ISSN 2959-6157

Design and application of UAV single target tracking system based on YOLOv5

Hongbo Fan

School of China West Normal University, Nanchong, Sichuan, 637009, China

Abstract:

In aerial reconnaissance, pursuit, and post-disaster search and rescue, rotorcraft UAVs are valued for their agility. A key challenge is target tracking in complex environments with no prior information, dense obstacles, GNSS signal loss, and occluded targets. This paper introduces a YOLOv5-based UAV tracking system to address detection and path planning. The quadrotor UAV model ensures stable flight through mathematical modeling and simulation. YOLOv5 enhances detection performance with real-time tracking. Feedback control enables autonomous obstacle avoidance and target tracking. Results show adaptability and robustness, supporting reconnaissance and rescue missions effectively.

Keywords: Quadrotor UAV, YOLOv5, Target Tracking, Feedback Control, Path Planning

1. Introduction

In recent years, UAV technology has advanced rapidly, particularly with multirotor UAVs in commercial applications like power line inspection, agriculture, search and rescue, and cargo delivery. Traditional research focused on high-altitude applications, but as UAVs expand into low-altitude and complex scenarios, the emphasis has shifted towards enhancing intelligence and autonomy. Advances have improved UAVs' ability to operate in complex environments, extending applications to aerial photography, surveillance, and search and rescue. Fixed-wing UAVs are unsuitable for target tracking due to their speed and size, whereas quadrotor UAVs excel in flexibility and maneuverability [1,2]. Autonomous UAV navigation and trajectory planning in complex environments remain challenging, requiring solutions for environmental perception, modeling, and efficient trajectory planning. This paper investigates algorithms for predicting moving target trajectories and rapid tracking path generation, constructing a UAV tracking system for unknown targets in complex environments, valuable for defense, broadcasting, security patrols, and search and rescue. Therefore, this research is of significant practical importance.

2. Literature review

Trajectory planning for quadrotor UAVs begins with determining trajectory representation and solving methods. Mellinger and Kumar [3] used fixed-duration splines, optimizing smoothness with the squared norm of the fourth-order time derivative, but lacked effective duration optimization for spline segments. Bry et al. [4] incorporated UAV dynamics, proposing a closed-form solution for the quadratic programming problem with high-order continuity and waypoint constraints, using RRT* for path generation. However, it struggles in dense obstacles and cannot explicitly include flight speed constraints. Deits et al. [5] and Gao et al. [6] used convex polyhedra to describe feasible regions, ensuring safety through linear constraints on Bézier curve control points. Deits' method suffered from low computational efficiency, while Gao's approach alternately optimized the trajectory's geometric and temporal characteristics, improving computational efficiency and safety.

3. Methods

3.1 UAV Attitude Modeling

3.1.1 *Establishing the Coordinate System*

The "North-East-Down" geographic coordinate system is chosen as the navigation frame (n-frame), and the "Forward-Right-Down" coordinate system is used as the body frame (b-frame). The n-frame is orthogonal relative to the Earth's horizontal plane, with X_n, Y_n and Z_n axes pointing north, east, and down. The b-frame is fixed to the UAV with X, Y and Z axes pointing forward, right, and down. The transformation matrix from the n-frame to the b-frame is represented by the direction cosine matrix:

$$C_n^b = \begin{pmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \cos\psi\sin\phi\sin\theta - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{pmatrix} \quad (1)$$

3.2 System Model Establishment

3.2.1 Attitude Estimation Based on Accelerometer

The accelerometer outputs the body acceleration. The gravitational acceleration in the navigation frame is $g_n = (0,0,g)^T$, and the output of the accelerometer in the body frame is $a_b = (a_x, a_y, a_z)^T$. The transformation relationship is:

$$\begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} = C_n^b \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = \begin{pmatrix} -g\sin\theta \\ g\cos\theta\sin\phi \\ g\cos\theta\cos\phi \end{pmatrix} \quad (2)$$

From the accelerations obtained by the above formula, the pitch and roll angles can be derived as:

$$\begin{cases} \theta = -\arcsin\left(\dfrac{a_x}{g}\right) \\ \phi = \arctan\left(\dfrac{a_y}{a_z}\right) \end{cases} \quad (3)$$

