An Autonomous VIO-based Quadcopter

Introduction

The project integrates all trajectory planning and control modules of SE(3) to stereo visual-inertial odometry (VIO) with an Error-State Kalman Filter (ESKF). The objective is full end-to-end integration to obtain an onboard state estimation, quad rotor planning, and control.

Controller Design

Implemented a geometric nonlinear PD controller designed to align the body-frame thrust axis with the desired force direction to enable agile maneuvers. The controller consists of two main loops: a position controller, which gets the desired acceleration, and a rotation controller, which computes the desired torques. The controller is modelled as:

$$\ddot{r}_{\text{des}} = \ddot{r}_T - K_d(\dot{r} - \dot{r}_T) - K_p(r - r_T)$$

$$F_{\text{des}} = m \cdot \ddot{r}_{\text{des}} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \qquad b_3 = R \cdot \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \qquad u_1 = b_3^\top \cdot F_{\text{des}}$$

$$R_{\text{des}} = \begin{bmatrix} b_{1,\text{des}} & b_{2,\text{des}} & b_{3,\text{des}} \end{bmatrix}$$

$$e_R = \frac{1}{2} \cdot \left(R_{\text{des}}^\top R - R^\top R_{\text{des}}\right)^\vee \qquad e_\omega = \omega$$

$$u_2 = I \cdot (-K_R e_R - K_\omega e_\omega)$$

$$\begin{bmatrix} u_1 \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = A \cdot \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}, \qquad \omega_i = \sqrt{\max(0, \omega_i^2)}$$

The meaning of the equations and the purpose are concisely mentioned as follows, intending to build an intuition of the nature of the values and their effect. The nonlinear SE(3) controller computes thrust and moments for trajectory tracking. Desired acceleration is calculated using position and velocity errors. Which is used further to get the net force F_{des}, from which thrust u_1 is obtained by projecting onto the body's z-axis. The desired orientation matrix R_{des} is constructed using F_{des} the desired yaw. Orientation errore_R and angular velocity error e_ω are then used to compute the control moment u_2 $K_p K_d K_R$ *and* K_ω defines how strongly the controller reacts to errors in position, velocity, orientation, and angular velocity, respectively. The tuning process began with manual trial-and-error by modifying one gain at a time and observing the system response. Initially, equal gains were used across all three axes, but this led to bottlenecks, as specific axes required different levels of responsiveness. To address this, the tuning approach was restructured: gains were scaled independently per axis and parameterized using a base value, damping ratio, and scaling constants. This enabled finer control and iterative refinement. The values were chosen based on simulation feedback, with saturation and instability used as indicators for retuning. While Ziegler–Nichols tuning was explored, it lacked the precision needed. For Project 3, the gains from HW1 were retained. In lab testing, further adjustments were needed to account for real-world constraints like actuator limits and motor speed caps, as noted in Table 1, demonstrating the changes made from project 1 to project 3 and also stating the sim-to-real gap.

Overall, for efficient iterative tuning, the mentioned correlation was used involving ξ = 4.5, interdependency of Kω = constant * Kr, and similarly, the equation of Kp = base * diag[...] and Kd = ξ * $(base)^{0.5}$ * diag[...] were used.

HWs	Kp	Kd	Kω	Kr
1	[7.4, 7.4, 20.35]	[3.9, 3.9, 6.3]	[18, 18, 1] * *180 * 0.05	[18, 18, 1] * 180
Lab 1.4	[4, 4, 4.25]	[2.5, 2.5, 2.5]	[6.5, 6.5, 6.5]	[63, 63, 63]

Table 1: Gains Tuned

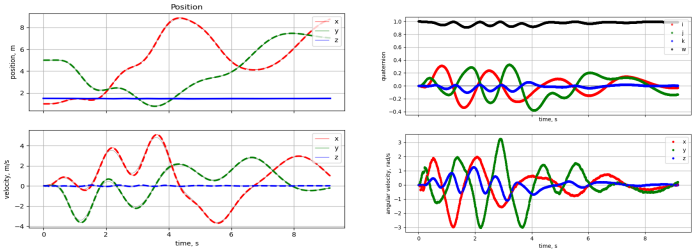

Fig 1 : (left)Velocity and position in m/s and m. and (right) Orientation and angular velocity vs time

Path Planning

Path planning was implemented using the A* algorithm. The heuristic used was Euclidean distance, and nodes were explored only if they were within the map bounds and outside obstacles. The A* cost function is defined as: $f(n) = g(n) + h(n)$. where $g(n)$ is the cost from the start to the node n, and $h(n)$ is the heuristic cost from n to the goal. While experimenting with its implementation, it was observed that it produced a lot of nearby waypoints and which eventually would create a very jagged path for the drone when the trajectories are generated. To reduce the number of waypoints and smooth the path, I applied the Ramer–Douglas–Peucker (RDP) algorithm, which prunes intermediate waypoints based on a fixed threshold. It gave me a more compact set of key waypoints for trajectory generation. Tuning the resolution and margin was critical. A finer resolution allowed better path quality and more aggressive maneuvers by giving the drone more clearance, while higher margins were necessary to avoid collisions at higher speeds. After extensive trial and error, for the project I used: Resolution = [0.22, 0.11, 0.12], Margin = 0.6, with Velocity = 2.8 m/s. These values offered a good trade-off between computation time and trajectory feasibility while

maintaining safe clearance from obstacles. In the lab, we constructed the map with Resolution = [0.2, 0.2, 0.2] m, Margin = 0.2 m to account for tracking noise and controller inaccuracy which was observed a lot due to Vicon. A* was used for path planning, and RDP (Ɛ = 0.1 m) was applied. Tuning resolution and margin together was necessary for scenarios like flying through tight spaces (e.g., the window). For the final integration project 4, as it had VIO, the values taken were Resolution = [0.22, 0.11, 0.12]m, Margin = 0.5 m, and RDP(Ɛ = 0.2 m). Different values were selected as the trajectory generation method changed from project 1 and project 3 because of time constraints Which is described further.

