UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

SCHOOL OF MECHANICAL,
INDUSTRIAL & AERONAUTICAL
ENGINEERING

Drone Control Assignment - Group 28

MECN4029A - Mechatronics II

Gavriel Dirmeik	2126237
Tiaan Geldenhuys	2850040
Runé Edeling	2400293
Moshe Jacobson	2461271

A project report submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Bachelor of Science in Engineering.

Johannesburg, May 2025

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

SCHOOL OF MECHANICAL, INDUSTRIAL & AERONAUTICAL ENGINEERING

Disclosure - Use of Artificial Intelligence (AI) Generated Content 2025

Students must acknowledge all use of AI.

Select all applicable statements and complete sections 2, 3, 4, and 5 if applicable.

1. ***Disclosure: Generated / manipulated text - list each occurrence***

   ☒ We acknowledge the use of GPT, 4, April, 2024 (https://chat.openai.com/) to rewrite the content of the report making it sound formal and professional. We entered the following prompt(s) on Prompt date 04,05/2024:

   'Rewrite the following text in formal academic English, suitable for an engineering report. Pay attention to the tone and language used to ensure that it is appropriate for an engineering report'

   The output from these prompts was used to throughout the report.

2. ***Disclosure: Generated / manipulated code - list each occurrence***

   ☒ We acknowledge the use of MatLab AI Chat Playground, April, 2024 (https://www.mathworks.com/matlabcentral/playground/new) to Rewrite, edit and logic check code. AI helped to look for bugs and logic errors throughout the code as well as neaten up and add efficiency where needed.. We entered the following prompt(s) on 12 April 2024:

   'Prompt me to select a folder of images, then read them, and let me select pixel co-ordinates with my mouse'

   The output from these prompts was used to make a framework of codes I don't know and then develop it.

☒ We declare that this disclosure is complete and truthful.

Student number: 2461271 2850040 2400293 2461271

Course code: MECN4029A

Date: 21 May 2025

Contents

List of Figures

List of Tables

1 Executive Summary

This report details the design, analysis, and implementation of a control system for a quadrotor drone operating within a server room environment. The aim of the project was to develop a reliable and responsive drone control strategy capable of stabilising the vehicle in all six degrees of freedom and enabling complex autonomous manoeuvres for thermal inspection in constrained indoor spaces.

A complete non-linear mathematical model of the drone was developed based on Newton–Euler rigid-body dynamics, capturing the full coupling between translational and rotational states. The model was linearised about the hover equilibrium to allow classical control design using transfer functions, root locus, and Nyquist plots. Stability analysis confirmed that the open-loop system is marginally stable and requires active feedback control for practical use.

A cascaded control architecture was adopted. The inner loop directly regulates the altitude and attitude angles using PD and PID controllers designed via analytical coefficient-matching methods. Outer-loop PID controllers were implemented for horizontal positioning, using inner-loop attitude commands to achieve thrust reorientation. While the linear model permitted analytical tuning, the outer-loop gains were manually adjusted to achieve stable performance due to the complexity of the coupled dynamics.

The control system was tested under several challenging scenarios, including free-fall, inverted orientation, and a free-hand throw. Both linear and non-linear simulations confirmed that PID control eliminated steady-state error and significantly improved robustness compared to PD control. A trajectory-following simulation demonstrated full system stability during dynamic manoeuvres through a 3D server room layout.

Hardware deployment was intended but hindered by technical limitations; a simulated trajectory animation was used as an alternative demonstration. Despite these constraints, the project successfully validated the effectiveness of the designed control system and demonstrated its potential for real-world application in thermal inspection tasks within data centre environments.

2 System Parameters and Modelling

2.1 Drone Control Scenario

High-density server rooms concentrate significant electrical power within compact spaces, leading to considerable heat generation from electrical hardware. Typically, server racks can be powered through overhead cable trays that deliver high current to the top of each server rack [1]. Under continuous load, conductors can heat up, and because hot air rises, the upper sections of the racks tend to become hot zones [2]. Therefore, managing heat in server rooms is essential but increasingly difficult, driven by evolving hardware designs, higher equipment densities, and ageing infrastructure [3]. These factors complicate both airflow control and the detection of thermal issues [3]. In cases where thermal build-up goes undetected, the consequences can lead to reduced hardware lifespan, intermittent failures, or, in severe cases, fire hazards [4].

From an engineering perspective, this presents a monitoring and fault prevention problem that requires a solution capable of: continuously identifying localized overheating before critical failure occurs, mapping thermal profiles within a server room, and seamlessly integrating into the facility's existing monitoring systems. To address this problem, a quadcopter drone equipped with a thermal imaging camera is proposed. The drone will follow a pre-programmed path through server aisles, hovering above racks to scan for abnormal temperature patterns. Thermal imagery will enable early detection of hotspots and proactive maintenance responses, while the drone allows continuous inspection of numerous racks, supporting real-time, scheduled, or on-demand surveys.

To proceed with the design development, the quadcopter control scheme will be designed and validated for a small-scale server room environment as shown in Figure 1 below [5]. Once operational, the designed control strategy can be scaled to accommodate larger data centers with more complex rack layouts, ensuring the broad applicability and robustness of the solution.

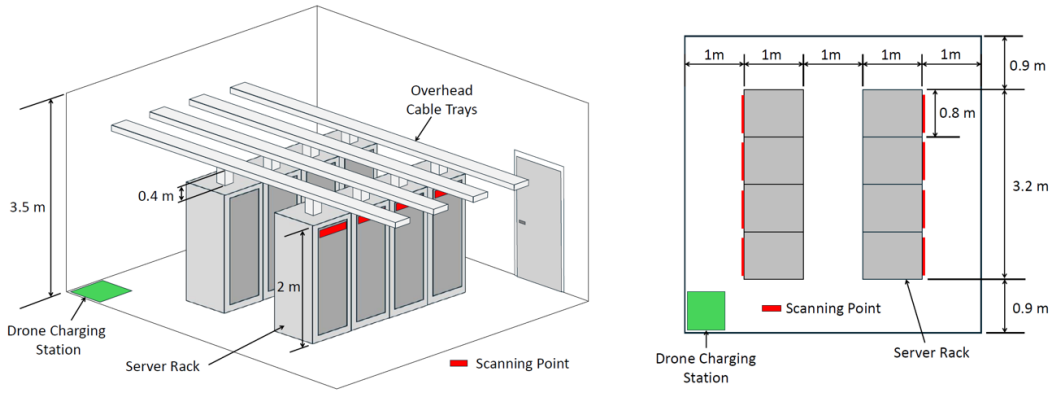


Figure 1: Drone space used for design development [5].

2.2 Non-Linear System Model

The quadrotor is modelled as a fully nonlinear rigid-body system consisting of four rotors arranged in a "plus" (+) configuration, as shown in Figure 2 [6]. Its key physical properties are summarised in Table 1 [6]. Two right-handed Cartesian coordinate frames are used: an inertial (global) frame, which is fixed to the Earth and defined by axes X, Y, and Z (with Z pointing upward); and a body-fixed (local) frame centred at the drone's centre of mass, with x pointing forward through rotor 1, y pointing left through rotor 4, and z pointing upward [6]. Additionally, the drone's orientation is described using Euler angles ϕ (roll about the x-axis), θ (pitch about the y-axis) and ψ (yaw about the z-axis) [6].

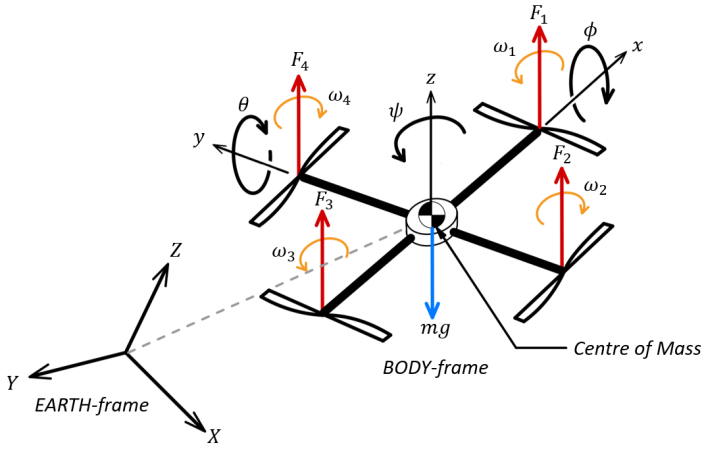


Figure 2: Illustration of the drone model with body and earth reference frames [6].