3.2.2 Attitude Estimation Based on Magnetometer

The magnetometer outputs the geomagnetic field vector. On the Earth's surface, the geomagnetic field typically points toward magnetic north, consisting of a northward component and a vertical component, but no eastward component. Therefore, the magnetic field vector in the geomagnetic coordinate system can be expressed as (*N*,0,*D*). When the geomagnetic coordinate system coincides with the navigation frame *n*, $m_n = (M, 0, M_0)$ represents the magnetometer output in the navigation frame. Assuming the magnetometer output in the body frame *b* is $m_n = (m_x, m_y, m_z)^T$, it can be transformed using the transformation matrix C_n^b as follows:

$$m_b = C_n^b m_n \quad (4)$$

After simplification, this becomes:

$$\begin{pmatrix} m_x \\ m_y \\ m_z \end{pmatrix} = \begin{pmatrix} M\cos\theta\cos\psi_m - M_0\sin\theta \\ M\cos\theta\sin\psi_m \\ M\sin\theta + M_0\cos\theta \end{pmatrix} \quad (5)$$

Thus, the horizontal component of the geomagnetic vector is:

$$\begin{pmatrix} M\cos\theta\cos\psi_m - M_0\sin\theta \\ M\cos\theta\sin\psi_m \end{pmatrix} \quad (6)$$

The yaw angle can then be calculated as:

$$\psi_m = \arctan\left(\frac{m_y}{m_x}\right) \quad (7)$$

Finally, the magnetic yaw angle is given by:

$$\psi_{\text{mag}} = \psi_m + \alpha \quad (8)$$

where α is the magnetic declination, correcting the angle difference between magnetic north and true north. This method uses geomagnetic information from the magnetometer, applying a mathematical model and coordinate transformation to accurately estimate the UAV's yaw angle.

3.2.3 Attitude Estimation Based on Gyroscope

The gyroscope outputs the angular acceleration of the sensitive carrier. The dynamic model update of gyroscope angular velocity based on quaternions can be expressed as:

$$\dot{q} = \frac{1}{2} q \otimes \omega^b \quad (9)$$

where $\omega^b = (\omega_x, \omega_y, \omega_z)$ represents the angular velocity measured by the gyroscope, and $q = (q_0, q_1, q_2, q_3)^T$ represents the attitude quaternion. The matrix form of equation (10) can be expressed as:

$$\begin{pmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & \omega_z & -\omega_y \\ \omega_y & -\omega_z & 0 & \omega_x \\ \omega_z & \omega_y & -\omega_x & 0 \end{pmatrix} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{pmatrix} \quad (10)$$

Using the Picard iterative method to solve the quaternion differential equation (9), its discrete form is given by:

$$q(t_k) = \left[\cos\left(\frac{\Delta\varphi}{2}\right) + \sin\left(\frac{\Delta\varphi}{2}\right)\frac{\Delta\Omega}{\Delta\varphi}\right] q(t_{k-1}) \quad (11)$$

where $\triangle\varphi = \sqrt{\alpha_x^2 + \alpha_y^2 + \alpha_z^2}$ is the angular increment during the sampling interval $[tk-1, tk]$, $\alpha_i = \int_{t_{k-1}}^{t_k} \omega_i dt$, $i = x, y, z$, and $\triangle\Omega = (\alpha_x \quad \alpha_y \quad \alpha_z)^T$.

The transformation matrix from the body frame to the navigation frame, represented using quaternions, is given by:

$$C_n^b = \begin{pmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2(q_1q_2+q_0q_3) & 2(q_1q_3-q_0q_2) \\ 2(q_1q_2-q_0q_3) & q_0^2-q_1^2+q_2^2-q_3^2 & 2(q_2q_3+q_0q_1) \\ 2(q_1q_3+q_0q_2) & 2(q_2q_3-q_0q_1) & q_0^2-q_1^2-q_2^2+q_3^2 \end{pmatrix} \quad (12)$$

Thus, the UAV attitude angles can be expressed using quaternions as:

$$\begin{cases} \phi = \arctan\left(\dfrac{2(q_0q_1+q_2q_3)}{q_0^2-q_1^2-q_2^2+q_3^2}\right) \\ \theta = \arcsin\left(2(q_0q_2-q_1q_3)\right) \\ \psi = \arctan\left(\dfrac{2(q_0q_3+q_1q_2)}{q_0^2+q_1^2-q_2^2-q_3^2}\right) \end{cases} \quad (13)$$

3.3 Motion Target Visual Detection Method

Considering the requirements for target tracking in complex environments and factors like UAV cost and payload capacity, this paper adopts a stereo vision system for detecting and locating moving targets. A neural network-based model, trained using the YOLOv5 algorithm, detects moving targets. The detection results are converted into world coordinates to determine the target's position.

