AERSP 497 Autonomy

Final Project

Name	Contribution	Contribution %	Honor Statement Sign
Jackson Fezell	Code & Report	33	JF
Nicholas Giampetro	Code & Report	34	NG
Ankit Gupta	Code & Report	33	AG

Introduction

In this project, we developed a control pipeline for a simulated quadcopter, using MATLAB. The primary objective is to enable the quadcopter to navigate a predetermined path with precision and stability. This project requires a comprehensive approach that includes trajectory planning, position and attitude control, motor control, and dynamics.

The MATLAB code provided to us includes:

- Lookup_waypoints.m
- Trajectory_planner.m
- Position_controller.m
- Attitude_planner.m
- Attitude_controller.m
- Motor_model.m
- Dynamics.m
- Plot_quadmotor_errors.m
- Main.m

The code conducts a comparison between the desired and actual states of motion, with the desired state being determined through MATLAB's ODE45 function. ODE45, utilizing the Runge-Kutta method, serves as an explicit solver that is especially effective for linear differential equations. Within the 'dynamics.m' function, specific equations of motion are defined and subsequently passed through ODE45 for processing.

To compute the desired state, the code relies on functions such as 'position_controller.m', 'attitude_planner.m', and 'attitude_controller.m'. These functions encompass the Equations of Motion pertinent to a quadcopter's operation, thereby enabling precise control and maneuverability.

Additionally, 'plot_quadrotor_errors.m' allows visualization of the quadrotor's performance. This function generates graphs to illustrate the differences between the desired and actual states across various parameters, including position, velocity, acceleration, angular velocity, and orientation. It also helps in highlighting and analyzing the errors between the desired and actual states of the quadcopter.

System Design

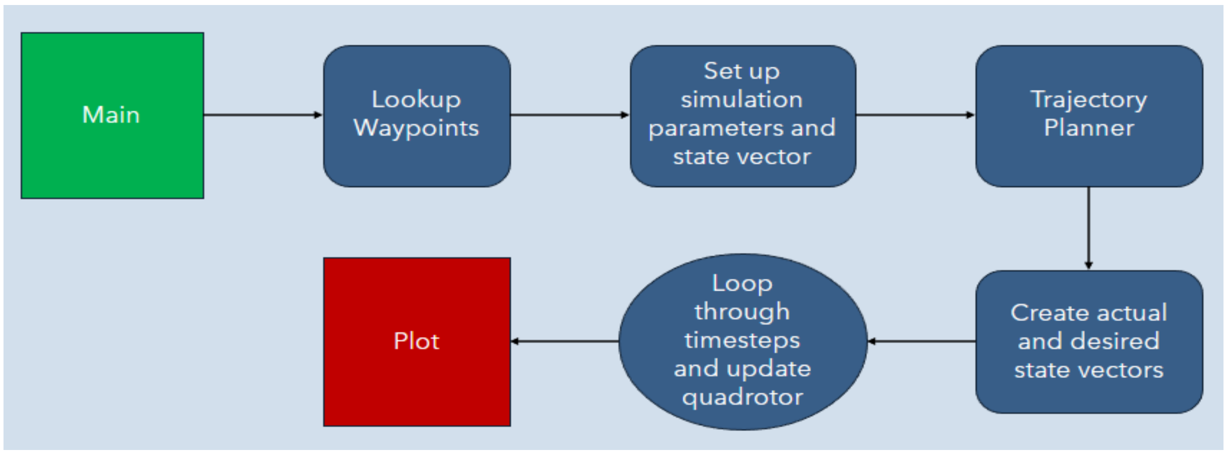
Main
Lookup Waypoints
Set up simulation parameters and state vector
Trajectory Planner
Create actual and desired state vectors
Loop through timesteps and update quadrotor
Plot

Figure 1. Overview of System Flow

Figure 1 presents a flowchart detailing the architecture of a MATLAB simulation program for a quadrotor. The simulation is initiated via the "Main" function, which governs the simulation's execution sequence. This central function proceeds to the "Lookup Waypoints" module to acquire predetermined sets of coordinates that the quadrotor is to navigate through. Subsequently, the "Main" function configures the necessary simulation parameters and state vectors before invoking the "Trajectory Planner" function. This function is responsible for determining a viable flight path for the quadrotor. Utilizing additional functions that manage position and attitude planning, the simulation program systematically generates a graphical representation that depicts the quadrotor's various positional and orientational states over a defined time frame.

