimation errors. This ensured smooth, dynamically feasible trajectories that the controller could track reliably.
- The waypoint sparsification was done strategically to prune the unnecessary waypoints using RDP and collinearity check which helped with to obtain a smoother trajectory which suited the tuned gains. These gains enabled precise trajectory tracking while damping the effects of estimation noise, preventing the quadcopter from becoming unstable in dynamic or cluttered environments.

IV. Result of the Changes

The unchanged codebase successfully met the Project 3 requirements, enabling the quadcopter to navigate six obstacle-filled maps with safe and fast trajectories. The adaptive margin in `world_traj.py` ensured collision-free paths by iteratively reducing the safety margin from $0.60\,\mathrm{m}$ to a minimum of $0.10\,\mathrm{m}$ in steps of $0.05\,\mathrm{m}$, allowing the planner to find feasible paths even in tight spaces. The spline used (similar to proj_1.3):

$$s_i(t) = a_i + b_i t + c_i t^2 + d_i t^3, \quad t \in [t_k, t_{k+1}],$$

where a_i, b_i, c_i, and d_i are coefficients computed to minimize acceleration, and t_k are the segment times. The trajectory smoothing techniques (RDP sparsification, collinear point removal, and cubic spline interpolation) produced dynamically feasible trajectories that the controller tracked accurately, even with VIO-induced noise.

Testing on the provided maps via `util/test.py` showed that the quadcopter achieved safe flights and competitive completion times. The controller's gains, tuned in Project 1.3, maintained stability, and the VIO module provided state estimates with sufficient accuracy to avoid collisions and reach the goal.

Below I attach the plots for 2 of the 3 sandbox maps (maze and over_under) which show the performance of the pipeline. I attach the plots to show the planned path, trajectory followed and the trace of the covariance matrix to show a reliable performance of my framework.