3.3.1 Target Detection Algorithm Based on YOLOv5

To address target detection in complex environments, this paper employs a detection scheme based on convolutional neural networks (CNNs), which have significantly advanced image processing since 2012. Traditional methods, reliant on manually designed features, lack robustness in dynamic scenes. Learning-based detection methods are categorized into two types: two-stage detection, offering high precision through coarse detection and refinement, and single-stage detection, which is faster by directly classifying each Region of Interest (ROI) as background or target. YOLO (You Only Look Once) series is preferred for its efficiency in multi-scale and multi-target tasks [8]. This paper utilizes YOLOv5 [9] for detecting moving targets, integrating pose detection post YOLO detection to output final results.

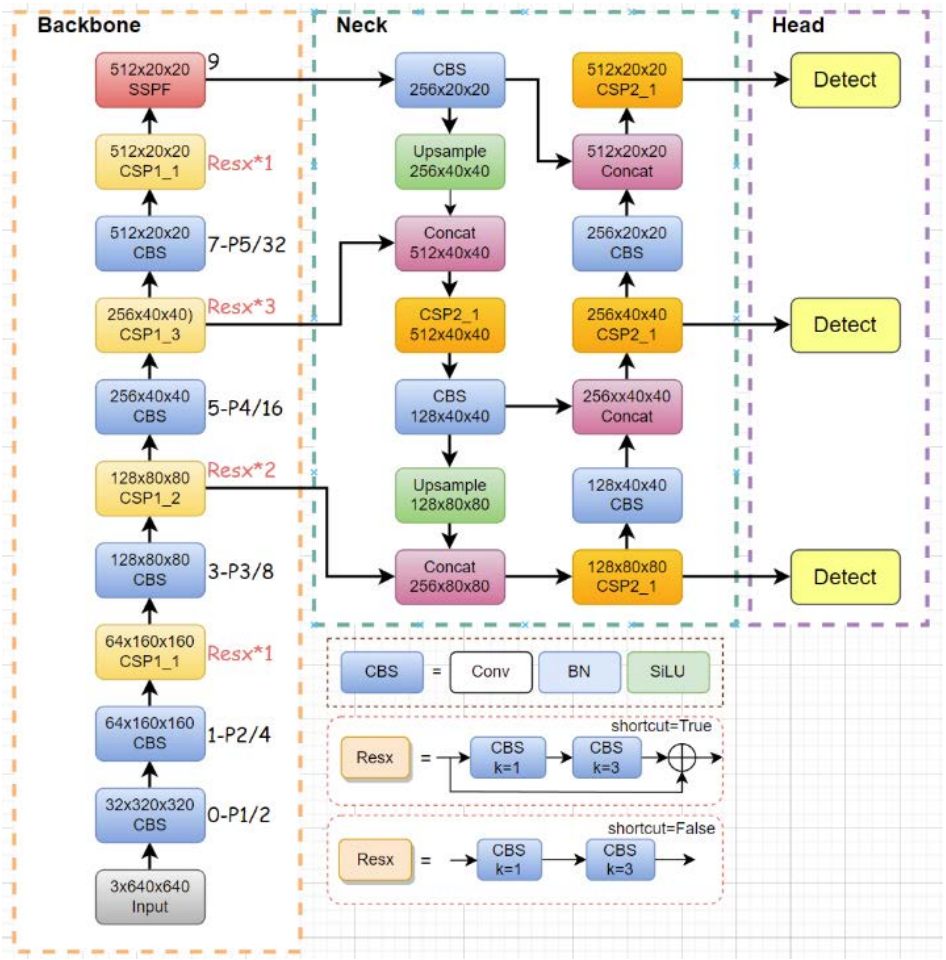

Figure 1 YOLOv5 system diagram.

The YOLO algorithm divides an image into grid cells, detecting targets via the cell containing the target's center point. It uses regression to determine the Bounding-Box coordinates and confidence. YOLO's network has two main parts: the Backbone for feature extraction, pre-trained on image datasets, and the Head for final detection. YOLOv5's structure includes Input, Backbone, Neck, and Output. Input employs Mosaic augmentation, enriching the dataset by scaling and stitching images, achieving good results with small Mini-batches on a single GPU. The Backbone retains image information using Focus, CSP, and SPP structures. CSP enhances learning and reduces costs; SPP increases the receptive field. The Neck generates feature pyramids for multi-scale object handling, preserving positional and categorical information.