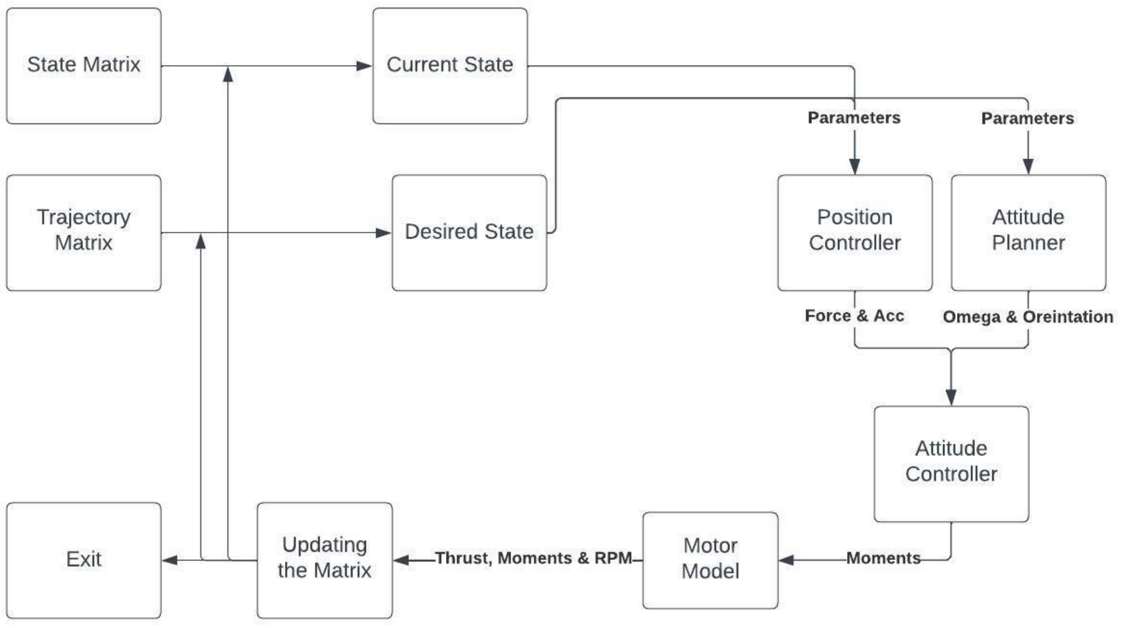

Figure 2. Control System Diagram

Figure 2 shows the control system for the MATLAB simulation program for a quadrotor. The "State Matrix" and "Trajectory Matrix" serve as inputs, representing the vehicle's current operational state and the desired trajectory, respectively. They feed into the "Current State" and "Desired State" modules, establishing the vehicle's actual condition and the targeted state it should achieve. Then based of the parameters, "Position Controller" and "Attitude Planner" determine the necessary adjustments to the vehicle's position and orientation. These control signals, which contains information like force, acceleration, orientation are then utilized by the "Attitude Controller" to generate moments for the "Motor Model," which simulates the responses of the vehicle's propulsion system which includes thrust moments and RPM, leading to changes in state. The feedback loop, through "Updating the Matrix", continuously monitors and adjusts the vehicle's state towards the desired state. Lastly, the "Exit" signifies the end of the feedback loop.

Hover Performance

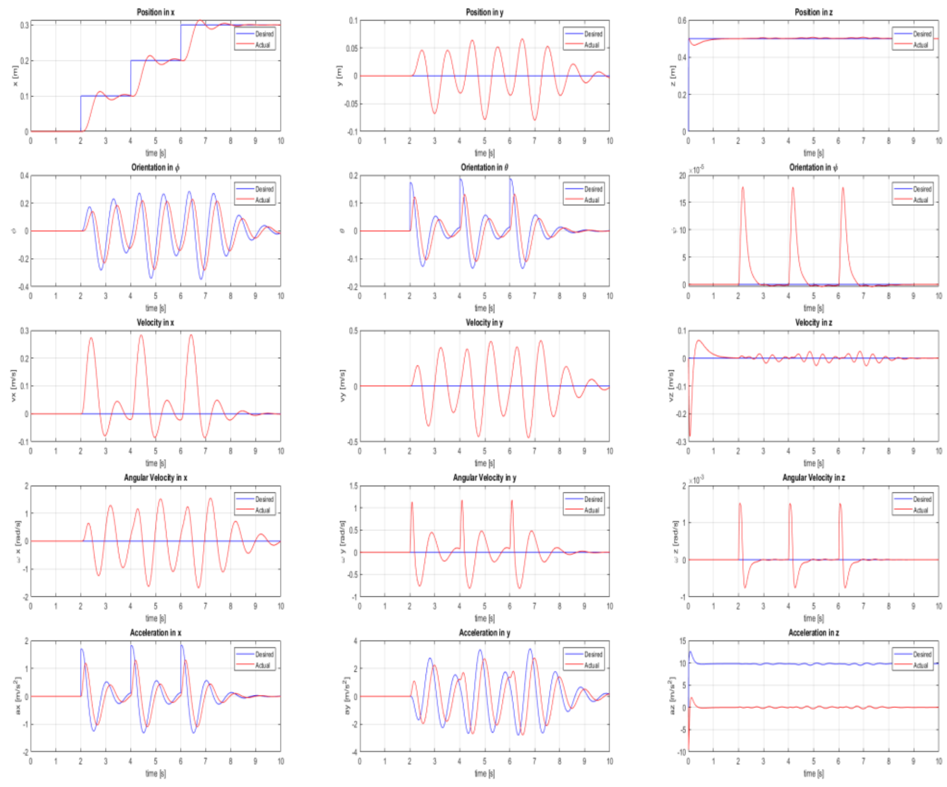

Figure 3. Actual State v/s Desired State (Set 1)

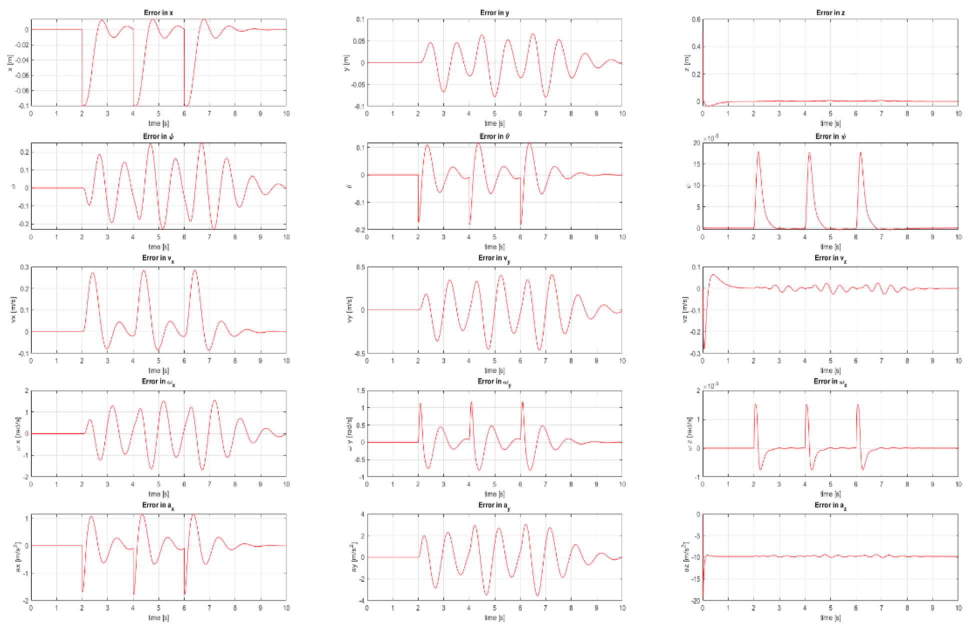

Figure 4. Plot of Error (Set 1)

