Quadrotor Simulation with Differential Flatness Control and Motion Planning Module

Based on Tal & Karaman (2021)

Technical Report

January 6, 2026

Abstract

This report provides a comprehensive documentation of a quadrotor simulation implementing differential flatness-based trajectory tracking control following Tal & Karaman (2021). The simulation features a 6-DOF quadrotor with cascaded control architecture using Incremental Nonlinear Dynamic Inversion (INDI). Additionally, a motion planning module has been developed to enable autonomous navigation through RRT-based path planning and minimum-snap trajectory generation. Three mission types are supported: point-to-point navigation, static obstacle avoidance, and dynamic replanning for pop-up threats.

Contents

1 Introduction

This simulation implements the control architecture described in Tal & Karaman (2021) [1], which presents an accurate tracking method for aggressive quadrotor trajectories using incremental nonlinear dynamic inversion combined with differential flatness. The key contributions include:

1. A cascaded control structure with position (outer) and attitude (inner) loops
2. Incremental control formulation that reduces model dependency
3. Differential flatness for computing feedforward angular velocity and acceleration
4. Extension with RRT-based motion planning for autonomous navigation

1.1 Coordinate System

The simulation uses the North-East-Down (NED) coordinate frame:

- **Inertial Frame:** $\mathcal{I} = \{N, E, D\}$ with D pointing downward
- **Body Frame:** $\mathcal{B} = \{b_x, b_y, b_z\}$ with b_x forward, b_y right, b_z down
- **Position:** $\mathbf{x} = [N, E, D]^T$ (meters)
- **Velocity:** $\mathbf{v} = [\dot{N}, \dot{E}, \dot{D}]^T$ (m/s)
- **Attitude:** Quaternion $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ (body$\rightarrow$inertial)
- **Angular Velocity:** $\boldsymbol{\Omega} = [p, q, r]^T$ (body frame, rad/s)

2 Quadrotor Dynamics

2.1 Translational Dynamics

The translational dynamics of the quadrotor in the inertial frame are given by Newton's second law:

$$m\ddot{\mathbf{x}} = m\mathbf{g} + \mathbf{f} \tag{1}$$

where:

- m is the total mass of the quadrotor
- $\mathbf{g} = [0, 0, g]^T$ is the gravitational acceleration vector in NED frame
- $\mathbf{f}$ is the total thrust force expressed in the inertial frame

The thrust force is generated along the negative body z-axis and transformed to the inertial frame:

$$\mathbf{f} = -T \cdot \mathbf{b}_z = -T \cdot R\mathbf{e}_3 \tag{2}$$

where T is the thrust magnitude, R is the rotation matrix from body to inertial frame, and $\mathbf{e}_3 = [0, 0, 1]^T$.

The acceleration in the inertial frame becomes:

$$\ddot{\mathbf{x}} = g\mathbf{e}_3 - \frac{T}{m}\mathbf{b}_z \tag{3}$$

2.2 Rotational Dynamics

The rotational dynamics are governed by Euler's equation in the body frame:

$$J\dot{\boldsymbol{\Omega}} = \boldsymbol{\mu} - \boldsymbol{\Omega} \times J\boldsymbol{\Omega} \tag{4}$$

where:

- $J = \text{diag}(J_{xx}, J_{yy}, J_{zz})$ is the inertia tensor (assuming principal axes)
- $\boldsymbol{\mu} = [\mu_x, \mu_y, \mu_z]^T$ is the control torque vector
- $\boldsymbol{\Omega} \times J\boldsymbol{\Omega}$ is the gyroscopic coupling term

2.3 Quaternion Kinematics

The attitude is represented using unit quaternions $\mathbf{q} = [q_w, q_x, q_y, q_z]^T$ with $\|\mathbf{q}\| = 1$. The quaternion derivative is:

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{\Omega} \end{bmatrix} \tag{5}$$

where $\otimes$ denotes quaternion multiplication. The rotation matrix from body to inertial frame is:

$$R(\mathbf{q}) = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix} \tag{6}$$

2.4 Motor Model

Each motor is modeled as a first-order system with time constant τ_m:

$$\dot{\omega}_i = \frac{1}{\tau_m}(\omega_{i,\text{cmd}} - \omega_i) \tag{7}$$

The thrust and torque generated by motor i are:

$$T_i = k_T \omega_i^2 \tag{8}$$

$$\tau_i = \pm k_M \omega_i^2 \tag{9}$$

where k_T is the thrust coefficient, k_M is the moment coefficient, and the sign depends on the motor rotation direction.