The output generates anchor boxes with confidence scores, target classes, and bounding box coordinates. YOLOv5's loss function has three parts: box regression (mean squared error), object confidence (presence), and classification (cross-entropy) The total $loss$ function is calculated as:

$$Loss = \omega_1 box\_loss + \omega_2 obj\_loss + \omega_3 cls\_loss \quad (13)$$

$$box\_loss = 1 - IoU + \frac{\rho^2\left(b, b_{gt}\right)}{c^2} + \frac{\rho^2\left(b, b_{gt}\right)}{C_\omega^2} + \frac{\rho^2\left(h, h_{gt}\right)}{C_h^2} \quad (17)$$

where C_ω and C_h are the width and height of the smallest enclosing box, and ω, ω_{gt}, h, h_{gt} are the widths and heights of predicted and ground truth boxes. This enhancement addresses the limitations of CIoU by incorporating width and height differences directly into the loss calculation.

where ω_1, ω_2, and ω_3 are weights for each component. The box_loss uses CIoU (Complete Intersection over Union), calculated as:

$$box\_loss = 1 - IoU + \frac{\rho^2\left(b, b_{gt}\right)}{c^2} + \alpha v \quad (14)$$

with IoU representing the intersection over union, b and b_{gt} as the center coordinates of predicted and ground truth boxes, $d = \rho^2\left(b, b_{gt}\right)$ as the Euclidean distance between centers, c as the diagonal length of the smallest enclosing box, α as a positive weight, and v measuring aspect ratio similarity.

$$\alpha = \frac{v}{\left(1 - IoU\right) + v} \quad (15)$$

$$v = \frac{4}{\pi^2}\left(\arctan\frac{\omega_{gt}}{h_{gt}} - \arctan\frac{\omega}{h}\right) \quad (16)$$

To improve IoU, the width and height ratios are decomposed and separately discussed, resulting in the updated loss:

3.3.2 Implementing YOLOv5 Model in MATLAB

Here is a MATLAB script for implementing a YOLOv5 model for object detection, accompanied by comprehensive English annotations:

```
% Implementing YOLOv5 Object Detection in MATLAB
% This script assumes you have a trained YOLOv5 model and a preprocessed input image.

% Load the trained YOLOv5 model
model = load('yolov5_model.mat'); % Replace with the path to your YOLOv5 model

% Load the input image
inputImage = imread('input_image.jpg'); % Replace with the path to your input image

% Preprocess the image: resize and normalize
inputSize = [640, 640]; % Define the input size for YOLOv5 (height, width)
imageResized = imresize(inputImage, inputSize); % Resize the image to the input size
imageNormalized = double(imageResized) / 255; % Normalize pixel values to the range [0, 1]

% Convert the image to a deep learning array
inputDlarray = dlarray(imageNormalized, 'SSC'); % Convert to dlarray for deep learning

% Run the YOLOv5 model on the input image
% The model should return bounding boxes, scores, and class labels
[boundingBoxes, scores, classLabels] = predict(model, inputDlarray);

% Visualize the detection results
figure;
imshow(inputImage); % Display the input image
hold on;
for i = 1:size(boundingBoxes, 1)
    % Extract bounding box coordinates
    bbox = boundingBoxes(i, :);
    score = scores(i);
    label = classLabels(i);

    % Draw the bounding box
    rectangle('Position', bbox, 'EdgeColor', 'r', 'LineWidth', 2); % Draw a red rectangle

    % Display the label and score
    labelStr = sprintf('%s: %.2f', label, score); % Create a label string with class and score
    text(bbox(1), bbox(2), labelStr, 'Color', 'yellow', 'FontSize', 12, 'FontWeight', 'bold'); % Display the label
end
hold off;

% Save the detection result
saveas(gcf, 'detection_result.jpg'); % Save the figure as an image

```

Figure 2 YOLOv5 Model in MATLAB

This MATLAB script uses a YOLOv5 model for object detection. It loads a pre-trained YOLOv5 model and processes an input image to match the model's requirements. The script then runs the model to predict bounding boxes, confidence scores, and class labels for detected objects. Results are visualized by overlaying bounding boxes and labels on the input image, which is saved as an image file, ensuring efficient object detection and visualization using MATLAB.