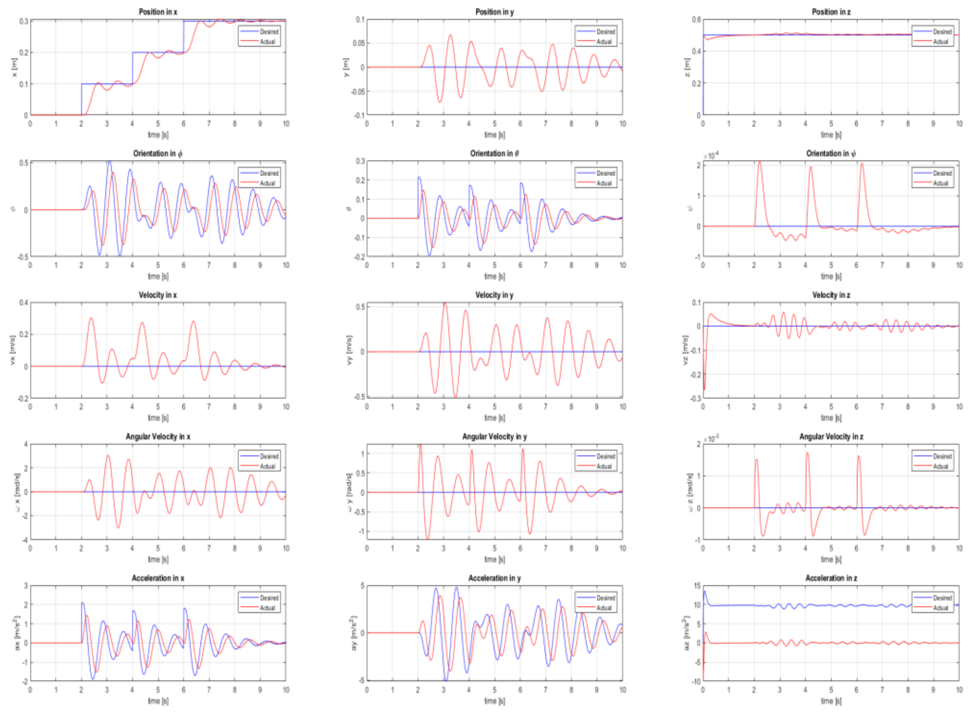

Figure 5. Actual State v/s Desired State (Set 2)

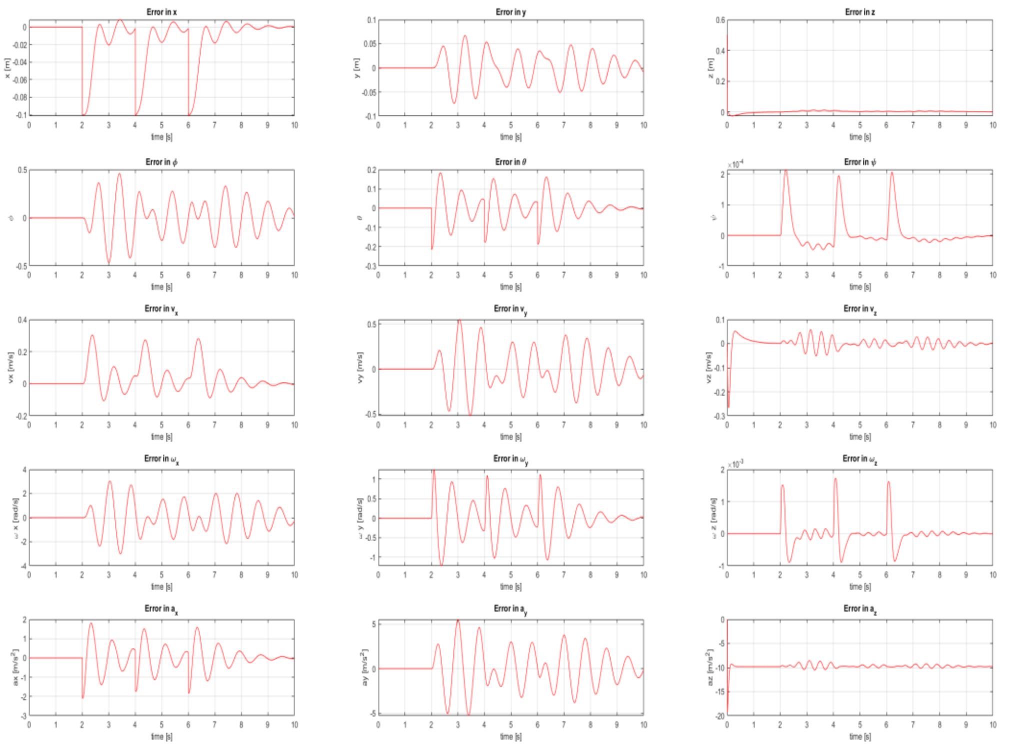

Figure 6. Plot of Error (Set 2)

The implementation of a Proportional-Derivative (PD) feedback controller is evaluated for the aerial robot's ability to hover and transition between multiple waypoints. Figure 3 and Figure 4 display the robot's positional, orientational, and velocity states in the x, y, and z dimensions, along with their respective errors over time. The desired values are plotted against the actual values recorded during the simulation, indicating the system's performance in tracking the set trajectory. From the error plots, it is evident that while the robot generally follows the desired trajectory, there are instances of significant deviation, particularly visible in the oscillations around the waypoints. These oscillations suggest that the PD controller parameters might require further tuning to dampen the system's response and improve stability. The error plots are crucial for identifying the precise moments when the robot deviates from the desired pose and velocity, which correspond to the introduction of new waypoints. The system's oscillatory behavior around the waypoints indicates a need to assess the controller gains and possibly implement more sophisticated control strategies to achieve a smoother convergence to the desired states without overshooting or oscillating.