2.5 Control Allocation

The relationship between motor speeds and the control inputs is given by the control effectiveness matrix G_1:

$$\begin{bmatrix} \mu_x \\ \mu_y \\ \mu_z \\ T \end{bmatrix} = G_1 \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \tag{10}$$

For an X-configuration quadrotor with arm lengths l_x and l_y:

$$G_1 = \begin{bmatrix} l_y k_T & -l_y k_T & -l_y k_T & l_y k_T \\ l_x k_T & l_x k_T & -l_x k_T & -l_x k_T \\ -k_M & k_M & -k_M & k_M \\ k_T & k_T & k_T & k_T \end{bmatrix} \tag{11}$$

The motor speed commands are obtained by inverting this relationship:

$$\boldsymbol{\omega}^2 = G_1^{-1} \begin{bmatrix} \boldsymbol{\mu}_{\text{cmd}} \\ T_{\text{cmd}} \end{bmatrix} \tag{12}$$

with saturation limits $\omega_{\min} \leq \omega_i \leq \omega_{\max}$.

3 Differential Flatness

3.1 Flat Outputs

A system is differentially flat if there exist outputs (flat outputs) such that all states and inputs can be expressed as functions of the flat outputs and their derivatives. For a quadrotor, the flat outputs are:

$$\boldsymbol{\sigma} = [\mathbf{x}^T, \psi]^T = [x, y, z, \psi]^T \tag{13}$$

where $\mathbf{x}$ is the position and ψ is the yaw angle.

This means that given a trajectory $\boldsymbol{\sigma}(t)$ with sufficient derivatives, all states and inputs can be computed algebraically without integration.

3.2 Thrust Vector from Trajectory

From the translational dynamics (Eq. 3), the specific thrust vector (thrust per unit mass) is:

$$\boldsymbol{\tau} = \ddot{\mathbf{x}} - g\mathbf{e}_3 = -\frac{T}{m}\mathbf{b}_z \tag{14}$$

The thrust magnitude is:

$$\tau = \|\boldsymbol{\tau}\| = \frac{T}{m} \tag{15}$$

And the body z-axis direction is:

$$\mathbf{b}_z = -\frac{\boldsymbol{\tau}}{\tau} = -\frac{\ddot{\mathbf{x}} - g\mathbf{e}_3}{\|\ddot{\mathbf{x}} - g\mathbf{e}_3\|} \tag{16}$$

3.3 Attitude from Thrust and Yaw

Given the desired thrust direction $\mathbf{b}_z$ and yaw angle ψ, the complete rotation matrix is constructed as:

$$\mathbf{b}_x = \frac{[\cos\psi, \sin\psi, 0]^T \times \mathbf{b}_z}{\|[\cos\psi, \sin\psi, 0]^T \times \mathbf{b}_z\|} \times \mathbf{b}_z \tag{17}$$

$$\mathbf{b}_y = \mathbf{b}_z \times \mathbf{b}_x \tag{18}$$

$$R = [\mathbf{b}_x, \mathbf{b}_y, \mathbf{b}_z] \tag{19}$$

3.4 Angular Velocity from Jerk (Equation 14)

Following Tal & Karaman (2021), the angular velocity reference is computed from the jerk (third derivative of position). Define the matrices:

$$[\mathbf{i}_z]_\times = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{20}$$

The yaw rate mapping matrix S relates yaw rate to body rates:

$$S = \frac{1}{b_{x,1}^2 + b_{x,2}^2} \begin{bmatrix} -b_{x,2} & b_{x,1} \end{bmatrix} \begin{bmatrix} 0 & -b_{z,1} & b_{y,1} \\ 0 & -b_{z,2} & b_{y,2} \end{bmatrix} \tag{21}$$

where $b_{x,i}$ denotes the i-th component of $\mathbf{b}_x$.
The angular velocity reference $\mathbf{\Omega}_{\text{ref}}$ and thrust rate $\dot{\tau}$ are obtained by solving:

$$\begin{bmatrix} \tau R[\mathbf{i}_z]_\times^T & \mathbf{b}_z \\ S & 0 \end{bmatrix} \begin{bmatrix} \mathbf{\Omega}_{\text{ref}} \\ \dot{\tau} \end{bmatrix} = \begin{bmatrix} \dddot{\mathbf{x}} \\ \dot{\psi} \end{bmatrix} \tag{22}$$