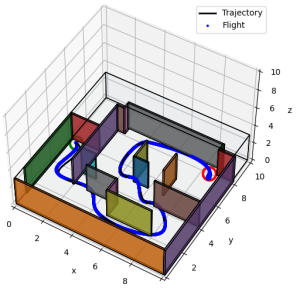

Fig. 1: Trajectory for Maze

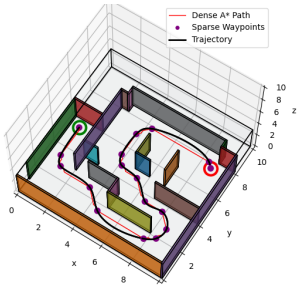

Fig. 2: Planned Path for Maze

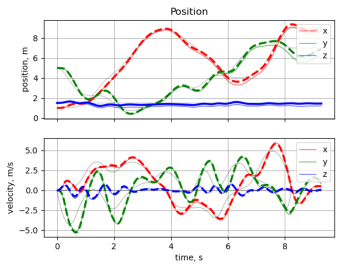

Fig. 3: Position & Velocity with Time

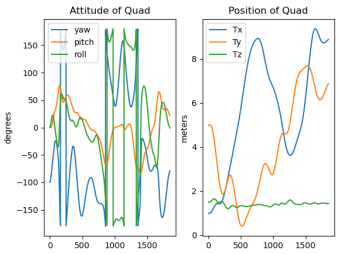

Fig. 4: Attitude & Position of Quadrotor

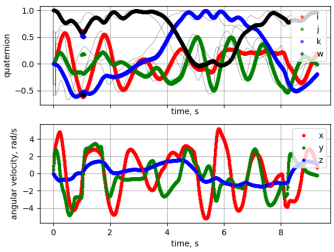

Fig. 5: Orientation vs Time

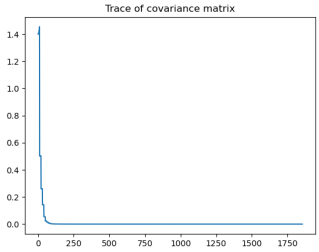

Fig. 6: Trace of Covariance Matrix

V. Results and Additional Discussion

The pipeline I implemented effectively mitigated VIO noise, while the SE(3) controller ensured precise tracking. However, improvements could enhance the performance for further studies by incorporating loop closure or feature tracking, as discussed in lecture on SLAM, could reduce state estimation drift, especially in feature-sparse maps. Also adopting model predictive control (MPC) for trajectory generation, could optimize paths in real-time, improving responsiveness in dynamic environments. Lastly, fusing VIO with additional sensors like LiDAR could improve map accuracy, reducing reliance on the $5\,\mathrm{m}$ local map in the EC task.

Future work could also focus on includes testing with real quadcopters to validate simulation results and implementing MPC for the EC planner to handle dynamic obstacles more efficiently.

VI. Extra Credit: Local Planning Implementation

A. Implementation

For the EC task I implement a local planner in `world_traj.py`, enabling the quadcopter to replan trajectories within a $5\,\mathrm{m}$ local map. Key components included:

- Local Occupancy Map: A $0.1\,\mathrm{m}$ resolution map, updated at the quadcopter's position, captures obstacles within $5\,\mathrm{m}$. The margin adjusts from $0.38\,\mathrm{m}$ to $0.2\,\mathrm{m}$ in $0.05\,\mathrm{m}$ steps.
- A* Planning: The `graph_search.py` generates a dense path, verified for collision-free points. The planner retries with reduced margins if no path is found.
- Replanning Logic: The `replan` method checks for collisions every $0.1\,\mathrm{s}$, triggering replanning if a collision is detected or the trajectory is nearly complete. The planning horizon adjusts from $7.5\,\mathrm{m}$ to $2.0\,\mathrm{m}$ in $1.0\,\mathrm{m}$ steps.
- Trajectory Generation: The dense path is sparsified using RDP ($\epsilon = 0.1$) and collinearity checks (atol $= 10^{-3}$). Waypoints are snapped to free voxel centers using `find_nearest_free_center` if occupied. A cubic spline with clamped boundary conditions generates a smooth trajectory.
- Adaptive Timing: Segment times use velocities from $2.2\,\mathrm{m\,s^{-1}}$ to $4.2\,\mathrm{m\,s^{-1}}$, based on segment lengths ($1.2\,\mathrm{m}$ to $3.2\,\mathrm{m}$).

This aligns with lecture concepts on real-time planning and polynomial trajectory generation, ensuring dynamic adaptation to obstacles.

Differences in the implemented EC Local Planning code:

- Epsilon Value & Margins: The EC code uses a smaller =0.1, retaining more waypoints for finer control in complex environments, reflecting the need for precise navigation with limited sensor range, also I have kept the margin to decrease from 0.38.
- Collision Adjustment: The EC code adds a post-sparsification step in plan_traj to check waypoints for occupancy and snap them to free voxel centers using find_nearest_free_center, enhancing collision avoidance. Project 1.3 assumes the A* path is collision-free, which may fail in dense maps.
- Frequency: Sparsification occurs repeatedly in EC due to replanning, unlike the one-time computation in Project 1.3.
- Error Handling: The EC code checks for uninitialized or invalid splines Project 1.3 assumes valid splines.
- Dynamic Context: The EC code uses relative time (t_rel = t - self.traj_start_time) to support replanning, while Project 1.3 uses absolute time since the trajectory is fixed.
- Stability: The EC's frequent replanning may cause velocity discontinuities at replan points, which Project 1.3 avoids with a single global trajectory.

B. Performance Comparison

Compared to the main task (global planning), the EC planner is slower due to frequent replanning and conservative velocities. For an example, I am considering the the maze map to compare the performance of the 2 implementations from my code.