Figure 5 and Figure 6 show the impact of modifying the gains in the position and attitude control loops of an aerial robot's PD feedback controller. Adjusting the gains has a direct effect on the performance metrics of the system, such as overshot, settling time, and steady-state error. From the Figure, it is apparent that certain gain values may cause the system to oscillate more, indicating a potentially underdamped response. Conversely, other gain adjustments could lead to a more damped response, reducing oscillations and improving the robot's ability to precisely follow the desired trajectory and maintain a stable hover.

Line-Tracking Performance

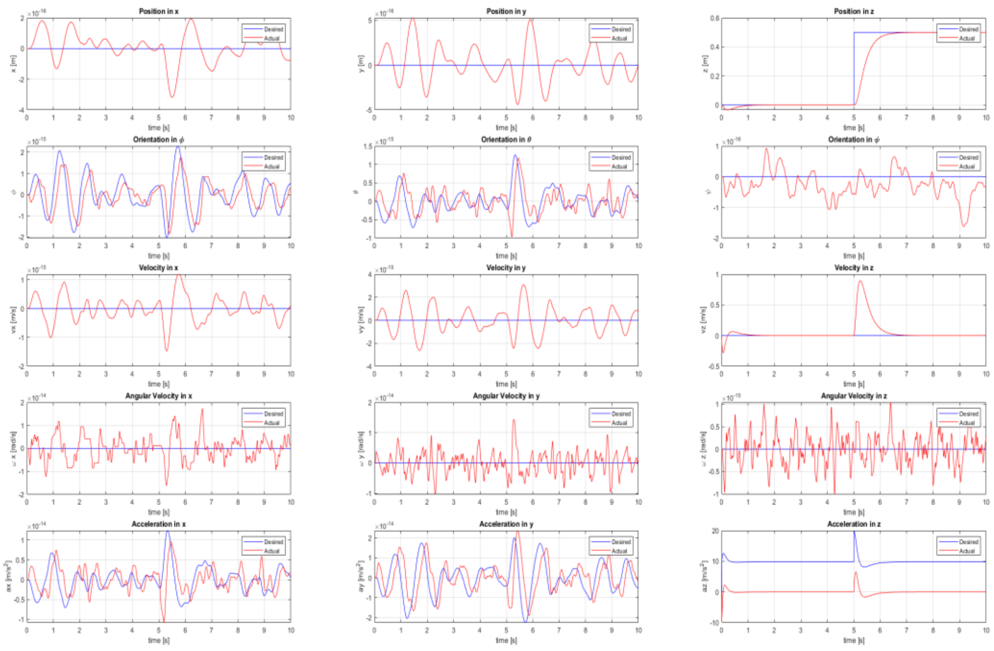
Figure 7. Actual State v/s Desired State (Set 1)

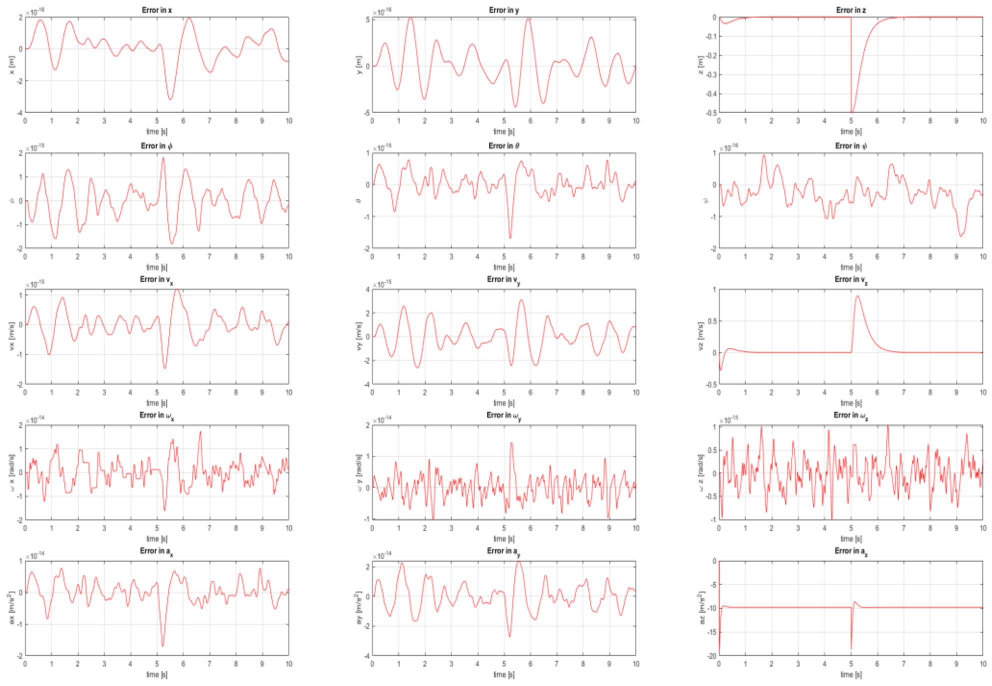
Figure 8. Plot of Error (Set 1)