3.5 Angular Acceleration from Snap (Equation 15)

The angular acceleration reference $\dot{\mathbf{\Omega}}_{\text{ref}}$ is computed from the snap (fourth derivative of position):

$$\begin{bmatrix} \tau R[\mathbf{i}_z]_\times^T & \mathbf{b}_z \\ S & 0 \end{bmatrix} \begin{bmatrix} \dot{\mathbf{\Omega}}_{\text{ref}} \\ \ddot{\tau} \end{bmatrix} = \begin{bmatrix} \ddddot{\mathbf{x}} \\ \ddot{\psi} \end{bmatrix} - \begin{bmatrix} R(2\dot{\tau} I + \tau[\mathbf{\Omega}]_\times)[\mathbf{i}_z]_\times^T \mathbf{\Omega} \\ 0 \end{bmatrix} \tag{23}$$

where $[\mathbf{\Omega}]_\times$ is the skew-symmetric matrix of $\mathbf{\Omega}$.

4 Control Architecture

The control system uses a cascaded structure with an outer position loop and an inner attitude loop.

$$\text{Trajectory} \xrightarrow{\text{Position}} \mathbf{a}_c \xrightarrow{\text{Attitude}} \mathbf{q}_{\text{inc}} \xrightarrow{\text{Flatness}} (\mathbf{\Omega}_{\text{ref}}, \dot{\mathbf{\Omega}}_{\text{ref}}) \xrightarrow{\text{PD}} \boldsymbol{\alpha}_{\text{cmd}} \xrightarrow{\text{Motor}} \boldsymbol{\omega}_{\text{mot}} \tag{24}$$

4.1 Position Control (Outer Loop)

The position controller computes a commanded acceleration using PD control with acceleration feedforward (Equation 17 from [1]):

$$\mathbf{a}_c = K_x(\mathbf{x}_{\text{ref}} - \mathbf{x}) + K_v(\mathbf{v}_{\text{ref}} - \mathbf{v}) + K_a(\mathbf{a}_{\text{ref}} - \mathbf{a}_f) + \mathbf{a}_{\text{ref}} \tag{25}$$

where:

- $K_x = \text{diag}([18, 18, 13.5])$ - Position gain matrix
- $K_v = \text{diag}([7.8, 7.8, 5.9])$ - Velocity gain matrix
- $K_a = \text{diag}([0.5, 0.5, 0.3])$ - Acceleration gain matrix
- $\mathbf{a}_f$ - Filtered acceleration (from accelerometer)

The acceleration filter is a first-order low-pass filter with cutoff frequency 30 Hz:

$$\dot{\mathbf{a}}_f = \frac{1}{\tau_f}(\mathbf{a}_{\text{meas}} - \mathbf{a}_f), \quad \tau_f = \frac{1}{2\pi \cdot 30} \tag{26}$$

4.2 Incremental Thrust Command

The incremental specific thrust command is computed as (Equations 19-21):

$$\boldsymbol{\tau}_{bz,c} = \boldsymbol{\tau}_{bz,f} + \mathbf{a}_c - \mathbf{a}_f \tag{27}$$

where $\boldsymbol{\tau}_{bz,f}$ is the filtered specific thrust. The thrust command is:

$$T_{\text{cmd}} = m\|\boldsymbol{\tau}_{bz,c}\| \tag{28}$$

4.3 Incremental Attitude Command (Equations 22-26)

The attitude command is computed incrementally in the body frame. Given the current rotation matrix R and desired thrust direction $\boldsymbol{\tau}_{bz,c}$:

Step 1: Compute desired body z-axis in inertial frame:

$$\mathbf{b}_{z,\text{des}} = -\frac{\boldsymbol{\tau}_{bz,c}}{\|\boldsymbol{\tau}_{bz,c}\|} \tag{29}$$

Step 2: Transform to body frame:

$$\mathbf{b}^B_{z,\text{des}} = R^T \mathbf{b}_{z,\text{des}} \tag{30}$$

Step 3: Compute tilt quaternion (rotation from current $[0,0,1]^T$ to $\mathbf{b}^B_{z,\text{des}}$):

$$q_{\text{tilt}} = \frac{1}{\sqrt{2(1+d)}} \begin{bmatrix} \sqrt{2(1+d)}/2 \\ \mathbf{e}_3 \times \mathbf{b}^B_{z,\text{des}} \end{bmatrix} \tag{31}$$

where $d = \mathbf{e}_3 \cdot \mathbf{b}^B_{z,\text{des}}$.