3.4 Feedback Control Path Planning

3.4.1 Applying Feedback Control to Path Planning

Feedback control measures a variable, compares it to a target, and adjusts inputs to minimize error, continuously maintaining the desired state (Figure 3).

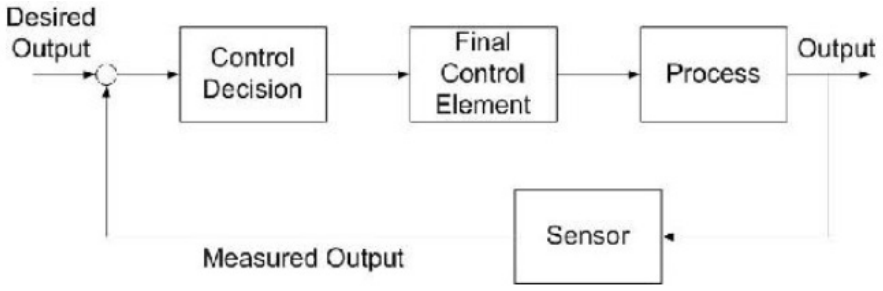

Figure 3 Feedback control loop.

In UAV path planning, feedback control ensures the UAV accurately follows a planned trajectory despite disturbanc-

es. The process involves:
(1) Measurement: Onboard sensors measure the UAV's current position, velocity, and state variables.
(2) Comparison: These values are compared to the desired trajectory setpoints.
(3) Error Calculation: The difference (error) between measured and desired values is computed.
(4) Control Decision: Based on the error, the control system adjusts UAV control inputs (motor speeds, control surfaces).
(5) Implementation: Control inputs are adjusted to minimize the error, maintaining the UAV on the desired path.

3.4.2 MATLAB Implementation for Path Planning Based on YOLOv5 Detection Results

This MATLAB script demonstrates feedback control for UAV path planning using YOLOv5 detection results. It involves generating a path from the detected target position and using proportional control to navigate the UAV.
MATLAB Script Framework:
(1) Load YOLOv5 Detection Results: Load bounding box coordinates and calculate the target's center.
(2) Generate Path: Create a linear path from the UAV's initial position to the target using linspace.
(3) Implement Feedback Control: Apply proportional control to adjust UAV position based on error.
(4) Visualize Results: Plot the desired path, UAV trajectory, and target position to visualize adherence using feedback control.
As shown in Figure4:

```
// Load YOLOv5 Detection Results
boundingBoxes = [100, 150, 50, 50]; // Example detection result
targetPosition = [boundingBoxes(1) + boundingBoxes(3)/2, boundingBoxes(2) + boundingBoxes(4)/2];

// Define initial UAV position and generate the path
uavPosition = [0, 0];
pathX = linspace(uavPosition(1), targetPosition(1), 100);
pathY = linspace(uavPosition(2), targetPosition(2), 100);

// Feedback control parameters
Kp = 0.1; % Proportional gain
dt = 0.1; % Time step

// Simulate the UAV following the path
for i = 1:length(pathX)
    // Calculate error and control inputs
    errorX = pathX(i) - uavPosition(1);
    errorY = pathY(i) - uavPosition(2);
    controlInputX = Kp * errorX;
    controlInputY = Kp * errorY;

    // Update UAV position
    uavPosition(1) = uavPosition(1) + controlInputX * dt;
    uavPosition(2) = uavPosition(2) + controlInputY * dt;

    // Store trajectory
    uavTrajectoryX(i) = uavPosition(1);
    uavTrajectoryY(i) = uavPosition(2);
end

// Plot the results
figure;
plot(pathX, pathY, 'b--', 'LineWidth', 2); hold on;
plot(uavTrajectoryX, uavTrajectoryY, 'r-', 'LineWidth', 2);
plot(targetPosition(1), targetPosition(2), 'go', 'MarkerSize', 10, 'MarkerFaceColor', 'g');
xlabel('X Position'); ylabel('Y Position');
title('UAV Path Planning Using Feedback Control');
legend('Desired Path', 'UAV Trajectory', 'Target Position');
grid on; hold off;

```

Figure 4

MATLAB Script for UAV Path Planning Using Feedback Control Based on YOLOv5 Detection Results

This script provides a streamlined framework for implementing feedback control-based path planning using YOLOv5 detection results in MATLAB. Adjust the control parameters and path generation logic as needed to accommodate specific UAV dynamics and operational requirements.