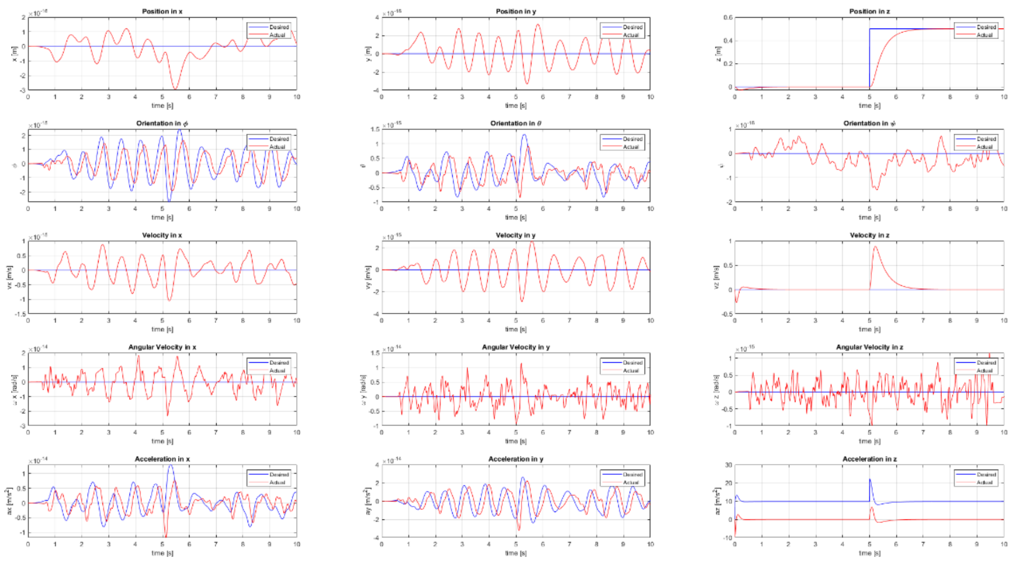

Figure 9. Actual State v/s Desired State (Set 2)

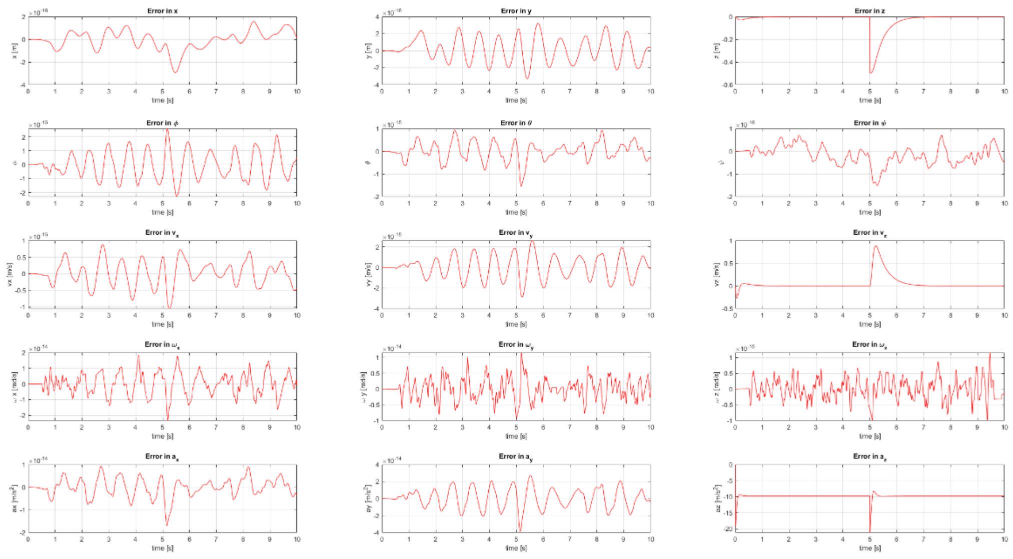

Figure 10. Plot of Error (Set 2)

Figures 7 to 10 illustrate the performance of a PD line tracking controller for an aerial robot's take off to cruise at a fixed height. These plots compare actual states versus desired states for two different sets of control gains and also shows the resulting error between actual and desired state.

In Set 1, the robot experiences significant oscillations in position, orientation and velocities as shown in error values (Figure 8), however they are most are at very small magnitudes. Since they did not have any movement input, it is not expected for them to move. For the values that did have inputs (z and yaw), their greater error values indicate that the gain might not be optimally tuned. The oscillations around the waypoints suggest that the controller might be overly aggressive or not damped properly, leading to an underdamped response.

Set 2 displays a distinct response, where error plots (Figure 8 and Figure 10) show oscillatory behavior, suggesting that the change in gains has affected the system's performance. The oscillations are less severe in Set 2, showing a better damped system but still not at the desired level of precision.

Both sets of figures demonstrate the sensitivity of the system to gain values in the position control (outer loop). Proper tuning of these gains is critical for minimizing errors and achieving smooth convergence to waypoints without sustained oscillations.