Step 4: Compute yaw correction:

$$\psi_{\text{err}} = \psi_{\text{ref}} - \text{atan2}(R_{\text{new}}(2,1), R_{\text{new}}(1,1)) \tag{32}$$

$$q_{\text{yaw}} = [\cos(\psi_{\text{err}}/2), 0, 0, \sin(\psi_{\text{err}}/2)]^T \tag{33}$$

Step 5: Total incremental command:

$$q_{\text{inc}} = q_{\text{tilt}} \otimes q_{\text{yaw}} \tag{34}$$

4.4 Attitude Control (Inner Loop)

The attitude controller uses quaternion-based PD control with angular acceleration feedforward:

$$\boldsymbol{\alpha}_{\text{cmd}} = \dot{\boldsymbol{\Omega}}_{\text{ref}} + K_\xi \boldsymbol{\xi}_e + K_\omega(\boldsymbol{\Omega}_{\text{ref}} - \boldsymbol{\Omega}) \tag{35}$$

where:

- $K_\xi = \text{diag}([175, 175, 82])$ - Attitude error gain
- $K_\omega = 0.8 \cdot \text{diag}([19, 19.5, 19.2])$ - Angular rate gain
- $\boldsymbol{\xi}_e$ - Attitude error in axis-angle representation

The attitude error is computed from the error quaternion:

$$q_e = q_{\text{cmd}} \otimes q^* \tag{36}$$

where q^* is the conjugate of the current quaternion. The axis-angle error is:

$$\boldsymbol{\xi}_e = 2\arccos(q_{e,w}) \cdot \frac{\mathbf{q}_{e,v}}{\sin(\arccos(q_{e,w}))} \tag{37}$$

4.5 Torque Command

The body torque command is computed from the angular acceleration command:

$$\boldsymbol{\mu}_{\text{cmd}} = J\boldsymbol{\alpha}_{\text{cmd}} + \boldsymbol{\Omega} \times J\boldsymbol{\Omega} \tag{38}$$

5 Yaw Planning

Three yaw planning modes are implemented:

5.1 Constant Yaw

$$\psi(t) = \psi_0, \quad \dot{\psi}(t) = 0, \quad \ddot{\psi}(t) = 0 \tag{39}$$

5.2 Tangent Yaw

The yaw angle follows the velocity direction:

$$\psi(t) = \text{atan2}(v_E, v_N) \tag{40}$$

The yaw rate is computed analytically:

$$\dot{\psi} = \frac{v_N a_E - v_E a_N}{v_N^2 + v_E^2} \tag{41}$$

5.3 Coordinated Yaw

Similar to tangent yaw but with smoothing for aggressive maneuvers.

6 Motion Planning Module

The motion planner extends the simulation with autonomous navigation capabilities.

6.1 System Architecture

Table 1: Motion Planner Module Components

Module	**File**	**Function**
Collision Checker	`collision_checker.m`	Real-time collision detection
Obstacle Manager	`obstacle_manager.m`	Static/dynamic obstacle handling
RRT Planner	`rrt_planner.m`	Path planning algorithm
Trajectory Smoother	`trajectory_smoother.m`	Minimum-snap trajectory generation
Waypoint Manager	`waypoint_manager.m`	Mission sequencing and replanning

6.2 Mission Types

1. **Task 1:** Point-to-point navigation without obstacles
2. **Task 2:** Navigation around static obstacles using RRT
3. **Task 3:** Dynamic replanning to avoid pop-up threats during flight

7 Collision Detection

7.1 Sphere Obstacles

For a sphere obstacle centered at $\mathbf{c}$ with radius r, the signed distance from point $\mathbf{p}$ is:

$$d_{\text{sphere}}(\mathbf{p}) = \|\mathbf{p} - \mathbf{c}\| - r \tag{42}$$

where $d < 0$ indicates the point is inside the obstacle.

7.2 Box Obstacles

For an axis-aligned box centered at $\mathbf{c}$ with half-dimensions $\mathbf{h} = [h_x, h_y, h_z]^T$:

$$d_{\text{box}}(\mathbf{p}) = \begin{cases} \|\max(\mathbf{d} - \mathbf{h}, \mathbf{0})\| & \text{if outside} \\ -\min(h_i - d_i) & \text{if inside} \end{cases} \tag{43}$$

where $\mathbf{d} = |\mathbf{p} - \mathbf{c}|$ (element-wise absolute value).

