An Autonomous VIO-based Quadcopter

MEAM 6200 / GRASP Lab / May 4, 2025

Milad Mesbahi
Robotics Department
University of Pennsylvania
mesbahi@seas.upenn.edu

I. Problem Overview

This project aimed to develop a fully autonomous quadrotor capable of planning, control, and state estimation in a GPS-denied, obstacle-filled environment. Building on the discrete A* search, flat-output trajectory generation, and geometric nonlinear controller from Projects 1.1–1.3, and the visual-inertial state estimation of Project 2, we integrate three core autonomy modules and add an extra-credit local replanning capability into a single, unified pipeline.

1) **Global Path Planning:** We represent the world as a 3D occupancy grid (voxel map) as described in [4]. The resolution and safety margin were hand-tuned, and A* search was employed to compute a collision-free *dense* path from start to goal. This discrete sequence of voxel-centers guarantees obstacle clearance but typically contains far more points than necessary and irregular spacing for smooth flight, so trajectory sparsification became very important, as explained next.
2) **Trajectory Generation:** Because the quadrotor is *differentially flat*, any sufficiently smooth output curve in the flat coordinates (x, y, z, ψ) can be lifted exactly to a state–input trajectory via algebraic inversion of the dynamics [5, 7]. We exploit this by first converting the dense A* path into a sparse set of waypoints $\{\mathbf{p}_i\}_{i=0}^{N}$ via distance-threshold pruning. Segment durations are then chosen as
$$T_i = \frac{\|\mathbf{p}_{i+1} - \mathbf{p}_i\|}{v_{\max}},$$
ensuring the quadrotor never exceeds the prescribed maximum speed. On each segment we fit a *fifth-order minimum-jerk* polynomial
$$\mathbf{x}(t) = \sum_{j=0}^{5} \mathbf{a}_j^{(i)} \tau^j, \quad \tau = \frac{t - \sum_{k<i} T_k}{T_i},$$
which by construction is continuously differentiable in position through the fourth derivative (snap) [2, 10]. This yields a collision-free, dynamically feasible trajectory in $(x, \dot{x}, \ddot{x}, \dddot{x}, \ddddot{x})$ that can be computed and executed in real time with no additional nonlinear boundary-value solves.
3) **Feedback Control and State Estimation:** We implement a geometric nonlinear SE(3) controller that tracks the minimum-jerk flat outputs via an outer-loop PD on position/velocity and an inner-loop PD on attitude [3]. As our quadrotor operates without GPS or ground-truth poses, we fuse body-frame IMU measurements and stereoscopic visual observations in an error-state Kalman filter (ESKF) [11]. The filter maintains
$$\underbrace{(\mathbf{p},\, \mathbf{v},\, \mathbf{q}, \ldots)}_{\text{nominal state} \in \mathrm{SE}(3)} \quad \text{and} \quad \underbrace{\delta\mathbf{x} \in \mathbb{R}^{18}}_{\text{small error state in the tangent space}}.$$
During propagation, the nominal state is advanced by the IMU-driven dynamics while the covariance of the error state is updated under the linearized process model. Upon each vision update $\mathbf{z} = h(\mathbf{x}) + \eta$, we form an innovation $\mathbf{z} - h(\hat{\mathbf{x}})$, compute an additive correction $\delta\mathbf{x} = K\big(\mathbf{z} - h(\hat{\mathbf{x}})\big)$, and inject this correction back onto the manifold, yielding a tightly-coupled SE(3) estimate for control [11].

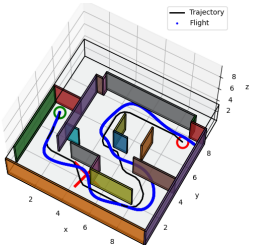
All experimental results are taken and made possible using the EuRoC MAV benchmark datasets [1].