Table 1: Quadcopter Parameters [6]

Parameter	Description	Value
m	Mass	$0.506\ \mathrm{kg}$
I_x	Moment of inertia about the x body axis	$8.11858 \times 10^{-5}\ \mathrm{kg \cdot m^2}$
I_y	Moment of inertia about the y body axis	$8.11858 \times 10^{-5}\ \mathrm{kg \cdot m^2}$
I_z	Moment of inertia about the z body axis	$6.12223 \times 10^{-5}\ \mathrm{kg \cdot m^2}$
l	Arm length	$0.235\ \mathrm{m}$
k_F	Thrust coefficient	$3.13 \times 10^{-5}\ \mathrm{N/(rad/s)^2}$
B	Drag constant	$75 \times 10^{-7}\ \mathrm{N \cdot m/(rad/s)^2}$

Each rotor, i, spins at angular velocity ω_i, producing a thrust $F_i = k_f \omega_i^2$ along the body z-axis and a reactive torque $\tau_\psi = K_m \omega_i^2$ about the same axis [7]. Additionally, each rotor produces a torque about the axis perpendicular to the arm by which it is connected. Rotors 1 and 3 generate a pitching torque $\tau_\theta = lF_i$, while rotors 2 and 4 generate a rolling torque $\tau_\phi = lF_i$ [7].

The total thrust ($U_1 = K_F(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)$) accelerates the drone in the body z-axis, while total roll ($U_2 = lK_f(\omega_4^2 - \omega_2^2)$); total pitch ($U_3 = lK_f(\omega_3^2 - \omega_1^2)$); and total yaw ($U_4 = B(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2)$) generate angular accelerations that act to rotate the drone about each of it's body axes [7]. Therefore, the angular velocities of each of the rotors can be directly related to the state of U_1 to U_4 via the following matrix conversion [7]:

$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \begin{bmatrix} K_F & K_F & K_F & K_F \\ 0 & -lK_F & 0 & lK_F \\ -lK_F & 0 & lK_F & 0 \\ -B & B & -B & B \end{bmatrix}^{-1} \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{1}$$

Since the drone is an under-actuated system, i.e. it has more degrees of freedom ($x, y, z, \phi, \theta, \psi$) than the number of control inputs (U_1, U_2, U_3, U_4), motion in the horizontal x direction and vertical y direction cannot be achieved directly [7]. Instead, to generate planar acceleration, the drone must be tilted in order to redirect the thrust vector [7]. To illustrate, a non-zero θ produces motion along X, whilst a non-zero ϕ produces motion along Y. Lastly, yawing by ψ re-orients the body axes without directly producing horizontal acceleration.

Moreover, hovering is maintained when the total thrust vector balances the drone's weight by:

$$U_1 = mg \tag{2}$$

where $\phi = \theta = \psi = 0$, yielding zero translational and rotational accelerations and ensuring that the entire thrust vector is oriented parallel to the Z-axis from the earth reference frame. In this way, the drone maintains an upright constant elevation.

The Full Newton-Euler equations can be expressed as [7], [8]:

$$m\dot{\mathbf{v}}_{\mathcal{B}} = m\mathbf{g} + R_{\mathcal{B}\to\mathcal{I}}(\phi, \theta, \psi) \begin{bmatrix} 0 \\ 0 \\ U_1 \end{bmatrix} \tag{3a}$$

$$\mathbf{I}\dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I}\boldsymbol{\omega}) = \begin{bmatrix} U_2 \\ U_3 \\ U_4 \end{bmatrix} \tag{3b}$$

where $\mathbf{v} = [u, v, w]^\mathsf{T}$ are body-frame velocities, $\boldsymbol{\omega} = [p, q, r]^\mathsf{T}$ the body-frame angular rates, $\mathbf{I}$ the inertia tensor and $R_{\mathcal{B}\to\mathcal{I}}$ the rotation matrix [7]. The rotation matrix maps the body-fixed forces into the inertial, as can be seen below [7].

$$R_{\mathcal{B}\to\mathcal{I}} = \begin{bmatrix} \cos\theta\ \cos\psi & \cos\theta\ \sin\psi & -\sin\theta \\ \sin\phi\ \sin\theta\ \cos\psi - \cos\phi\ \sin\psi & \sin\phi\ \sin\theta\ \sin\psi + \cos\phi\ \cos\psi & \sin\phi\ \cos\theta \\ \cos\phi\ \sin\theta\ \cos\psi + \sin\phi\ \sin\psi & \cos\phi\ \sin\theta\ \sin\psi - \sin\phi\ \cos\psi & \cos\phi\ \cos\theta \end{bmatrix}$$

Since this coupling matrix involves terms such as sine, cosine and products of angles, it introduces trigonometric non-linearities whenever the drone attitude departs from zero.

By expressing these second-order equations as a set of twelve first-order ordinary differential equations, one obtains the non-linear state-space formulation [7]

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{U}) \tag{4}$$

with state space vector:

$$\mathbf{x} = [x, y, z, \dot{x}, \dot{y}, \dot{z}, ; \phi, ; \theta, ; \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}]^\mathsf{T} \tag{5}$$

where (x, y, z) are inertial positions, (u, v, w) the body-frame velocities, (ϕ, θ, ψ) the Euler angles, and (p, q, r) the angular rates [7]. This representation retains the full coupling between translation and rotation—including thrust re-orientation, gravity, and gyroscopic effects—and thereby provides a realistic basis for both time-domain simulation and controller design [7].

Moving on, utilizing the state space vector format along with the following assumptions made, as shown in Table 2 [6], [9], the design development for the non-linear drone model was ready.

Table 2: Modeling Assumptions [6], [9].

#	**Assumption**
1	The vehicle's centre of mass is fixed at its geometric centre.
2	The quadrotor behaves as a perfectly rigid body of fixed mass m, thereby neglecting structural flexibilities.
3	Each rotor produces thrust $F_i = k_F \omega_i^2$ and reactive torque $\tau_i = k_M \omega_i^2$ with constant coefficients k_F, k_M, so unsteady aerodynamic effects, blade flapping, rotor flapping dynamics, and motor electrical dynamics are omitted.
4	Aerodynamic drag on the vehicle body, rotor–rotor interference, and ground-effect phenomena are neglected.
5	The gravitational acceleration g is treated as a uniform constant acting in the inertial Z direction.
6	Actuators are considered ideal, with instantaneous response, unlimited bandwidth, and no saturation or time-delay.
7	Any variation in atmospheric density is neglected.

MatLab code that simulates the basic behaviour of the drone in a non-linear manner can be seen in Appendicies **??**. The somewhat trivial case of the drone in a hovered state at a height of 1 m can be seen in Figure 3 below. This is shown to familiarize the reader with the format of the data output.

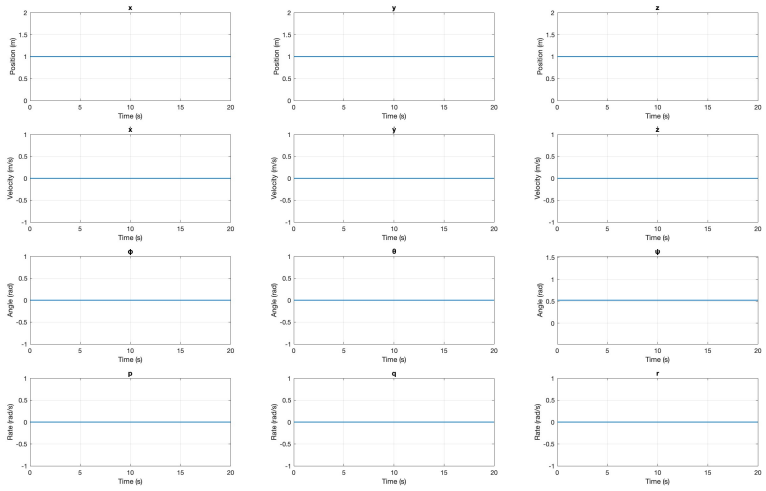


Figure 3: Drone in a Hovered State modelled in a Non-Linear Simulation

2.3 Linearised System

2.3.1 Deriving the Linear System

In order to perform any meaningful analysis of the system, linearisation of the full non-linear model about the hover equilibrium is essential because classical control-design methods such as root-locus, Routh–Hurwitz, Nyquist and Bode-plot analysis require a linear, time-invariant representation.