7.3 Safety Margin

A collision is detected when:

$$d(\mathbf{p}) < d_{\text{safety}} \tag{44}$$

where $d_{\text{safety}} = 0.5$ m by default.

8 RRT Path Planning

8.1 Algorithm Overview

The Rapidly-exploring Random Tree (RRT) algorithm builds a tree of feasible paths from the start position toward the goal by iteratively sampling random configurations and extending the nearest tree node toward them.

Algorithm 1 RRT Path Planning

```
1: Initialize tree $\mathcal{T}$ with $\mathbf{x}_{\text{start}}$
2: for $i = 1$ to $N_{\max}$ do
3:     if rand() $< p_{\text{goal}}$ then
4:         $\mathbf{x}_{\text{rand}} \leftarrow \mathbf{x}_{\text{goal}}$        ▷ Goal bias
5:     else
6:         $\mathbf{x}_{\text{rand}} \leftarrow$ SampleRandom($\mathcal{W}$)
7:     end if
8:     $\mathbf{x}_{\text{near}} \leftarrow$ NearestNode($\mathcal{T}, \mathbf{x}_{\text{rand}}$)
9:     $\mathbf{x}_{\text{new}} \leftarrow$ Extend($\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{rand}}, \Delta$)
10:    if CollisionFree($\mathbf{x}_{\text{near}}, \mathbf{x}_{\text{new}}$) then
11:        Add $\mathbf{x}_{\text{new}}$ to $\mathcal{T}$ with parent $\mathbf{x}_{\text{near}}$
12:        if $\|\mathbf{x}_{\text{new}} - \mathbf{x}_{\text{goal}}\| < \epsilon_{\text{goal}}$ then
13:            return ExtractPath($\mathcal{T}, \mathbf{x}_{\text{new}}$)
14:        end if
15:    end if
16: end for
17: return Failure
```

8.2 Extension Function

The extend function moves from $\mathbf{x}_{\text{near}}$ toward $\mathbf{x}_{\text{rand}}$ by step size Δ:

$$\mathbf{x}_{\text{new}} = \mathbf{x}_{\text{near}} + \min\left(\Delta, \|\mathbf{x}_{\text{rand}} - \mathbf{x}_{\text{near}}\|\right) \cdot \frac{\mathbf{x}_{\text{rand}} - \mathbf{x}_{\text{near}}}{\|\mathbf{x}_{\text{rand}} - \mathbf{x}_{\text{near}}\|} \tag{45}$$

8.3 Path Collision Check

For a path segment from $\mathbf{p}_1$ to $\mathbf{p}_2$, we sample points along the line:

$$\mathbf{p}(\alpha) = \mathbf{p}_1 + \alpha(\mathbf{p}_2 - \mathbf{p}_1), \quad \alpha \in [0,1] \tag{46}$$

with sampling interval $\Delta s = 0.1$ m, and check each point for collision.

8.4 RRT Parameters

Table 2: RRT Planner Parameters

Parameter	**Symbol**	**Default Value**
Maximum iterations	N_{max}	5000
Step size	Δ	0.5 m
Goal bias probability	p_{goal}	0.15
Goal tolerance	ϵ_{goal}	0.5 m
Safety margin	d_{safety}	0.3 m

8.5 Path Simplification

After finding a path, we simplify it by removing unnecessary waypoints while maintaining collision-free connectivity:

Algorithm 2 Path Simplification

```
1: $\mathcal{P}_{\text{simple}} \leftarrow \{\mathbf{p}_1\}$
2: $i \leftarrow 1$
3: while $i < |\mathcal{P}|$ do
4:     $j \leftarrow |\mathcal{P}|$
5:     while $j > i + 1$ do
6:         if CollisionFree($\mathbf{p}_i, \mathbf{p}_j$) then
7:             break
8:         end if
9:         $j \leftarrow j - 1$
10:    end while
11:    Add $\mathbf{p}_j$ to $\mathcal{P}_{\text{simple}}$
12:    $i \leftarrow j$
13: end while
```

9 Minimum-Snap Trajectory Generation

9.1 Differential Flatness Requirements

For differential flatness-based control, the trajectory must provide derivatives up to snap:

- Position: $\mathbf{x}(t)$

- Velocity: $\dot{\mathbf{x}}(t)$
- Acceleration: $\ddot{\mathbf{x}}(t)$
- Jerk: $\dddot{\mathbf{x}}(t)$
- Snap: $\ddddot{\mathbf{x}}(t)$

9.2 Polynomial Representation

We use 7th-order polynomials for each segment, providing 8 degrees of freedom to satisfy boundary conditions. For normalized time $\tau \in [0, 1]$ within segment k:

$$p_k(\tau) = \sum_{i=0}^{7} c_{k,i}\tau^i = c_0 + c_1\tau + c_2\tau^2 + c_3\tau^3 + c_4\tau^4 + c_5\tau^5 + c_6\tau^6 + c_7\tau^7 \tag{47}$$

9.3 Time Normalization

For a segment with duration T, the relationship between real time t and normalized time τ is:

$$\tau = \frac{t - t_k}{T_k}, \quad t \in [t_k, t_k + T_k] \tag{48}$$

Derivatives transform as:

$$\dot{p}(t) = \frac{1}{T}\frac{dp}{d\tau} \tag{49}$$

$$\ddot{p}(t) = \frac{1}{T^2}\frac{d^2p}{d\tau^2} \tag{50}$$

$$\dddot{p}(t) = \frac{1}{T^3}\frac{d^3p}{d\tau^3} \tag{51}$$

$$\ddddot{p}(t) = \frac{1}{T^4}\frac{d^4p}{d\tau^4} \tag{52}$$

9.4 Boundary Conditions

For each segment, we enforce 8 boundary conditions (4 at each endpoint):

At $\tau = 0$:

$$p(0) = c_0 = x_0 \tag{53}$$

$$p'(0) = c_1 = v_0 \cdot T \tag{54}$$

$$p''(0) = 2c_2 = a_0 \cdot T^2 \tag{55}$$

$$p'''(0) = 6c_3 = j_0 \cdot T^3 \tag{56}$$

At $\tau = 1$:

$$p(1) = \sum_{i=0}^{7} c_i = x_1 \tag{57}$$

$$p'(1) = \sum_{i=1}^{7} i \cdot c_i = v_1 \cdot T \tag{58}$$

$$p''(1) = \sum_{i=2}^{7} i(i-1) \cdot c_i = a_1 \cdot T^2 \tag{59}$$

$$p'''(1) = \sum_{i=3}^{7} i(i-1)(i-2) \cdot c_i = j_1 \cdot T^3 \tag{60}$$

9.5 Matrix Formulation

The boundary conditions form a linear system $\mathbf{A}\mathbf{c} = \mathbf{b}$:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 0 & 2 & 6 & 12 & 20 & 30 & 42 \\ 0 & 0 & 0 & 6 & 24 & 60 & 120 & 210 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \\ c_7 \end{bmatrix} = \begin{bmatrix} x_0 \\ v_0 T \\ a_0 T^2 \\ j_0 T^3 \\ x_1 \\ v_1 T \\ a_1 T^2 \\ j_1 T^3 \end{bmatrix} \tag{61}$$

9.6 Segment Time Allocation

Segment times are allocated proportionally to segment distances with velocity constraints:

$$T_k = \max\left(\frac{d_k}{\sum_i d_i} \cdot T_{\text{total}}, \frac{d_k}{v_{\max}} \right) \tag{62}$$

where $d_k = \|\mathbf{x}_{k+1} - \mathbf{x}_k\|$ is the segment distance.

After applying constraints, times are renormalized:

$$T_k \leftarrow T_k \cdot \frac{T_{\text{total}}}{\sum_i T_i} \tag{63}$$

9.7 Continuity Between Segments

For interior waypoints, we ensure C^3 continuity by matching:

$$\mathbf{x}_k^- = \mathbf{x}_k^+ \qquad \text{(position)} \tag{64}$$

$$\dot{\mathbf{x}}_k^- = \dot{\mathbf{x}}_k^+ \qquad \text{(velocity)} \tag{65}$$

$$\ddot{\mathbf{x}}_k^- = \ddot{\mathbf{x}}_k^+ \qquad \text{(acceleration)} \tag{66}$$

$$\dddot{\mathbf{x}}_k^- = \dddot{\mathbf{x}}_k^+ \qquad \text{(jerk)} \tag{67}$$