4. Results and Discussion

4.1 MATLAB Implementation of the Quadrotor UAV Feedback Control System

To build a comprehensive feedback control system for a quadrotor UAV using MATLAB, the implementation involves several key components as depicted in the provided diagram. Each component is crucial for achieving stable and accurate control of the UAV.

As shown in the Figure 5:

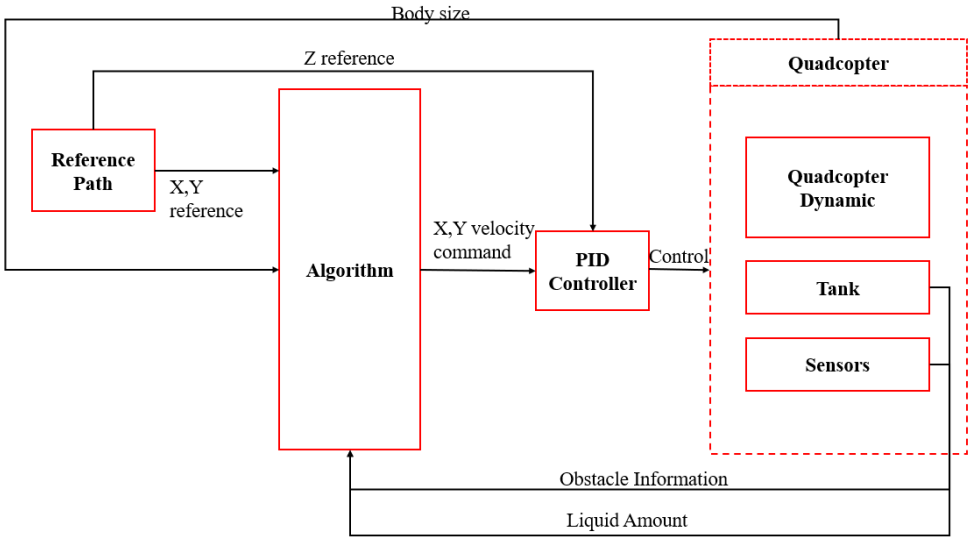

Figure 5 Operation drone control architecture.

(1) Reference Path Generation

The reference path, serving as the desired UAV trajectory, is generated using predefined waypoints or dynamically based on mission requirements. In MATLAB, this is achieved by interpolating between waypoints to form a smooth trajectory.

(2) Algorithmic Module

This module processes the reference path and generates velocity commands based on the UAV's current state and additional information, such as obstacle locations. It involves state estimation and control law computation.

(3) PID Controller

Responsible for minimizing the error between desired velocity commands and actual UAV velocity, the PID controller can be designed and tuned using MATLAB's Control System Toolbox.

(4) Quadcopter Dynamics

The dynamics of the quadcopter, including its response to control inputs, are accurately modeled using a state-space representation.

(5) Sensors and Feedback Loop

UAV sensors provide necessary feedback on the UAV's state, which is used to adjust control inputs. Sensor data is processed to estimate the UAV's position and velocity.

(6) Physical Model of Quadcopter

To implement the quadcopter's physical model in MATLAB, the following components are considered:

(1) Main Body Frame: Modeled using 6-DOF dynamic equations to simulate the entire UAV.

(2) Tank: Attached to the bottom of the quadcopter, modeled as a rigid body with variable mass and inertia.

(3) Sensors: Integrated with the UAV to gather environmental data and provide feedback for the control system.

4.2 *Experimental Results and Performance Analysis*

In this section, we present the experimental results and performance analysis of the quadrotor UAV's path tracking using a feedback control system. The UAV was tasked with following a predefined path while maintaining stability and accuracy. The figures provided illustrate the UAV's trajectory, Euler angles, and position errors over time.

4.2.1 Stability and Tracking Accuracy

The 3D plots (Figure 6, Figure 7, and Figure 8) show the UAV's path tracking over time. The UAV successfully

follows the predefined path, demonstrating high stability and precise tracking capabilities. The Euler angles, which represent the UAV's orientation, remain stable throughout the flight, indicating that the control system effectively maintains the desired attitude.