The process entails expanding the non-linear dynamic equations about the operating point defined by $\phi = \theta = \psi = 0$ and $U_1 = mg$, then retaining only first-order terms to yield a set of linear differential equations. This approximation remains valid for the small attitude and thrust deviations encountered during inner-loop stabilisation and produces transfer-function forms that allow the analytical determination of pole-zero locations, systematic gain tuning and verification of stability margins. Without such linearisation, application of these powerful to the fully coupled non-linear model would be prohibitively complex.

Linearisation proceeds by expanding the non-linear state-space equations about the hover operating point and retaining only first-order perturbations.

Denote the equilibrium as $\mathbf{x}_0$ satisfying $\dot{\mathbf{x}} = 0$ when $\phi = \theta = \psi = 0$,$U_2 = U_3 = U_4 = 0$ and $U_1 = mg$, and introduce perturbation variables $\delta\mathbf{x} = \mathbf{x} - \mathbf{x}_0$ and $\delta\mathbf{U} = \mathbf{U} - \mathbf{U}_0$ [7].

The vector field $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{U})$ is then approximated by its first-order Taylor expansion $\dot{\delta\mathbf{x}} \approx A\,\delta\mathbf{x} + B\,\delta\mathbf{U}$ [7], where:

$$A = \left.\frac{\partial f}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0,\,\mathbf{U}=\mathbf{U}_0} \tag{6a}$$

$$B = \left.\frac{\partial f}{\partial \mathbf{U}}\right|_{\mathbf{x}=\mathbf{x}_0,\,\mathbf{U}=\mathbf{U}_0} \tag{6b}$$

In practice, this entails linearising trigonometric terms via small-angle approximations (e.g. $\sin\phi \approx \phi$, $\cos\phi \approx 1$) and evaluating the partial derivatives of both the thrust–gravity coupling in the translational dynamics and the gyroscopic and rotor-torque terms in the rotational dynamics. The resulting time-invariant model $\dot{\delta\mathbf{x}} = A\,\delta\mathbf{x} + B\,\delta\mathbf{U}$, $\delta\mathbf{y} = C\,\delta\mathbf{x}$ admits a transfer-function representation for each SISO channel [7].

The extraction of each SISO transfer function proceeds in four steps [10].

1. **Write the linearised state-space model:**

$$\dot{\delta\mathbf{x}} \;=\; A\,\delta\mathbf{x} \;+\; B\,\delta\mathbf{U}, \quad \delta\mathbf{y} \;=\; C\,\delta\mathbf{x} \;+\; D\,\delta\mathbf{U}$$

2. **Transform to the frequency domain:**

   $s\,\Delta X(s) = A\,\Delta X(s) + B\,\Delta U(s)$

   hence,

   $\Delta X(s) = \big(sI - A\big)^{-1}\,B\,\Delta U(s)$

3. **Substitute into the output equation to obtain the transfer-function matrix:**

   $\Delta Y(s) = C\,(sI - A)^{-1}\,B\,\Delta U(s) \;+\; D\,\Delta U(s)$

   yielding,

   $G(s) = C\,(sI - A)^{-1}\,B \;+\; D.$

This process was carried out for each of the four input-output pairs seen in Table 3 below.

Table 3: Transfer Functions of Four Directly Actuated States [7].

State	Output/Input	Transfer Function
Altitude	z/U_1	$G_{\frac{z}{U_1}}(s) = \big[G(s)\big]_{1,1} = \frac{1}{m\,s^2}$
Roll	ϕ/U_2	$G_{\frac{\phi}{U_2}}(s) = \big[G(s)\big]_{2,2} = \frac{1}{I_x\,s^2}$
Pitch	θ/U_3	$G_{\frac{\theta}{U_3}}(s) = \big[G(s)\big]_{3,3} = \frac{1}{I_y\,s^2}$
Yaw	ψ/U_4	$G_{\frac{\psi}{U_4}}(s) = \big[G(s)\big]_{4,4} = \frac{1}{I_z\,s^2}$

To obtain the Transfer Functions for the forward and lateral motion, this process cannot be done because they are not directly actuated by inputs. As mentioned above, forward and lateral motion arise because a non–zero attitude angle redirects the total thrust vector into the horizontal plane.

Consider forward motion. For small angles, $\tan\theta \approx \theta$, so the body–fixed thrust U_1 produces an inertial acceleration $\ddot{x} = g\theta$ [7]. In the Laplace domain this is represented by the second-order integrator $\frac{g}{s^2}$ [7]. Cascading this with the SISO attitude dynamics $\frac{\Theta(s)}{U_3}(s) = \frac{1}{I_y s^2}$ will yield the Transfer Function for an output of x and input U_3 [7]. Extending this method to lateral motion, the final two Transfer Functions may be determined and are shown in Table 4 below.

Table 4: Transfer Functions of the Two Indirectly Actuated States [7].

State	Output/Input	Transfer Function
Forward	x/U_3	$G_{\frac{x}{U_3}}(s) = \frac{g}{s^2}\;G_{\frac{\theta}{U_3}}(s) = \frac{g}{I_y\,s^4}$
Lateral	y/U_2	$G_{\frac{y}{U_2}}(s) = -\frac{g}{s^2}\;G_{\frac{\phi}{U_2}}(s) = -\frac{g}{I_x\,s^4}$

A linear model of the drone can be seen in Appendix E. The output of which appears the same as Figure 3 above.

3 Stability Analysis

3.1 Time Domain Comparison of Linear and Non-Linear Models

To compare the time-domain responses of the linear and non-linear systems, each control input channel U_1, U_2, U_3, and U_4 were independently excited with a step, ramp, and sinusoidal input. The resulting system responses were simulated in both models to assess dynamic behaviour under identical excitation conditions. The initial conditions for the simulation tests assumed the drone to be in a hover state, with all attitude angles set to zero, and no initial translational or rotational velocities. The thrust value was set to balance the drone's weight, such that $U_1 = mg$.

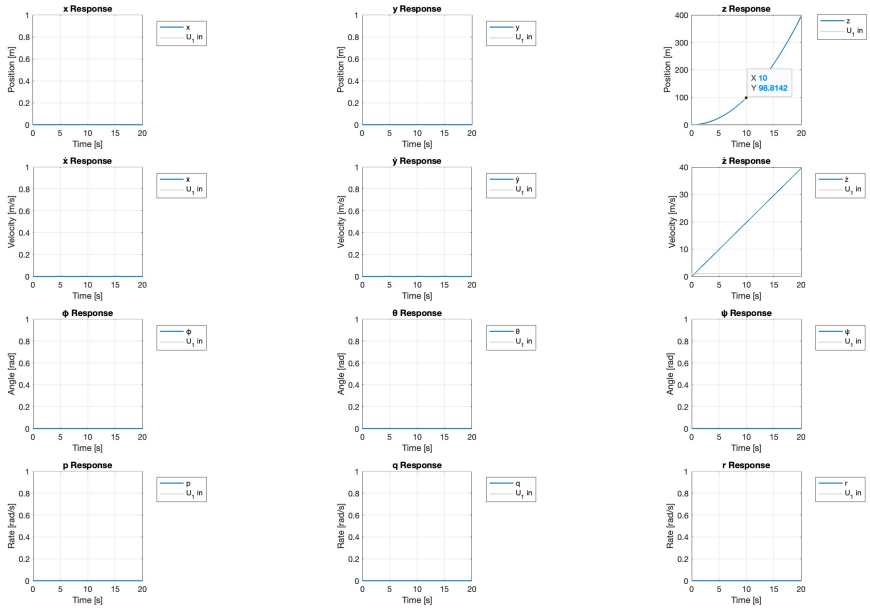
Although three types of input signals were considered, detailed analysis was only performed for each step input. The system responses to the ramp and sinusoidal inputs are provided in the Appendix L; however, further analysis was omitted to avoid redundancy and excessively long discussion. This decision is justified by the consistency of the observed behavioural differences between the linear and nonlinear models across all input types.