State Machine

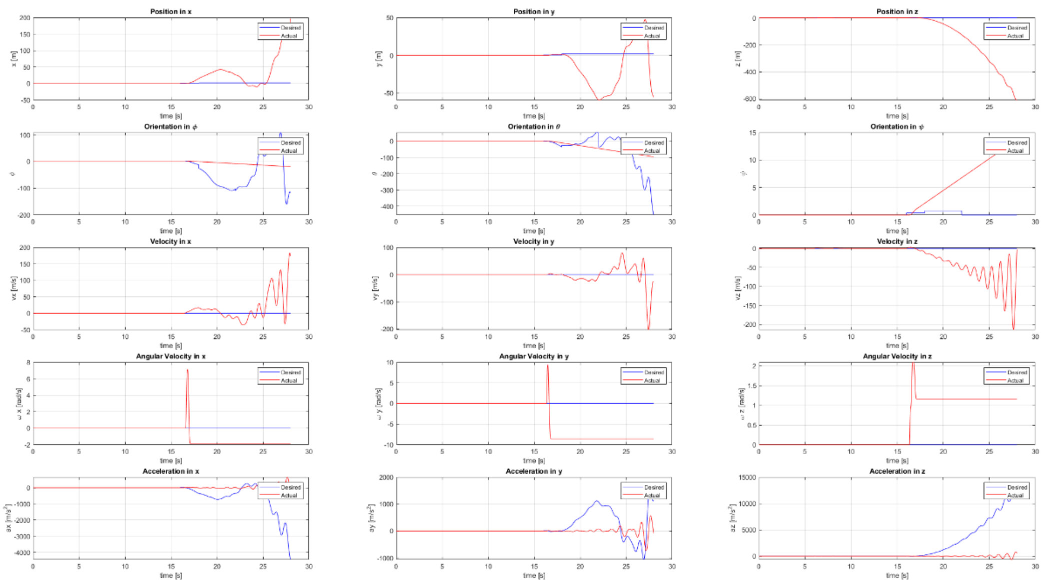

Figure 11. Actual State v/s Desired State

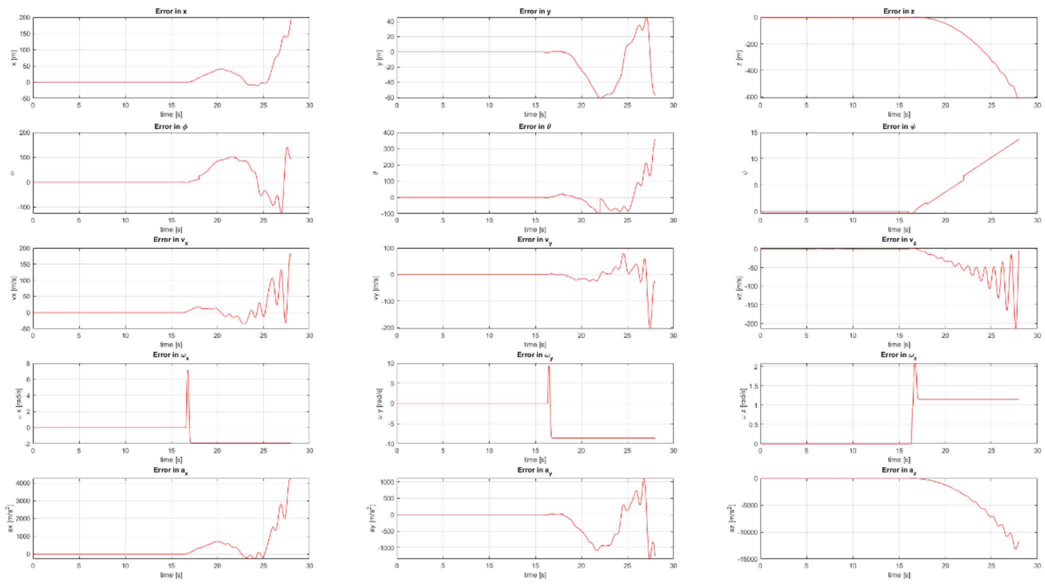

Figure 12. Plot of Error

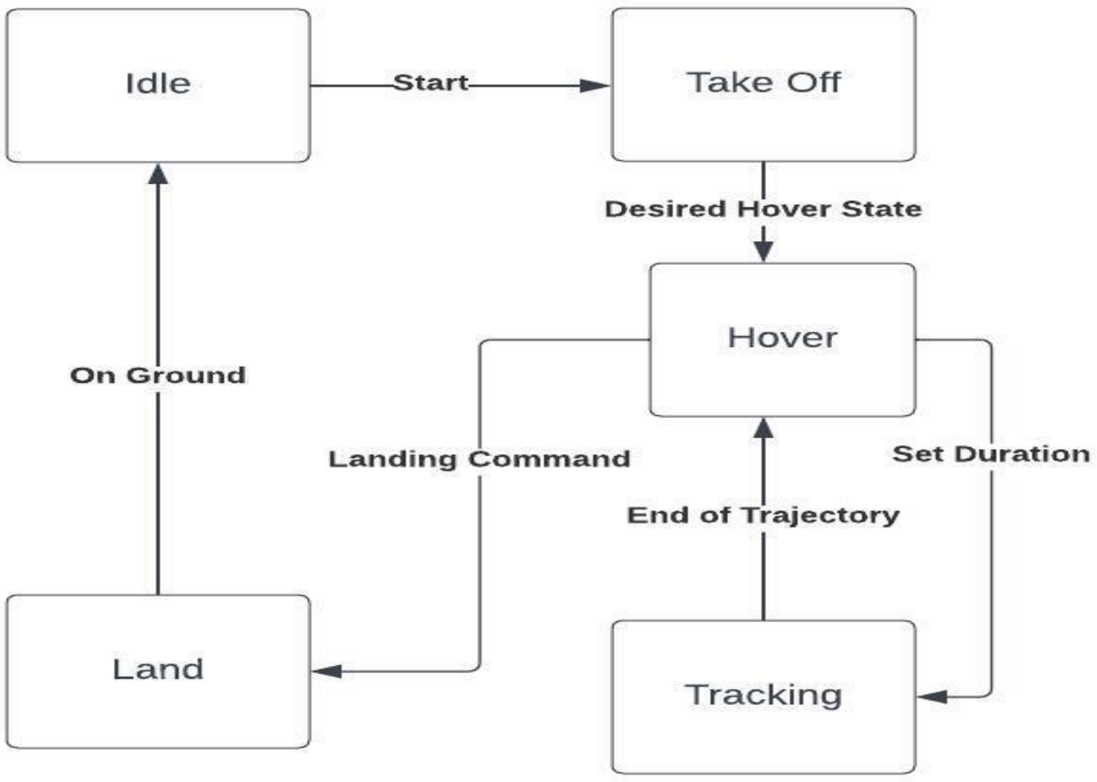

Figure 13. State Machine Diagram

Figure 13 is a State Machine Diagram for the aerial robot, designed to manage various phases of flight, including takeoff, hovering, trajectory, tracking, and landing. The system starts at an **Idle** state where no control inputs are generated. After receiving Start command, it transitions to **Take Off** state, where a trajectory tracking controller guides the robot to a predefined hover altitude. Once it reaches its desired hover, it transitions to **Hover** state. While in **Hover** state, the robot maintains its position for a set duration, ensuring stability before it proceeds to **Tracking** state, where it follows a predetermined path. Upon completion of this path, the robot re-enters into **Hover** state to gain stability. After Landing command, transitioning to **Land** state to descend and touch down safely. After landing, it reverts to **Idle** state, completing the flight cycle and ready for next operation.