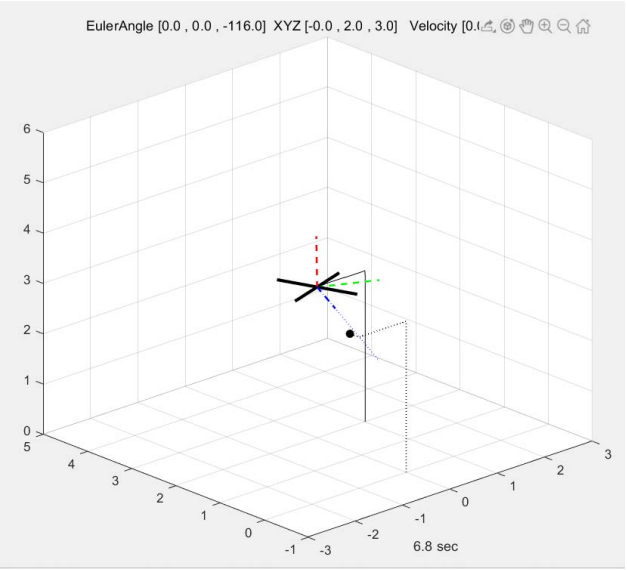

Figure 6 Initial phase of UAV trajectory.

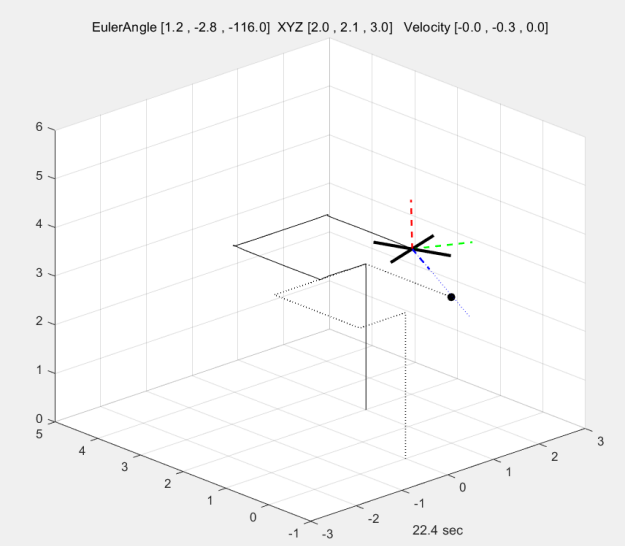

Figure 7 Mid-phase of UAV trajectory

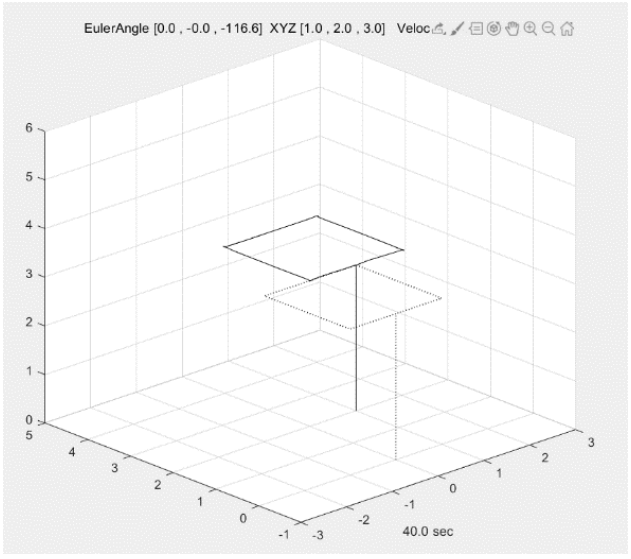

Figure 8 Final phase of UAV trajectory.

4.2.2 Error Analysis

Graphs of position errors in X, Y, and Z axes reveal the control system's accuracy. Errors remain within limits, demonstrating the PID controller's robustness. The quaternion distance graph confirms stability, showing minimal angular deviations during tracking.

4.2.2 Performance Metrics

1. Quaternion Distance: The quaternion distance graph indicates that the UAV maintains a stable orientation with minimal fluctuations. This metric is crucial for ensuring the UAV's precise maneuvering and overall stability.
Observed in Figures 9, 10, and 11.
2. Position Errors:
X Position Error: The error in the X position remains consistently low, indicating accurate lateral control.
Y Position Error: The Y position error shows slight variations but stays within a manageable range, reflecting effective longitudinal control.
Z Position Error: The Z position error is minimal, highlighting the UAV's ability to maintain the desired altitude accurately.
Observed in Figures 9, 10, and 11.