Owing to linearisation about the hover equilibrium, only the first-order, constant coupling terms (for example $\dot{u} = g\theta$ and $\dot{v} = -g\phi$) are retained in the state-space model, whereas all higher-order, state-dependent interactions are discarded. As a result, the linear simulation exhibits simplified, fixed inter-state couplings, while the full nonlinear model continues to reflect the varying, magnitude-dependent behaviour. This is evident in the analysis to follow.

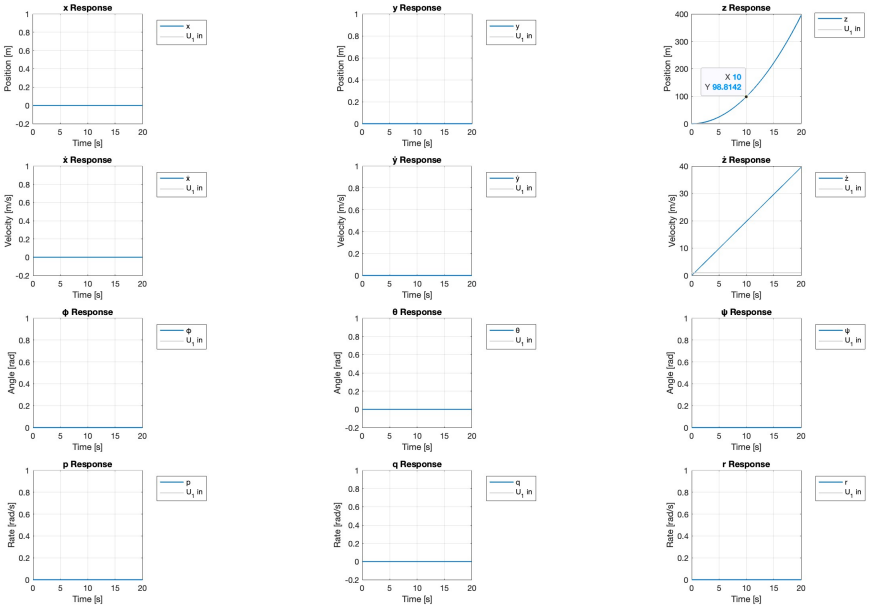
The code used for the simulation of these states can be seen in Appendix A (linear) and Appendix B (non-linear).

3.2 Drone Response to Inputs in U_1

Consider first, the responses to a step excitation in the input U_1, these can be seen in Figure 4 below. Each excitation had an amplitude of 1 Newton which results in an approximate acceleration of 2 m/s in the step input.



(a)



(b)

Figure 4: Response of the (a) linear and (b) non-linear systems to a step input in U_1 from an initial hovering state.

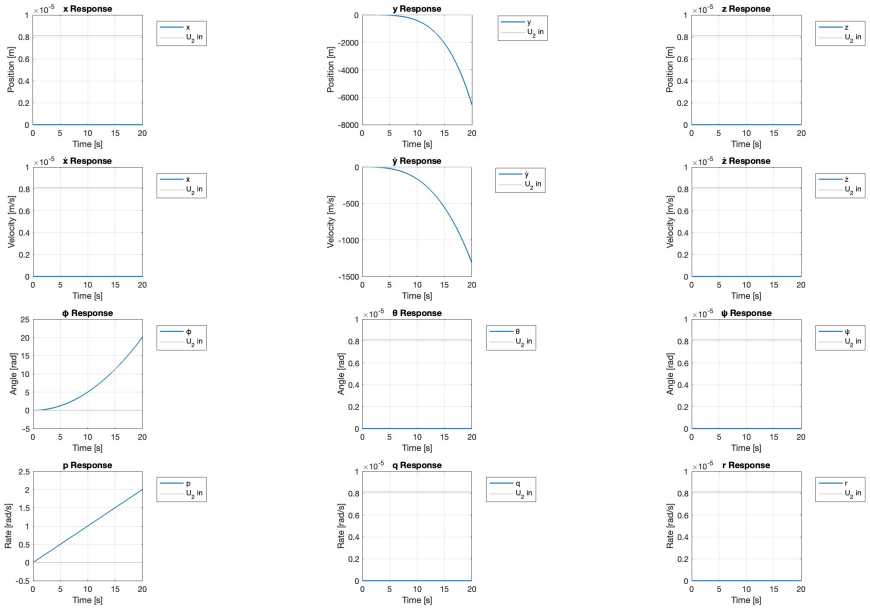
The responses to step inputs in U_1 are identical for both the linaer and non-linear models. In both simulations, z increases in the same manner while x, y, ϕ, θ, ψ all remain constant at zero. This outcome is expected, as the U_1 input does not introduce any non-linearities or coupling with states other than z. It simply issues an equal thrust command to all four rotors, generating a net upward force. Given that the drone is initially level, this results in a purely vertical acceleration in accordance with Newton's Second Law, resulting in $\ddot{z} = \frac{U_1}{m}$.

The step response, therefore, generates a constant vertical acceleration of and thus a linearly increasing vertical velocity and a quadratically increasing vertical position.

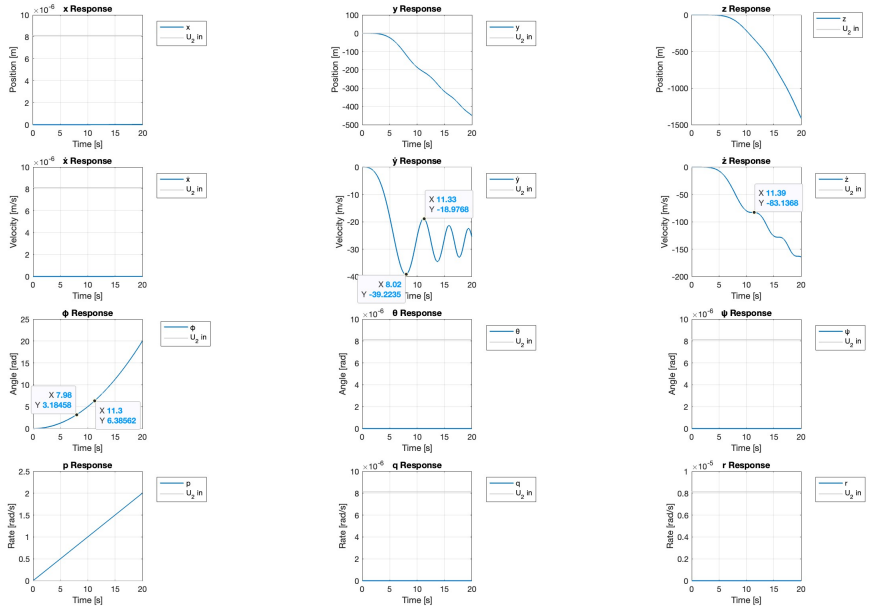
3.3 Drone Response to Inputs in U_2 and U_3

It is important to note here, that this analysis was performed for the system response to U_2; however, the response to U_3 demonstrates analogous behaviour. In that case, θ, influenced by U_3, is coupled with motion in the x-direction as apposed to ϕ, coupled with U_2 which is coupled to motion in the y-direction. Therefore, the analysis presented for U_2 extends naturally to U_3, and a separate analysis is not included.

The response of the system to inputs in U_2, which governs the roll angle, is now considered. The results of the two simulatins can be seen in Figure 5, where a step input of 8.12×10^{-6}N·m was applied. This magnitude was selected to produce an angular acceleration of approximately 0.1 rad/s for the step input.



(a)



(b)

Figure 5: Response of the (a) linear and (b) non-linear systems to a step input in U_2 from an initial hovering state.