Method	Planning Time (s)	Flight Time (s)	Flight Distance (m)
Vanilla Implementation	0.9	9.3	29.3
Local Planner (EC)	1.5	10.0	25.2

TABLE I: Comparison of Vanilla Implementation vs. My Local Planner (EC)

However, the EC method excels in dynamic scenarios, adapting to newly discovered obstacles within $5\,\mathrm{m}$, unlike the static global path. Against the simulator baseline (no replanning), the EC planner reduces collisions in complex maps (e.g., maze) by updating the map and replanning, though it struggles in tight spaces.

Other aspects for the performance include the sandbox plots that my EC implementation produced as per my controller implementation, planner and trajectory generator.

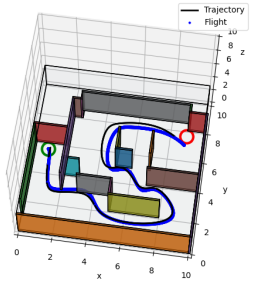

Fig. 7: Extra Credit Maze Trajectory

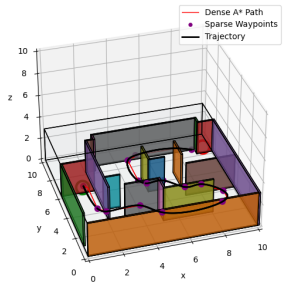

Fig. 8: Extra Credit Maze Planned Path

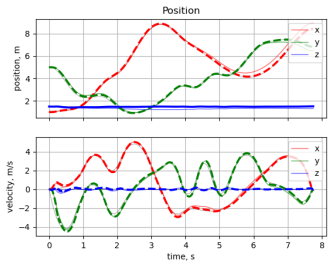

Fig. 9: Extra Credit Maze Poistion vs Time

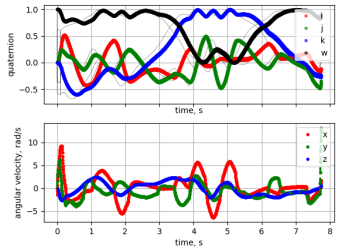

Fig. 10: Extra Credit Maze Orientation vs Time

I also noticed for my codes personally comparing my Lab experience for 1.4, it was pretty easy to pass through thinner spaces using the EC implementation than our 1.4 implementation, like a window space.

C. Failure Cases

The EC implementation faces two main issues:

- Paths Through Obstacles: In a few maps, the path occasionally intersects obstacles due to the $0.2\,\mathrm{m}$ resolution missing small gaps or the $0.5\,\mathrm{m}$ margin blocking narrow passages. The `find_nearest_free_center` method sometimes fails to find a free voxel within $0.6\,\mathrm{m}$ (3 voxels).
- Not Reaching Global Goal: The planner stops short of the goal in many maps for my implementation which I feel is the case for my failing gradescope cases, as the $7.5\,\mathrm{m}$ horizon and $0.1\,\mathrm{s}$ replaning interval cause early termination when local goals are unreachable. For example, I attach the plot for the window map when it kept failing due to stopping at a local goal rather than the global goal. Other failed cases for my gradescope submission include the hidden cases for slalom, stairwell and switchback, which according to me require some tuning in my trajectory parameters incldluing the resolution and epsilon for RDP.

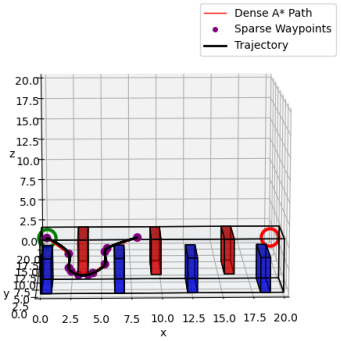

Fig. 11: Planned Path for Over Under

These issues stem from coarse resolution, conservative replanning, and suboptimal goal selection, exacerbated by VIO noise is what I feel for my case and some sort of mishandling for the provided occupancy map.

D. Bibliography

- 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.
- 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. 5
- Skydio, "Skydio 2+." https://www.skydio.com/skydio-2-plus. Autonomous Drones for business, public safety and creative endeavors.
- 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