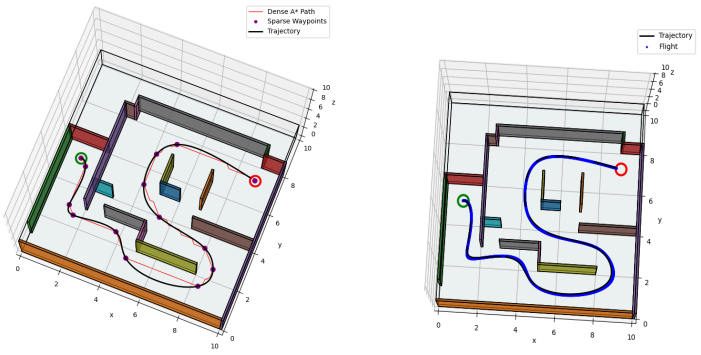

Fig2: (left) A* path generation and RDP pruning (threshold =0.2), (right 3D flight path using min snap) in meters

Trajectory Generation

Trajectory generation is performed using a minimum snap strategy where each segment between waypoints is modeled as a 7th-degree polynomial. The trajectory function first allocates time to each segment based on inter-point Euclidean distances and the target velocity. It then solves for polynomial coefficients using solvetraj, which applies position constraints at segment boundaries and uses a least-squares solution to fit the trajectory. At runtime, the trajectory function evaluates the segment active at the given time to return position, velocity, and acceleration. To ensure smooth convergence near the end of the path, the drone's velocity is scaled down for the final segment. Higher-order derivatives (jerk and snap) are set to zero in this implementation to reduce complexity. The update function packages this data into the desired flat output format for the controller. Below is the process:

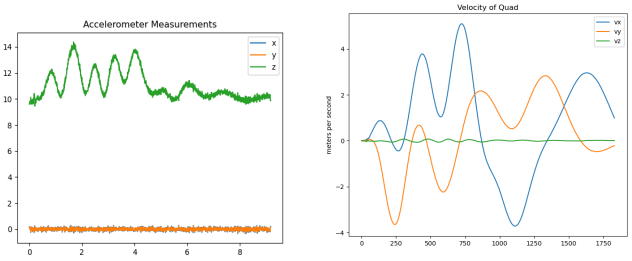

Fig3: (left) Accelerometer estimate, (right) velocity estimate

1. 7th-degree Polynomial Basis Vector:

$$\phi(t) = [1,\ t,\ t^2,\ t^3,\ t^4,\ t^5,\ t^6,\ t^7]$$

2. Linear System to Solve Coefficients for Segment i:

$$A = \begin{bmatrix} \phi(t_0) \\ \phi(t_1) \end{bmatrix}, \quad b = \begin{bmatrix} \text{waypoint}_i \\ \text{waypoint}_{i+1} \end{bmatrix}$$

$$\text{Solve:} \quad A \cdot c_i = b \quad \text{(Least squares)}$$

3. Segment Time Allocation:

$$T_i = \frac{\|\text{waypoint}_{i+1} - \text{waypoint}_i\|}{\text{velocity}}$$

4. Trajectory Evaluation at Time t in Segment i:

$$\text{position} = \text{waypoint}_i + \text{unit\_direction} \cdot (t - t_{\text{start}})$$

$$\text{velocity} = \text{unit\_direction} \cdot \text{velocity}$$

The issue I faced was that I could not increase the speed of he drone and didn't have precise control over it, and just minor disturbances would push the drone into an unstable state. A similar approach was used in the Lab session as well. However, to improve the performance for project 3, I implemented min snap trajectory generation. The minimum snap method models each trajectory segment as a 7th-degree polynomial and solves for coefficients that minimize high-order derivatives, ensuring smoother motion. Unlike earlier linear methods, it provided continuity in position and velocity while reducing jerk and snap $A\ .\ c\ =\ b$, where A encodes polynomial basis constraints (e.g., position at segment ends), c contains the polynomial coefficients, and b represents the waypoint values. As a direct effect, I was able to use higher speeds and much control over the drone. The speed regimes were manually tuned, improved from approx.. 2.3 to 2.4m/s to 8m/s, and also better control over acceleration.

VIO and ESKF

The ESKF combines stereo visual odometry and IMU information to estimate the complete state of the drone. The procedure:

- Nominal state update: Advances the position, velocity, and orientation, along with the help of IMU acceleration and angular rate through kinematic motion equations.
- Covariance Update: Updates the covariance matrix for errors using the system process model and the existing nominal state, adding the process noise.
- Measurement Update: Updates the nominal state from stereo visual observations. If the innovation (residual) is less than the threshold, it calculates the Kalman gain and makes the correction. Outlier data are rejected.

No structural modifications were introduced to the VIO implementation of Project 2. The ESKF yielded stable and quickly converging state estimates during the entire project.