Firstly, note that both simulations exhibit an identical increase in ϕ over time. This behaviour arises from the direct application of Newton's Second Law, given by $\ddot{\phi} = \frac{U_2}{I_x}$, which results in a constant

angular acceleration. Consequently, the angular velocity p increases linearly, while the roll angle ϕ increases quadratically, as observed in both the linear and nonlinear models. Although the linear model accurately captures the evolution of ϕ in response to the roll input, it fails to account for the effects that emerge from a changing roll angle, this become more apparent as ϕ becomes large due to the breakdown of the small-angle assumptions inherent to the linearisation process.

Consider first the responses of y and $\dot{y}$. As the roll angle, ϕ, increases, the thrust vector tilts, generating an acceleration in the y-direction, as discussed in Section 1. In the linear model, this lateral acceleration is approximated using the small-angle assumption as $\ddot{y} = -g\phi$, where g appears due to the linearisation around the hover equilibrium. In contrast, the non-linear model employs the more accurate relationship $\ddot{y} = -\frac{U_1}{m}\sin(\phi)$. Initially, both models produce similar results; however, as time progresses and ϕ increases, the predictions diverge. The linear model continues to apply a constantly increasing lateral acceleration, such that $\ddot{y} \propto t^2$. This leads to a cubically increasing velocity $\dot{y}$, and consequently, the y-position grows quartically with time, i.e., $y \propto t^4$.

On the other hand, the non-linear model displays a completely different response structure in y and $\dot{y}$ and this highlights a significant limitation of the linear simulation: its inability to accurately the physical behaviour associated with large roll angles.

When $\phi = \pi$, the drone is inverted, and any further increase in ϕ causes the thrust vector to rotate such that it now induces acceleration in the positive y-direction as opposed to the negative y-direction as it initially had. The linear model does not account for this effect at all. As a result, it continues to predict acceleartion in the negative y-direction and the drone continually accelerates, with an increasing rate as the roll increases as mentioned.

In contrast, the nonlinear model accurately captures the drone's inversion. At approximately 8 seconds, when the roll angle reaches $\phi = \pi$, the drone flips over. From this point, the $\dot{y}$ response begins to reduce in magnitude—becoming less negative—as the direction of thrust shifts into the positive y-direction. Around 11.3 seconds, when $\phi = 2\pi$, the drone completes another full rotation, causing the thrust vector to flip once again and reverse the direction of acceleration in y. This flipping process repeats at an increasingly rapid rate as the roll rate continues to grow, reducing the time between successive inversions. Eventually, the lateral velocity stabilises as the roll rate tends toward infinity and the direction of net thrust oscillates rapidly. This behaviour, although accurate mathematically is clearly unrealistic, however it highlights the complexity of the non-linear model and it's ability to pick up to the nuances inherent in the system.

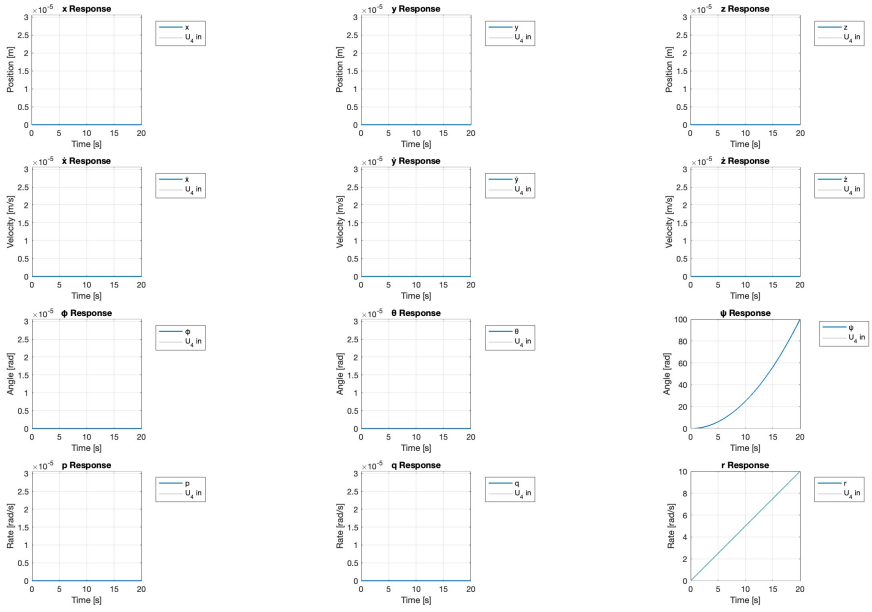
Another key limitation of the linear model is its inability to capture the coupling between roll angle and vertical motion. As the drone rotates and the thrust vector tilts away from the vertical, it no longer fully supports the drone's weight, resulting in a net downward force. This causes the drone

to descend—a behaviour entirely absent from the linear simulation, which assumes decoupled dynamics and maintains constant altitude. Further, each time the roll angle completes 2π radians of rotation, the thrust vector is once again directed upwards. Therefore, momentarily, the drone should be in a hover state with a zero acceleration in the z direction. This behaviour is accurately depicted in the non-linear model as well.

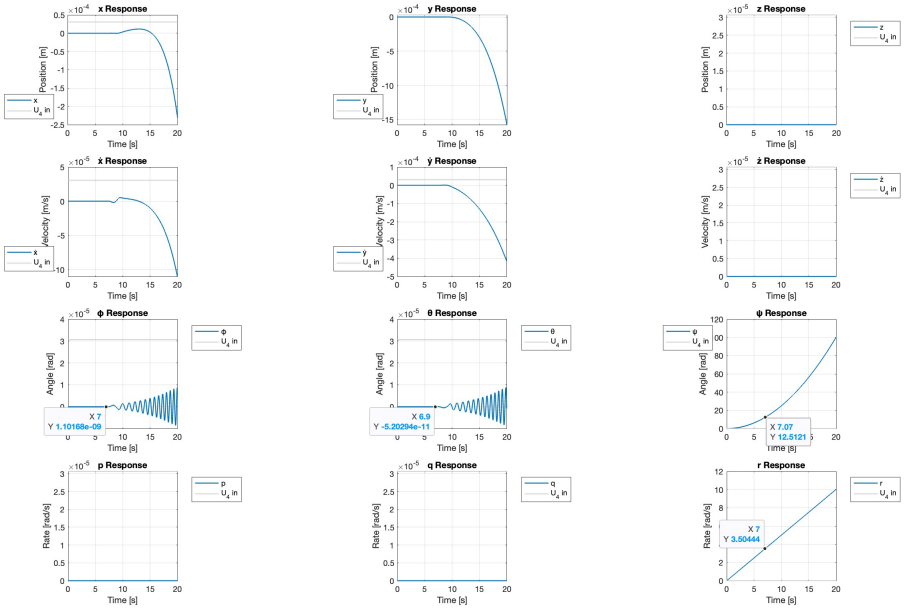
The final effect not captured by the linear model is the coupling between roll rate and yaw rate. In the nonlinear simulation, this interaction arises from the $\boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega}$ term in the Euler equations, which accounts for gyroscopic coupling due to the body's rotational inertia. As a result, a sustained roll induces a small yaw acceleration in the nonlinear model — a dynamic entirely absent from the linearised formulation. This will become more apparent in the analysis of U_4 to follow.

3.4 Drone Response to Inputs in U_4

The step input applied by U_4 had a value of 3.065×10^{-5}N·m, generating an angular acceleration of 0.5 rad/s, and the resulting responses of both models are presented in Figure 6 below.



(a)



(b)

Figure 6: Response of the (a) linear and (b) non-linear systems to a step input in U_4 from an initial hovering state.

Observing the two responses, they can be seen to be identical at low yaw rates (less than about 3.5 rad/s occuring at around 7 s) while as the yaw rate increases, their responses become more divergent. This is illustrating a key element of the drone dynamics lost in the linearization process.

The moment about the z-axis is governed by the same relationship in both the linear and nonlinear models, given by $\ddot{\psi} = \frac{U_4}{I_z}$. Consequently, both models yield identical results for ψ and r with time as the torque generated by the step input applies a constant angular acceleartion which tends to increase the angular position, ψ, quadratically.