Gain Selection and Tuning

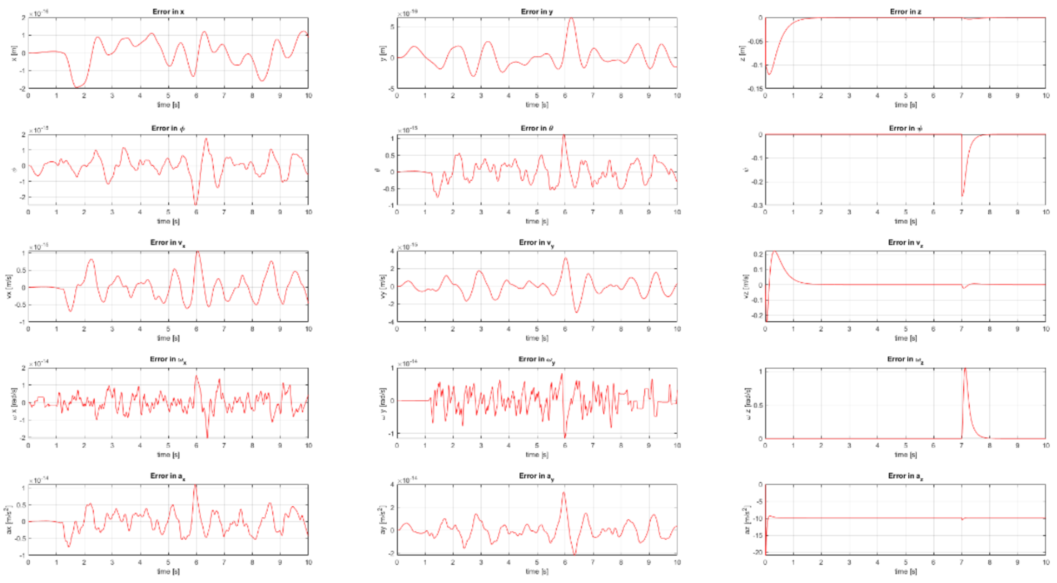

Figure 14. Plot of Error for Base Gains

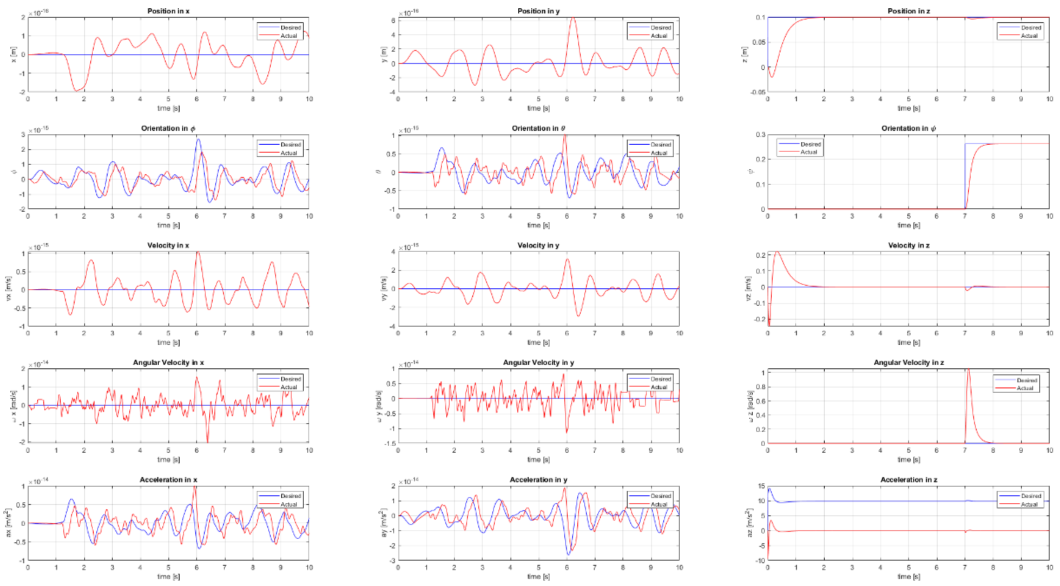

Figure 15. Plot of Pose for Base Gains

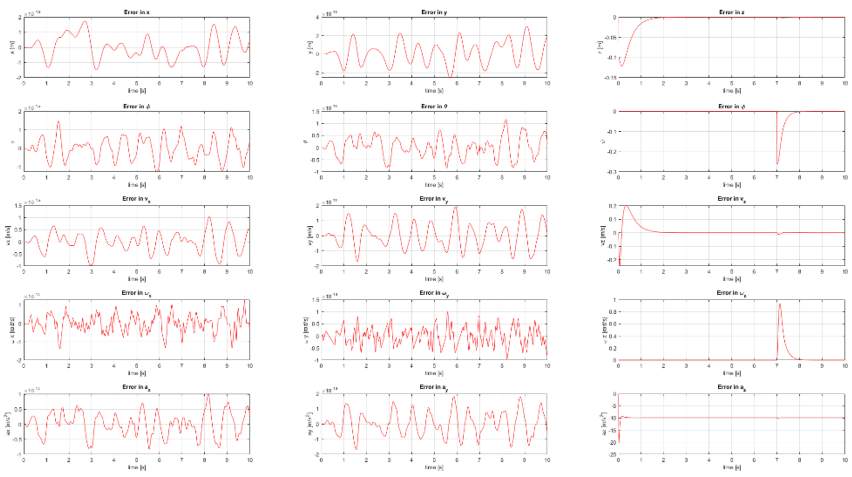

Figure 16. Plot of Error for Gain Set 1

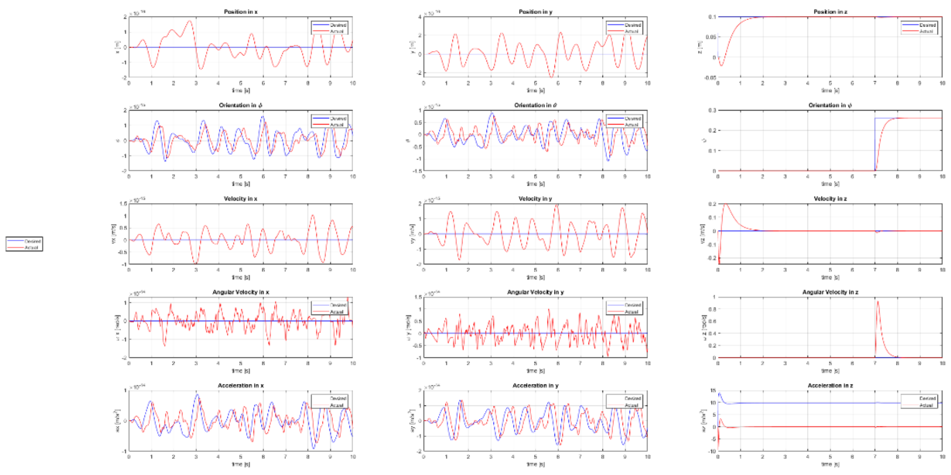

Figure 17. Plot of Pose for Gain Set 1

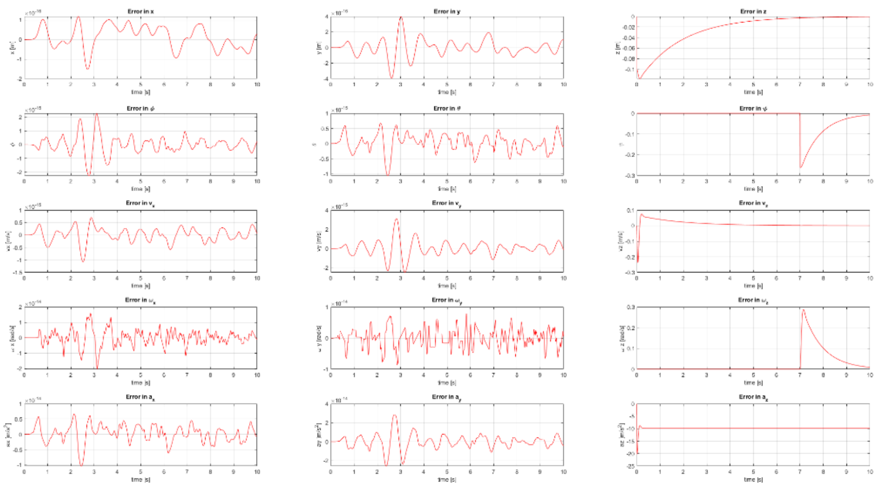

Figure 18. Plot of Error for Gain Set 2

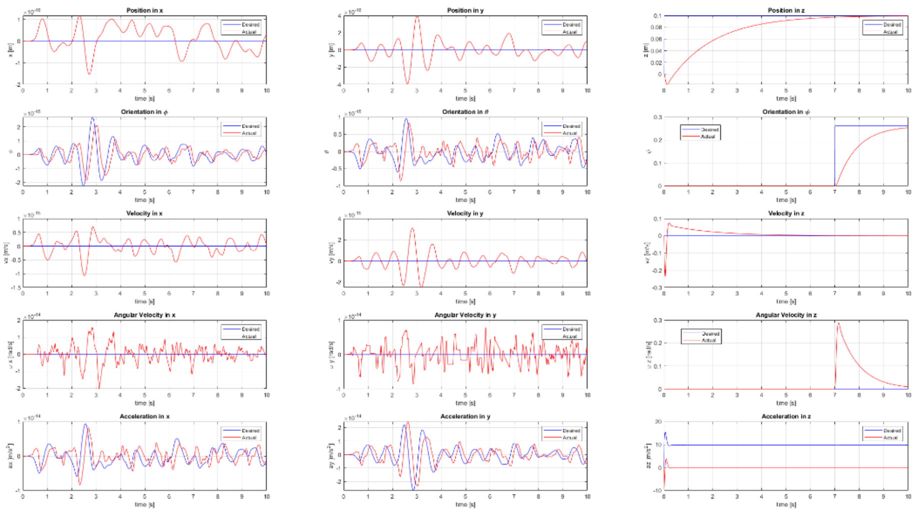

Figure 19. Plot of Pose for Gain Set 2

Figures 14 to 19 represent an initial input – a change in the height – to an assumed hover state. The input causes the aerial robot to track to the new height at the first-time step and settle on the new position. In addition to a change in the height, there was also an input to the yaw that causes the robot to increase its heading by 15°. Both of these changes can be seen differently in the set of three error and pose graphs depending on the gain values.

Comparing the three sets of data, the second set of gain values shows a much slower reaction time to the input compared to the first and base set. The slow damping came from reducing the gain values drastically on z and heading. The first set of gain values shows little change from the base set, even those most gain values were lowered. When implementing a heading change, it can also be noted that there is a drop in height from the already stabilized height of 0.1 meters.

	X	Y	Z	Xdot	Ydot	Zdot	Psi
Rise [s]	1.6	0.3	0.7	0.2	0.1	5e-7	0.4
Settling [s]	10	10	10	10	10	10	7.7
Max Percent Overshoot [%]	40.5	115.3	5.3e-4	50.7	874	1.2e7	0
Steady-State Value	0	0	0.1	0	0	0	0.26

Table 1. Control Analysis Performed by Matlab's 'Stepinfo' Command

The table above shows the rise time, settling time, max percent overshoot, and steady-state value of the position, velocity, and heading using the base gain values in question five. These values were calculated using the Matlab's 'Stepinfo' command, so depending on how it read the graphs can determine how accurate the values are. For example, the settling time is ten seconds for most of the states, but it can be seen on the graphs that they reach a steady value before the ten second mark. For the maximum percentage overshoot in the y velocity and z velocity, there were very large, quick spikes that occurred and caused the overshoot value to skyrocket and are not representative of the entire system when stabilizing.

Conclusion

This Matlab simulation program effectively demonstrates the complexity and capabilities of designing a control system for quadrotors using Linear Approximation. By implementing control pipeline using trajectory and attitude planning, position and attitude control, control allocation and motor control, the project emphasizes between desired and actual state using different gain values for the control system. Through the careful selection and tuning of gains, the aerial robot's performance was finely optimized, achieving stable hover, and precise waypoint transitions. The state machine showed the importance of structured mode of transitions in autonomous system, enabling the aerial robot to perform complex sequences of actions like takeoff, hovering, trajectory tracking and landing.