10 Dynamic Replanning

10.1 Pop-up Threat Detection

The obstacle manager maintains a list of threats with activation times:

$$\text{Obstacle } i \text{ active} \Leftrightarrow t \geq t_{\text{activate},i} \tag{68}$$

10.2 Replanning Trigger

Replanning is triggered when the remaining path collides with newly activated obstacles:

Algorithm 3 Replan Check

```
1: for each segment in remaining path do
2:     for each sample point $\mathbf{p}$ along segment do
3:         if $d(\mathbf{p}, \mathcal{O}_{\text{active}}) < d_{\text{safety}}$ then
4:             return NeedsReplan
5:         end if
6:     end for
7: end for
8: return PathClear
```

10.3 Smooth Transition During Replanning

To avoid discontinuities when replanning, we preserve the current kinematic state as initial conditions for the new trajectory:

$$\begin{aligned} \mathbf{x}_0^{\text{new}} &= \mathbf{x}(t_{\text{replan}}) \\ \mathbf{v}_0^{\text{new}} &= \mathbf{v}(t_{\text{replan}}) \\ \mathbf{a}_0^{\text{new}} &= \mathbf{a}(t_{\text{replan}}) \end{aligned} \tag{69}$$

This ensures C^2 continuity at the replan point, preventing the drone from stopping before following the new path.

10.4 Replanning Algorithm

Algorithm 4 Dynamic Replanning

```
Require: Current state $(\mathbf{x}, \mathbf{v}, \mathbf{a})$, current time $t$, active obstacles $\mathcal{O}$
1: $\mathbf{x}_{\text{start}} \leftarrow \mathbf{x}$
2: $\mathcal{P} \leftarrow \text{RRT}(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}, \mathcal{O})$
3: if RRT succeeds then
4:     $\mathcal{P}_{\text{simple}} \leftarrow \text{SimplifyPath}(\mathcal{P})$
5:     $T_{\text{remain}} \leftarrow \max(T_{\text{total}} - t, T_{\text{min}})$
6:     trajectory $\leftarrow \text{MinSnapTraj}(\mathcal{P}_{\text{simple}}, T_{\text{remain}}, \mathbf{v}, \mathbf{a})$
7:     Shift time base: $t_{\text{traj}} \leftarrow t_{\text{traj}} + t$
8:     return Success
9: else
10:    return Failure (continue on original path)
11: end if
```

11 Simulation Parameters

11.1 Physical Parameters

Table 3: Quadrotor Physical Parameters

Parameter	**Symbol**	**Value**	**Unit**
Mass	m	1.0	kg
Inertia (x,y,z)	J	diag([0.01, 0.01, 0.02])	kg·m^2
Arm length (x)	l_x	0.2	m
Arm length (y)	l_y	0.2	m
Thrust coefficient	k_T	1.0×10^{-5}	N/(rad/s)^2
Moment coefficient	k_M	2.0×10^{-6}	Nm/(rad/s)^2
Motor time constant	τ_m	0.02	s
Gravitational acceleration	g	9.81	m/s^2

11.2 Control Gains

Table 4: Control Gains (from Tal & Karaman 2021, Table II)

Gain	**Symbol**	**Value**
Position	K_x	diag([18, 18, 13.5])
Velocity	K_v	diag([7.8, 7.8, 5.9])
Acceleration	K_a	diag([0.5, 0.5, 0.3])
Attitude	K_ξ	diag([175, 175, 82])
Angular rate	K_ω	0.8· diag([19, 19.5, 19.2])

11.3 Simulation Settings

Table 5: Simulation Settings

Parameter	**Value**	**Description**
dt	0.002 s	Simulation timestep (500 Hz)
ω_c	$2\pi \cdot 30$ rad/s	Filter cutoff frequency
τ_f	0.0053 s	Filter time constant
ω_{min}	0 rad/s	Minimum motor speed
ω_{max}	$2.5 \cdot \omega_{\text{hover}}$	Maximum motor speed

12 Results and Validation

12.1 Mission 1: Point-to-Point Navigation

- Start: $[0, 0, 0]$ m
- Goal: $[10, 5, -3]$ m
- Duration: Automatically calculated based on distance
- Result: Drone reaches goal within 0.5 m tolerance

12.2 Mission 2: Static Obstacle Avoidance

Three spherical obstacles are placed along the direct path:

- Obstacle 1: Center $[5, 2, -2]$ m, radius 1.5 m
- Obstacle 2: Center $[8, 4, -2.5]$ m, radius 1.2 m
- Obstacle 3: Center $[10, 5, -1.5]$ m, radius 0.8 m

The RRT planner finds a collision-free path, and the drone maintains positive clearance from all obstacles throughout the flight.