However, what the linearised model fails to capture, is the coupling that the yaw rate has with the roll and pitch angles. At low yaw rates, this coupling is not negligible, however as r increases, it becomes more apparent. After about seven seconds, the increasing yaw rate begins to induce small secondary roll and pitch angles, captured in the non-linear model through the coupling term $\boldsymbol{\omega} \times \mathbf{I}\boldsymbol{\omega}$ in the Euler equations. These induced variations in roll and pitch further result in small accelerations in the x and y directions as the direction of the thrust vector shifts. As the yaw rate grows, so too do these affects. However, it should be noted that even after 20 seconds with a yaw rate of 10 rad/s (velocities reached by power drills), the ϕ and θ angles are on the order of 10^{-5} which allows them to be confidently neglected.

3.4.1 Analysis Conclusion

Overall, the linear model provides an accurate representation of the drone's behaviour under small, nominal motions. The thrust dynamics are modelled precisely, and the yaw response closely matches that of the nonlinear model. The primary limitation lies in the handling of extreme roll and pitch angles, where the drone becomes fully inverted—a phenomenon not captured in the linear analysis. Nevertheless, the linear model offers a reliable foundation for assessing system stability and designing control strategies, with the understanding that discrepancies may arise under more extreme conditions.

3.5 Stability Analysis

With the system linearised and transfer functions extracted, a stability analysis can now be performed using Nyquist plots to assess the nature of the drone's stability about the hover equilibrium.

Before conducting this formal analysis, it is instructive to consider the system's expected stability characteristics from a physical perspective.

When the drone is hovering and subjected to a perturbation in the vertical position, it would be expected that it remain hovering at the new altitude, as no restorative force is inherently generated to bring it back to it's initial hover position. This implies marginal stability in the z-direction. The same reasoning extends to the x and y directions: displacements result in the drone maintaining its new position, indicating marginal stability in translational motion.

A similar response is anticipated for perturbations in the attitude angles. For example, if the drone experiences a roll perturbation, it is expected to maintain the new roll angle in the absence of corrective torque from U_2. This persistent tilt would, however, redirect the thrust vector, leading to lateral motion in the y-direction and a gradual descent in the z-direction as insufficient lift is being generated to maintain a hover. Thus, the system exhibits marginal stability in attitude as well, with coupled effects on position resulting from changes in orientation.

3.5.1 Nyquist Plots

For each of the six transfer functions, a Nyquist plot was generated in MATLAB using the code provided in Appendix H. The four inner-loop transfer functions (corresponding to z, ϕ, θ, and ψ) share identical structural forms, differing only by scalar coefficients associated with mass or moments of inertia. As a result, their Nyquist plots are identical in shape.

Similarly, the transfer functions for x and y follow the same structure as those for θ and ϕ, respectively, but represent fourth-order integrators rather than second-order integrators. This increase in order does not alter the fundamental nature of the Nyquist plots, as all poles remain located at the origin. When plotted over the frequency range $\omega = 0$ to ∞, all the transfer function magnitudes transition from infinity to zero, with the only difference being the rate at which they do so. The x and y responses decaying more rapidly than the attitude responses due to the higher order s term in the denominator. Therefore, only a single Nyquist Plot was generated which may be applied as analysis to all the transfer functions.

The resulting Nyquist plot for the uncontrolled system is shown in Figure 7 and was generated using the code in Appendix **??**. It was plotted in both the standard form and the log form.

In the standard representation of the Nyquist plot (left), the entire locus lies along the negative real axis. It does not encircle the critical point $-1 + 0j$, but rather makes tangential contact with it—a characteristic indicative of marginal stability.

To provide improved visual interpretation of this behaviour, consider the logarithmic Nyquist plot on the right. In this representation, the outer circle represents the point at infinity. It can be seen that

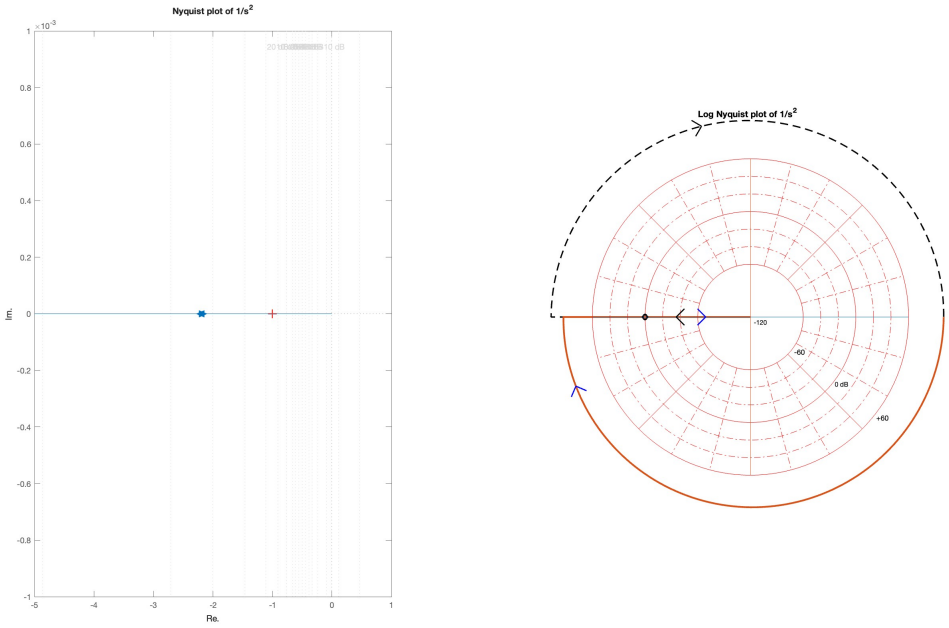


Figure 7: Nyquist Plots

the Nyquist plot sweeps from negative infinity to zero and back to negative infinity along the negative real axis.

3.5.2 Marginal Stability Demonstration in Linear and Non-Linear Systems

To further illustrate the marginal stability of the system, a time-domain analysis can be conducted through simulation. This approach not only confirms the marginal nature of the open-loop stability but also highlights the differences between the linear and nonlinear system responses. For this analysis, the drone is initially assumed to be in a hovering state. Perturbations are then introduced by modifying the six initial state conditions, allowing observation of the resulting system behaviour under both modelling approaches. Consider the Figures 8, 9 and 10 below.

The only difference that occurs is when either the pitch or roll angles are given a change in initial condition.

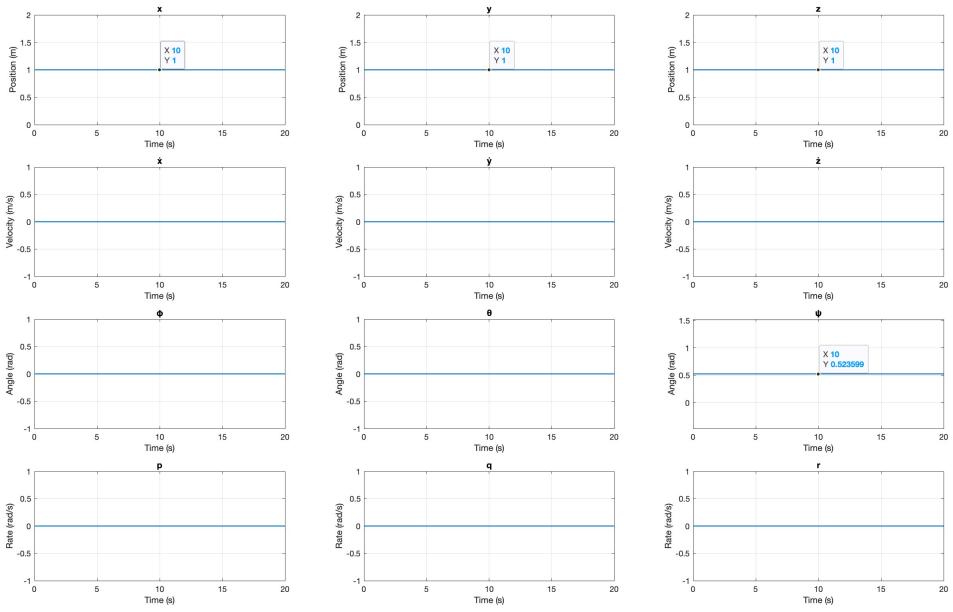


Figure 8: The response of both the linear and non-linear system to a change in initial condition of the x, y, z and ψ states.