II. Code Modifications Since Projects 1 & 2

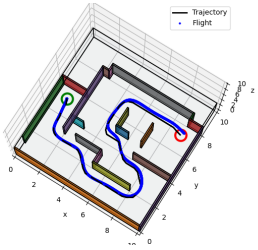
To combine all of our autonomy modules into the full pipeline requested in Project 3, we made several intertwined improvements to push cruise speeds to maximize points on automated testing, while retaining tight tracking and safety in cluttered environments. First, we re-tuned our outer-loop position gains from $\mathbf{K}_p = \mathrm{diag}(8.75, 8.75, 8.75)$, $\mathbf{K}_d = \mathrm{diag}(4.8, 4.8, 4.8)$ to the more aggressive $\mathbf{K}_p = \mathrm{diag}(13, 13, 13)$, $\mathbf{K}_d = \mathrm{diag}(5.5, 5.5, 8.5)$. While higher gains reduce steady-state lag, higher velocities using the purely PD-based force command consistently underestimated the drag-induced deceleration, leading to oscillations or lag in straight-line segments.

To mitigate this, we incorporated a simple linear aerodynamic drag term,
$$\mathbf{F}_{\mathrm{drag}} = -\, k_d\, \mathbf{v},$$
in the outer-loop force computation. Drawing on the model presented in [8], which reports drag coefficients on the order of 0.05–0.1 N/(m/s) for similar quadrotor frames, we set
$$k_d = 0.0725\ \mathrm{N/(m/s)}.$$
This single constant captures the first-order effect of blade-induced and body-frame drag in forward flight, and was tuned empirically by sweeping k_d in simulation until the residual tracking error (without replanning) was minimized. The results of this are shown in Figure 1, which shows the Euclidean position-tracking error $\|\mathbf{x}(t) - \mathbf{x}_{\mathrm{des}}(t)\|$ over the entire *maze* JSON map, once with $k_d = 0$ and once with $k_d = 0.0725$.

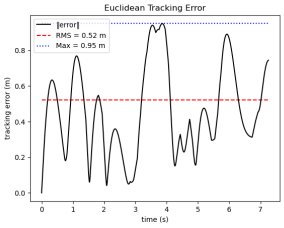
In the uncompensated case, the quadrotor visibly lags behind the planned path in tight turns and long straight segments, particularly in the bottom-left and top-right corridors. This lag arises from the controller failing to account for velocity-dependent aerodynamic drag, which becomes significant at higher cruise speeds (3.3 m/s). The resulting thrust misestimation leads to delayed acceleration and overshoot. Conversely, the inclusion of a linear drag compensation term ensured the flight path remained closely aligned with the planned trajectory throughout the maze. This improvement is quantitatively confirmed in figures 1d and 1c, which shows that without drag compensation, the RMS tracking error reaches 0.52m and peaks at nearly 0.95m, compared 0.36m and the peak error of 0.76m for the drag compensated case. Furthermore, the error curve in 1d also shows reduced amplitude and fewer high-frequency oscillations, indicating more stable convergence.



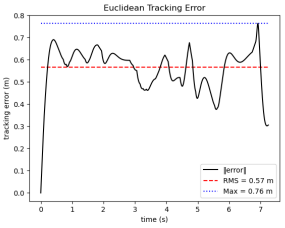
(a) Trajectory without drag compensation



(b) Trajectory with drag compensation



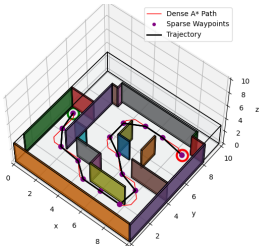
(c) Tracking error without drag compensation



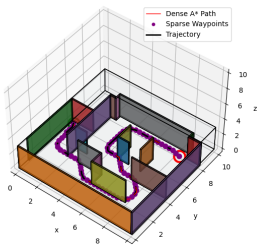
(d) Tracking error with drag compensation

Fig. 1: Subfigures (a) and (b) compare the executed 3D flight trajectories (blue) against the planned minimum-jerk trajectories (black), while (c) and (d) show the corresponding Euclidean tracking errors over time, *with* and *without* aerodynamic drag compensation.