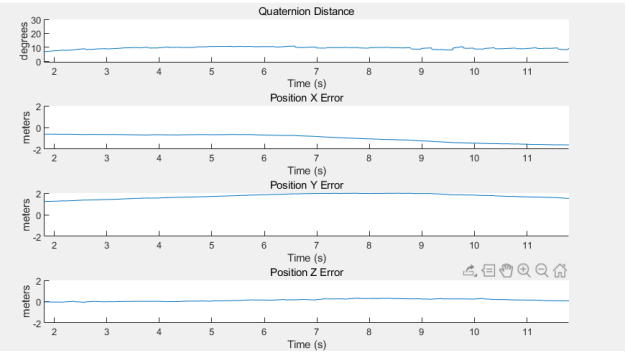

Figure 9 Position and orientation errors during initial phase.

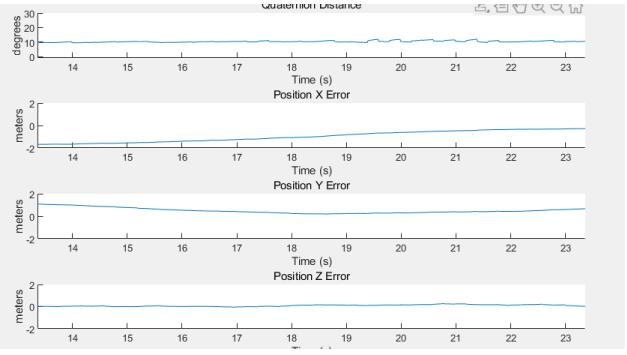

Figure 10 Position and orientation errors during mid-phase.

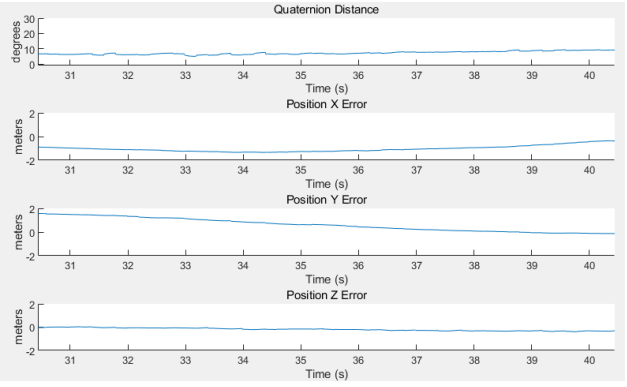

Figure 11 Position and orientation errors during mid-phase.

5. Conclusion

The experimental results confirm the feedback control system's effectiveness in ensuring stable and accurate path tracking for the quadrotor UAV. The UAV follows the predefined path with minimal deviations, and the control system maintains stability under various conditions. Low position errors and stable Euler angles demonstrate the PID controller's robustness and the system's capability for real-world scenarios. This analysis highlights the proposed strategy's potential for practical UAV applications, including surveillance and search and rescue. Future work will optimize control algorithms and extend capabilities for more complex environments.

References

[1] Patrona F, Mademlis I, Tefas A, et al. Computational UAV cinematography for intelligent shooting based on semantic visual analysis[C]. 2019 IEEE International Conference on Image Processing (ICIP), 2019: 4155-4159.

[2] Joubert N, Goldman D B, Berthouzoz F, et al. Towards a drone cinematographer: Guiding quadrotor cameras using visual composition principles[J]. arXiv preprint arXiv:1610.01691, 2016.

[3] Mellinger D, Kumar V. Minimum snap trajectory generation and control for quadrotors[C]. 2011 IEEE international conference on robotics and automation, 2011: 2520-2525.

[4] Bry A, Richter C, Bachrach A, et al. Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments[J]. The International Journal of Robotics Research, 2015, 34(7): 969-1002.

[5] Deits R, Tedrake R. Efficient mixed-integer planning for UAVs in cluttered environments[C]. 2015 IEEE international conference on robotics and automation (ICRA), 2015: 42-49.

[6] Gao F, Wang L, Zhou B, et al. Teach-repeat-replan: A complete and robust system for aggressive flight in complex environments[J]. IEEE Transactions on Robotics, 2020, 36(5): 1526-1545.

[7] Redmon J, Divvala S, Girshick R, et al. You only look once: Unified, real-time object detection[C]. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016: 779-788.

[8] Jocher G. You only look once version 5[CP/OL]. [2022-12-05], https://github.com/ultralytics/yolov5/tree/v6.0.