12.3 Mission 3: Pop-up Threat Avoidance

- Initial path: Direct line from $[0, 0, -2]$ to $[15, 0, -2]$ m
- Pop-up threat: Appears at $t = 4$ s at $[8, 0, -2]$ m with radius 2.0 m
- Replanning: Triggered immediately upon threat detection
- Smooth transition: Velocity and acceleration continuity maintained

13 Conclusion

This report documents a comprehensive quadrotor simulation that combines:

1. **6-DOF Dynamics:** Full translational and rotational dynamics with motor models
2. **Differential Flatness:** Algebraic computation of feedforward terms from trajectory derivatives
3. **INDI Control:** Incremental control formulation reducing model dependency
4. **Cascaded Architecture:** Outer position loop with inner attitude loop
5. **RRT Path Planning:** Efficient collision-free path generation
6. **Minimum-Snap Trajectories:** Smooth, differentiable trajectories for differential flatness
7. **Dynamic Replanning:** Real-time trajectory updates with kinematic continuity

The implementation successfully demonstrates aggressive trajectory tracking and autonomous navigation capabilities for quadrotor systems.

References

References

[1] Tal, E., & Karaman, S. (2021). *Accurate Tracking of Aggressive Quadrotor Trajectories Using Incremental Nonlinear Dynamic Inversion and Differential Flatness.* IEEE Transactions on Control Systems Technology.

[2] Mellinger, D., & Kumar, V. (2011). *Minimum Snap Trajectory Generation and Control for Quadrotors.* IEEE International Conference on Robotics and Automation (ICRA).

[3] LaValle, S. M. (1998). *Rapidly-Exploring Random Trees: A New Tool for Path Planning.* Technical Report, Computer Science Department, Iowa State University.

[4] Smeur, E. J. J., Chu, Q., & de Croon, G. C. H. E. (2016). *Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles.* Journal of Guidance, Control, and Dynamics.

A MATLAB Code Usage

A.1 Running the Original Demo

```
% In MATLAB, from test_rig directory:
demo_indi_6dof_ned   % Original trajectory tracking
```

A.2 Running the Mission Demos

```
% In MATLAB, from test_rig directory:
demo_mission_1   % Point-to-point navigation
demo_mission_2   % Static obstacle avoidance
demo_mission_3   % Pop-up threat avoidance
```

Each demo saves an MP4 video of the flight animation.

A.3 Using the Motion Planner API

```
% Add motion planner to path
addpath('motion_planner');

% Create waypoint manager
wm = waypoint_manager();

% Define mission
start_pos = [0; 0; -2];
goal_pos = [10; 5; -3];
obstacles = {struct('type', 'sphere', 'center', [5;2;-2], 'radius',
    1.5)};
bounds = struct('min', [-5;-5;-10], 'max', [15;10;0]);

% Plan and generate trajectory
wm.set_mission(start_pos, goal_pos, obstacles, bounds);
[success, waypoints] = wm.plan_path();
traj = wm.generate_trajectory(12.0);

% Get reference at time t
ref = wm.get_reference(t);
```

A.4 Replanning with Velocity Continuity

```
% During simulation, when replanning is needed:
current_vel = state.v;
current_acc = filters.a_f;

[success, new_traj] = wm.replan(state.x, t, obstacles, current_vel,
    current_acc);
```

B Block Diagram

The complete control system can be summarized as:

```
+------------------+     +------------------+     +------------------+
| Trajectory       | --> | Position Control | --> | Attitude Command |
| x, v, a, j, s    |     | Eq. 17           |     | Eq. 22-26        |
+------------------+     +------------------+     +------------------+
                                                          |
                                                          v
+------------------+     +------------------+     +------------------+
| Motor Commands   | <-- | Attitude Control | <-- | Flatness         |
| omega_1..4       |     | Eq. PD           |     | Eq. 14-15        |
+------------------+     +------------------+     +------------------+
                                                          |
                                                          v
                         +------------------+     +------------------+
                         | 6-DOF Dynamics   | <-- | Motor Dynamics   |
                         | Eq. 1-5          |     | 1st Order        |
                         +------------------+     +------------------+
```