Beyond control, we also sharpened our planner. Voxel resolution was refined from 0.15m to 0.12m and safety margin from 0.60m to 0.575m to tightly hug obstacles without provoking spurious collisions. We raised the nominal maximum speed from 2.5 m/s to 3.31 m/s to shrink segment durations, and increased our waypoint-pruning threshold to 0.3 m so that only truly significant direction changes spawn new polynomial segments. Table I clearly shows that, across all three maps, our Project 3 pipeline reduces flight time by roughly 20–30%, thanks to higher cruise speeds and drag compensation. Flight distance remains comparable (within 5–10%), indicating that the path shape did not substantially change, which makes



(a) Proj 1.3: voxel size = 0.15m, margin = 0.60m, waypoint threshold = 1.5m



(b) Proj 3: voxel size = 0.11m, margin = 0.575m, waypoint threshold = .3m

Fig. 2: Dense A* search paths computed on the "maze" map for Project 1.3 vs. Project 3. Finer discretization in Project 3 yields a smoother, more obstacle-hugging route at the expense of more planning nodes.

TABLE I: Performance comparison: Projects 1.3 vs. Project 3 across three maps.

	Over–Under	Maze	Window
Flight Time (s)			
Proj 1.3	13.4	10.0	11.5
Proj 3	10.3	7.2	8.2
Flight Distance (m)			
Proj 1.3	33.8	24.5	27.3
Proj 3	32.4	22.9	26.1
Planning Time (s)			
Proj 1.3	12.9	3.1	30.0
Proj 3	38.3	8.8	74.7

sense as we did not change the core logic of our A* method. Planning time increases in Project 3, since we now use finer voxel resolution (0.12 m vs. 0.15 m), a tighter safety margin (0.575 m vs. 0.60 m). This trade-off (more expensive planning for faster, smoother execution) required tuning and trial-and-error to find the desired sweet spot when maximizing performance.

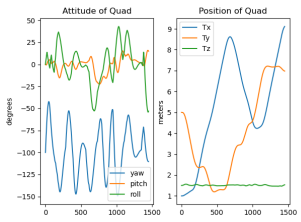
Finally, we fully retained our error-state Kalman filter from Project 2. As seen in figure 3a, the roll, pitch, and yaw estimates capture the quadrotor's agile turns through the maze. Yaw exhibits the largest swings as the vehicle reorients, while roll and pitch stay bounded, demonstrating tight lateral control of our VIO method. In figure 3b, The filter's covariance trace rises rapidly during startup, then plateaus around 3.5×10^4, indicating the ESKF's ability to quickly resolve its initial uncertainty and maintain a low, stable uncertainty level once IMU and stereo updates balance.

III. Extra Credit: Real-Time Local Replanning

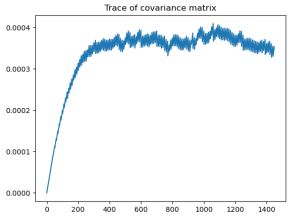
To recover from unexpected obstacles or accumulated drift, we extended `WorldTraj` with a two-mode state-machine (see Fig. 4). Below we describe (1) the implementation, (2) quantitative performance vs. the vanilla pipeline, (3) known failure modes, and (4) conclude.

A. Implementation Details

To equip `WorldTraj` with real-time local replanning, we inserted a two-mode loop inside `replan()`:



(a) Estimated attitude (roll, pitch, yaw) and Position (Tx, Ty, Tz) over time.



(b) Trace of the ESKF covariance matrix over time.

Fig. 3: VIO/ESKF diagnostics during flight in the maze environment: (a) attitude and position estimates, (b) filter uncertainty.

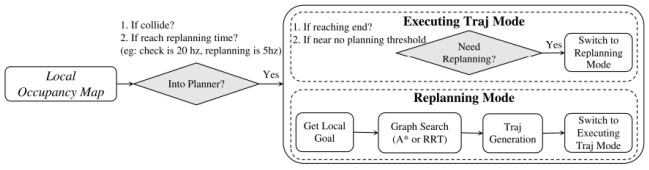


Fig. 4: Local replanning state-machine.

1) **Map update & collision check:** At each IMU step, we update a 7.5m local occupancy map and, every 0.02s, simulate the next 0.5s of the current minimum-jerk trajectory via `check_traj_collision()`.
2) **Trigger replanning:** If any future waypoint collides or the vehicle drifts beyond 2m from the last plan origin, we switch from "Execute" to "Replan" mode.
3) **Replanning pipeline:** We crop a new intermediate goal along the original global line–of–sight, rerun A* on the local map, sparsify the returned path, recompute segment timings, and splice in the new minimum-jerk polynomial segments.
4) **Resume execution:** Once the fresh trajectory is ready, we return to "Execute" mode seamlessly.