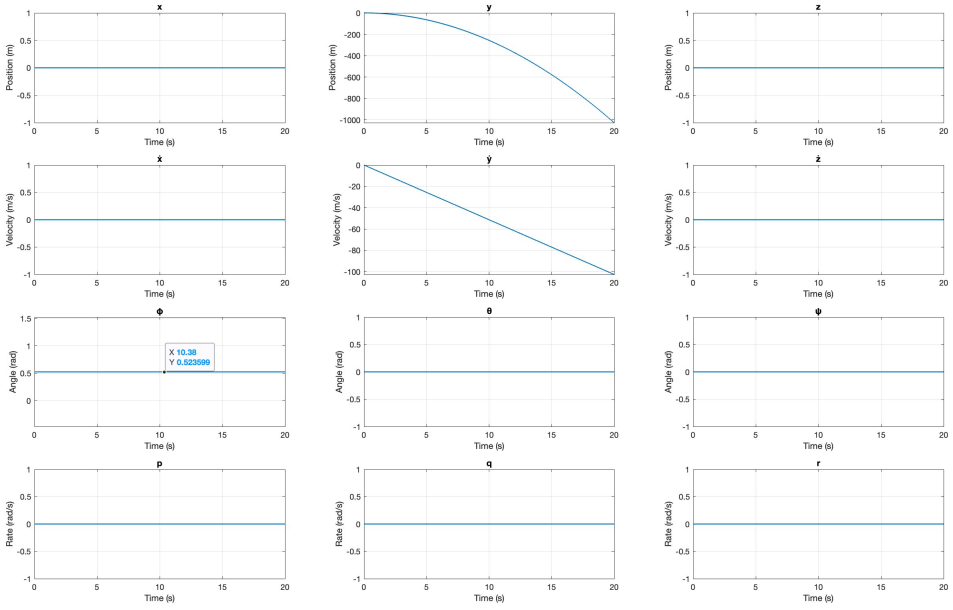


Figure 9: Response of the linear system to an initial roll angle of 30 degrees.

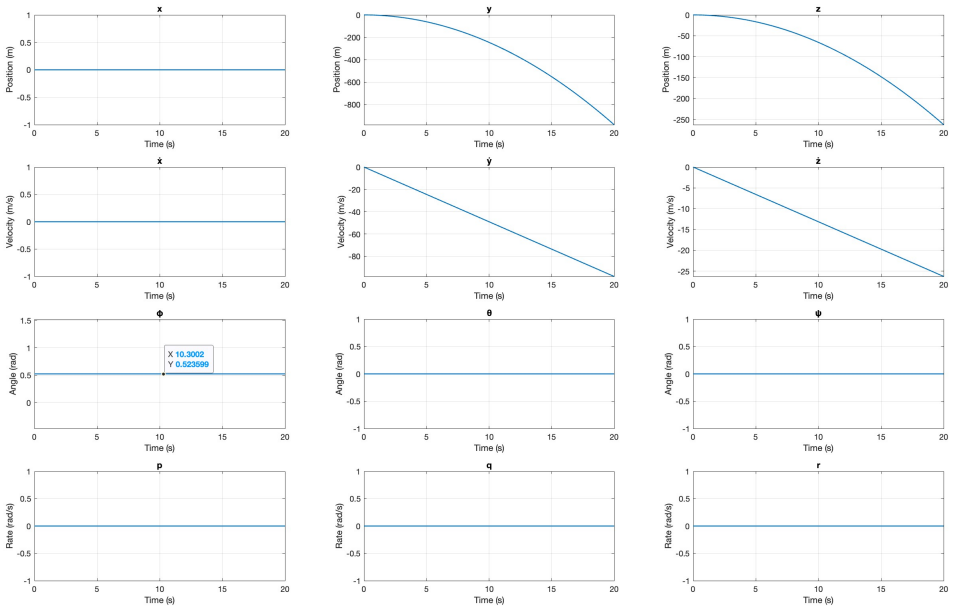


Figure 10: Response of the non-linear system to an initial roll angle of 30 degrees.

No deviation exists between the linear and nonlinear simulations when initial conditions are altered in the x, y, z, or ψ states. This is consistent with the absence of inherent coupling between these states. Such perturbations effectively translate the drone to a new position or rotate it about the z-axis, while maintaining a hovering state. These results demonstrate that the system does not exhibit any natural restoring action to return to the original equilibrium point, yet continues to hover in the new state.

Consider the affect of an initital perturbation of 30 degrees in ϕ as seen in Figures 9 and 10, the linear and non-linear responses are not identical. Note that this was only plotted for ϕ but can easily be extended to changes in the initial state of θ.

In both cases, the roll angle remains constant over time, consistent with the earlier marginal stability analysis. This behaviour confirms that, in the absence of active control input through U_2, no restoring torque is generated to return the drone to its original orientation. The system maintains the perturbed state without exhibiting corrective dynamics.

In both simulations, the y position exhibits a change over time, which is consistent with the physical expectation that tilting the drone alters the direction of the thrust vector, introducing a component in the y-direction. The linear model shows a faster acceleration in the y-direction, primarily due to the breakdown of the small-angle assumption at a perturbation of 30 degrees. While the nonlinear model experiences an acceleration of $\ddot{y} = g\sin(\phi)$, the linear model incorrectly approximates this as $\ddot{y} = g\phi$, which overestimates the acceleration at larger angles, resulting in the observed discrepancy.

An effect entirely missed by the linear simulation (due to the removal of the coupling between z and ϕ in the linearisation) is the reduction in vertical position that occurs as the drone is rotated. As the orientation changes, the thrust U_1, which was previously aligned to exactly counteract gravity in the hover state, is now partially directed along the y-axis. Since the magnitude of U_1 remains constant, the component acting in the z direction is reduced, resulting in insufficient lift to balance the gravitational force. Consequently, the drone begins to descend.

In this uncontrolled scenario, no restoring torque is present to return the drone to its equilibrium orientation, nor is there additional thrust introduced to maintain a stable hover.

3.6 Requierment for Control

The underlying dynamics of the drone system are marginally stable, meaning that in the absence of control, the system neither converges to an equilibrium state nor diverges, but instead maintains persistent oscillations or drifts under disturbance. This behaviour is unsuitable for practical operation, particularly in environments requiring precise positioning and sustained hovering, such as manoeuvring within a server room. To ensure the drone can maintain a stable hover and respond accurately to position and attitude commands, active control must be introduced. The control system provides the necessary stabilising feedback to suppress natural instabilities and achieve the level of performance required for safe and reliable navigation in constrained environments.

4 Controller Implementation

As the system is under-actuated, a cascaded control architecture is required to achieve full-state regulation across all six degrees of freedom. This structure comprises two nested control loops: an inner loop and an outer loop.

The inner loop is responsible for regulating the attitude angles (ϕ, θ, ψ) and the vertical position (z). These states can be controlled directly, owing to their clear one-to-one correspondence with the control inputs. U_1 controls z, U_2 controls ϕ, U_3 controls θ, and U_4 controls ψ, as previously discussed.

In contrast, the horizontal positions x and y are indirectly controlled through reorientation of the thrust vector, which is achieved by adjusting θ and ϕ via U_3 and U_2, respectively. Therefore, control of x and y is implemented in the outer loop. These controllers generate reference commands for the inner-loop attitude controllers, effectively setting the desired values of θ and ϕ required to achieve the target x and y positions. Since the yaw angle, ψ, has no affect on the x and y translation, it can be directly commanded to some desired angle.

When the drone reaches the commanded x and y positions, the output of the outer-loop controllers becomes zero, driving the reference angles for θ and ϕ to zero. This ensures that the drone stabilises with a level orientation.

Accordingly, the inner loops are designed with a higher bandwidth than the outer loops, ensuring they can promptly track the rapid attitude and altitude reference commands generated by the slower position controllers.

A schematic representation of this cascaded control structure is provided in the block diagram shown in Figure 11 [11].