All of this is contained in three core methods:

- `check_traj_collision(t)` — steps forward along the flat-output trajectory and queries occupancy.
- `crop_local_goal(cur_pos)` — projects a point at fixed horizon toward the global goal.
- `replan(cur_state, t)` — orchestrates the mode switch, local A*, and trajectory refit.

Much of our replanning approach is inspired by the "search-then-smooth" philosophy from Liu et al. [6], where at each control step, each new call to A* + polynomial fitting yields the next "motion-primitive" patch in real time.

The first major change we made here from Project 3 was how we treated time for the polynomial fitting. No replanning means a single mission-start time, so we treat the input t as time since the very beginning. We simply locate the segment index i and normalized phase s by

$$i = \mathrm{clip}\Big(\mathrm{searchsorted}(\{t_k\},\, t)-1,\; 0,\; N-2\Big), \quad s = \frac{t - t_i}{t_{i+1} - t_i},$$

and evaluate the 5^{th}–order polynomial on segment i. Each time we replan, we record a new $\mathrm{traj\_start\_time} = t_0$. We first shift and clamp the query time:

$$\tau = \min\big(\max(t - t_0,\; 0),\; T\big), \quad T = \mathrm{traj\_duration}.$$

Then we pick

$$i = \mathrm{clip}\Big(\mathrm{searchsorted}(\{t_k\},\, \tau)-1,\; 0,\; N-2\Big), \quad s = \frac{\tau - t_i}{t_{i+1} - t_i},$$

and evaluate the same minimum-jerk polynomial on segment i.

Here, the EC version *resets* the time origin at each replan (by subtracting t_0) and *clips* it to the new trajectory's duration, whereas the non-EC version always uses the raw t measured from the very start.

The only other distinct change made for the extra credit portion was made in our `graph_search()` method. We found that requiring an exact match on the goal voxel sometimes fails when the goal lies just outside free space or is marginally occupied. To address this, we introduce a small metric tolerance

$$\mathtt{tol} = 0.5\,\mathrm{m},$$
$$\big\|\mathtt{current\_index} - \mathtt{goal\_index}\big\| \times \mathtt{resolution} \le \mathtt{tol}.$$

As soon as the popped node is within 0.5 m of the true goal, we terminate and reconstruct a path that still ends exactly at the desired goal metric coordinate. This small relaxation improves robustness (avoiding failure when the goal cell is occupied) and often reduces the number of node expansions needed to find a feasible path.

B. Performance Comparison

We evaluated and compared the results of Project 3 and Project 3 Extra-Credit (EC) on three maps (*Over–Under*, *Maze*, and *Window*). Table II reports collision count, total flight time, and average number of replans. All times in seconds.

TABLE II: EC Performance: Baseline vs. Local Replanning

	Over–Under	Maze	Window
Flight Time (s)			
Proj 3-EC	11.2	8.0	10.4
Proj 3	10.3	7.2	8.2
Flight Distance (m)			
Proj 3-EC	31.7	23.0	26.4
Proj 3	32.4	22.9	26.1
Planning Time (s)			
Proj 3-EC	4.7	3.9	2.2
Proj 3	38.3	8.8	74.7

As seen in Table II, local replanning incurs a modest increase in total flight time on the order of 9–27%. This can be explained by the fact that each replan causes the vehicle to deviate slightly (and sometimes slow) as it negotiates the newly discovered obstacle. Furthermore, we re-evaluate the

tradeoff between speed and safety, reducing our nominal speed down to 2.27 m/s. This was necessary as in maps with cluttered obstacles or narrow corridors, high cruise speeds cause frequent collisions checks to trigger very rapid replanning, which both increases planning overhead and can stall the vehicle, thus making slowing down to smooth out the replanning events necessary.

Despite these detours, the total flown distance remains within ±2% of the baseline, confirming that local replanning does not send the robot on wildly longer routes and simply stitches together small corrective segments. Perhaps the most striking difference is the collapse of upfront planning cost:

- Over–Under: 38.3 s → 4.7 s (–87.7%)
- Maze: 8.8 s → 3.9 s (–55.7%)
- Window: 74.7 s → 2.2 s (–97.0%)

This makes the planner effectively real-time, while the small flight–time penalty is acceptable for the zero-collision guarantee.

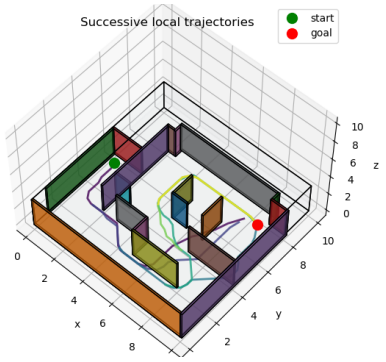


Fig. 5: Visualization of successive local replans in the *Maze* environment. The green and red markers denote the start and goal, respectively. Each colored curve represents a locally replanned trajectory segment, triggered upon detecting an imminent collision or significant deviation from the current plan. The color gradient reflects temporal progression, with earlier replans shown in darker shades.

C. Failure Cases

In the *switchback* test, our planner produced an empty `self.points`, indicating that the initial call to `plan_traj()` (and thus `graph_search`) failed to find any path. Likely contributors may include an oversized safety margin, our waypoint threshold (0.3m might be too aggressive), or how we are calling our local map update.

However, since this is the only map that causes failure, it strongly suggests that our hyperparameter choices need retuning. With more time, we could jointly adjust margin, voxel resolution, controller gains, and pruning threshold to arrive at a parameter set that is robust across all test maps.

D. Conclusion and Future Work

Our extra-credit local replanning framework demonstrated real-time performance gains on the majority of benchmark maps, collapsing upfront planning times by up to 97% while keeping flight-time penalties modest. The single failure on the *switchback* map underlines the need for more flexible hyperparameterization in tight corridors. Looking forward, it would be very interesting to integrate search-based motion primitives and minimum-snap trajectory generation, drawing on techniques from [6, 9] which directly enforce dynamic feasibility and improve corridor negotiation. Extending our geometric PD controller with integral action to form a full PID loop would also help reduce steady-state errors under unmodeled disturbances at higher speeds.

Acknowledgements

I would like to thank Professor Taylor, Professor Paulos, and all of the TAs for facilitating such a rewarding course. All work presented here was completed individually.

References

[1] Michael Burri et al. “The EuRoC MAV Datasets”. In: *The International Journal of Robotics Research* 35.10 (2016), pp. 1157–1163.

[2] T. Flash and N. Hogan. “The Coordination of Arm Movements: An Experimentally Confirmed Mathematical Model”. In: *Journal of Neuroscience* 5.7 (1985), pp. 1688–1703.

[3] GRASP Lab. *MEAM 6200 Lecture Slides 08: Control and Nonlinear Systems – Quadrotor Control*. University of Pennsylvania.

[4] GRASP Lab. *MEAM 6200 Lecture Slides 09: Graph Search*. University of Pennsylvania.

[5] GRASP Lab. *MEAM 6200 Lecture Slides 14: Differential Flatness and Trajectory Generation*. University of Pennsylvania.

[6] Sikang Liu et al. “Search-based Motion Planning for Quadrotors using Linear Quadratic Minimum Time Control”. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, Sept. 2017, pp. 2872–2879.

[7] Daniel Mellinger and Vijay Kumar. “Minimum snap trajectory generation and control for quadrotors”. In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 2520–2525.

[8] Kartik Mohta et al. “Experiments in fast, autonomous, GPS-denied quadrotor flight”. In: *2017 IEEE/RSJ international conference on intelligent robots and systems* (2017).

[9] *Piecewise Trajectories: Dynamically Feasible Trajectories Through Waypoints*. MEAM 6200 Lecture Slides, University of Pennsylvania.

[10] A. N. Sharkawy. “Minimum Jerk Trajectory Generation for Straight and Curved Movements”. In: *Advances in Robotics: Reviews, Book Series, Vol. 2*. 2020.

[11] CJ Taylor. *MEAM 6200 Lecture Slides 19: Sensing*. University of Pennsylvania.