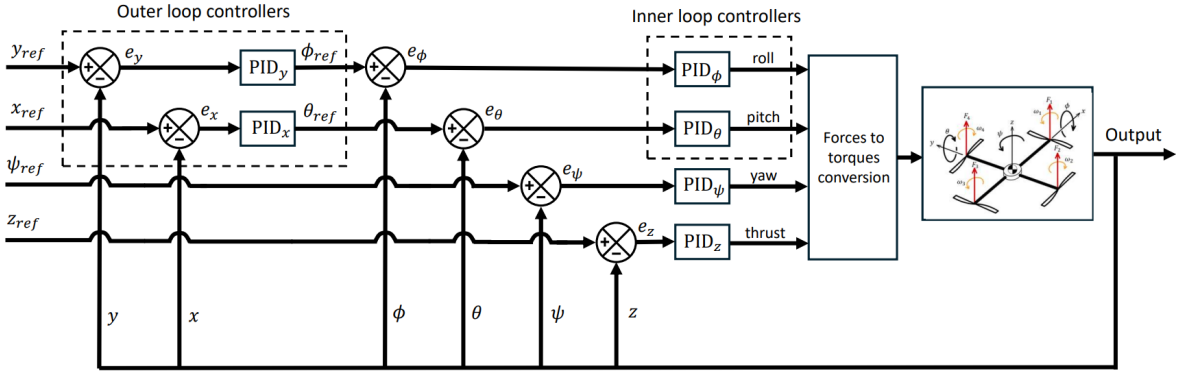


Figure 11: Cascaded control structure for quadrotor [11].

The control system design proceeds by first addressing the inner loops, which are tuned independently of the outer loops. Once satisfactory performance of the inner loops is achieved, the outer-loop controllers are then designed around the established inner-loop dynamics.

Prior to undertaking the controller design, it is necessary to define the system specifications around which the controller gains will be tuned. These specifications — settling time and percentage overshoot - were selected based on the operational requirements of the data centre environment. The corresponding values of ζ and ω_n for each of the four controllers were then calculated using the following equations.

$$\zeta = \sqrt{\frac{\left(\ln\left(\frac{\%OS}{100}\right)\right)^2}{\pi^2 + \left(\ln\left(\frac{\%OS}{100}\right)\right)^2}} \tag{7a}$$

$$\omega_n = \frac{4}{\zeta T_s} \tag{7b}$$

The summary of this process can be seen in Table 5 below.

Table 5: System Specifications and Calculated Parameters

State	$\%OS$	T_s	ζ	ω_n
z	10	5	0.5911	1.353
ϕ	10	2	0.5911	3.3835
θ	10	2	0.5911	3.3835
ψ	10	2	0.5911	3.3835

4.1 Inner Loops

All inner-loop transfer functions take the canonical form $G(s) = \frac{1}{Ms^2}$, where M represents either the mass or the relevant moment of inertia associated with each control channel, as detailed in Table 3. Consequently, the controller design methodology is consistent across all inner-loop channels; however, the resulting controller parameters will vary according to the specific dynamic properties and performance requirements of each channel.

To illustrate the design approach, a general procedure will be presented based on a transfer function of the form $G(s) = \frac{1}{Ms^2}$, with assumed defined percentage overshoot and settling time specifications. Any deviations required due to the unique characteristics of individual control channels will be

addressed as they arise. Nevertheless, the initial design methodology remains consistent across all inner-loop controllers.

Figure 12 shows the closed-loop configuration used throughout the inner-loop design. The reference command $R(s)$ is compared to the plant output $Y(s)$ to form the error $E(s)$. That error is processed by the controller, $C(s)$, whose output $U(s)$ drives the plant, $G(s) = \frac{1}{Ms^2}$. The plant output is then fed back (with unity gain) to complete the loop.

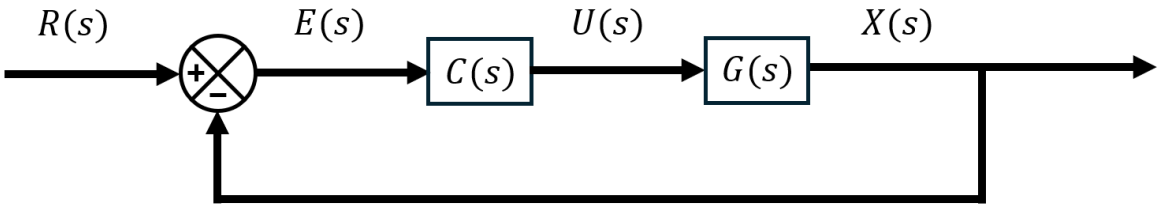


Figure 12: A representation of the inner control loop as a block diagram.

4.1.1 Proportional Control

Consider first the application of proportional control to the system, such that $C(s) = K_p$. Closing the loop yields the following closed-loop transfer function:

$$\frac{Y(s)}{R(s)} = \frac{K_P}{Ms^2 + K_p} \tag{8}$$

By inspection, it is evident that this system cannot achieve asymptotic stability. The characteristic equation, $Ms^2+K_P = 0$, lacks a first-order s term, and therefore immediately fails the Routh–Hurwitz criterion. As a result, proportional control alone is insufficient to stabilise the system.

This limitation is further illustrated by the root locus of the open-loop transfer function $C(s)G(s)$, shown in Figure 13. This was plotted with the code in Appendix I The root locus plot depicts the trajectories of the closed-loop poles as K_P varies. It is clear that the poles move along the imaginary axis but never enter the left-half plane, indicating that the system remains marginally stable for all values of K_P. This behaviour is characteristic of a double integrator system under pure proportional control. Therefore, it is necessary to consider Proportional and Integral control.

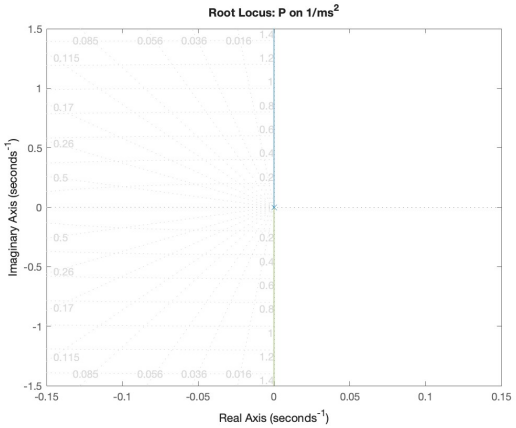


Figure 13: Root locus of the plant $\frac{1}{ms^2}$ under proportional control

4.1.2 Proportional and Integral Control

Applying the same analytical method as before, but using $C(s) = K_P + \frac{K_I}{s}$, the closed loop transfer function becomes:

$$\frac{Y(s)}{R(s)} = \frac{K_P s + K_I}{Ms^3 + K_P s + K_I} \tag{9}$$

As with the proportional case, the system once again fails the Routh–Hurwitz criterion for stability due to the absence of the s^2 term in the characteristic equation. Consequently, this configuration can be immediately ruled out as it cannot yield an asymptotically stable response. This conclusion is once again further supported by the corresponding root locus, which offers additional insight into the system's dynamic behaviour.

The root locus method requires that only a single gain parameter be varied at a time. However, in this case, the transfer function contains two gain terms, K_P and K_I, necessitating some reformulation.

Starting with the open loop transfer function:

$$C(s)G(s) = \frac{K_P s + K_I}{Ms^3} \tag{10}$$

Factoring out K_P from the numerator yields:

$$C(s)G(s) = K_P \cdot \frac{s + \frac{K_I}{K_P}}{Ms^3} \tag{11}$$

This form introduces a zero into the open-loop transfer function at the location $s = -\frac{K_I}{K_P} + 0j$ on the complex plane. As such, by selecting a desired zero location, the ratio $\frac{K_I}{K_P}$ can be determined accordingly, allowing for systematic controller design through root locus analysis. The location of the new zero was set at $5\omega_n$ to the left of the origin which ensures it does not dominate the response. This allowed for the plotting of the root locus seen in Figure 14 seen below. Since the controllers all follows the same form, this was plotted just for the U_1 controller to demonstrate the shape of the root locus.

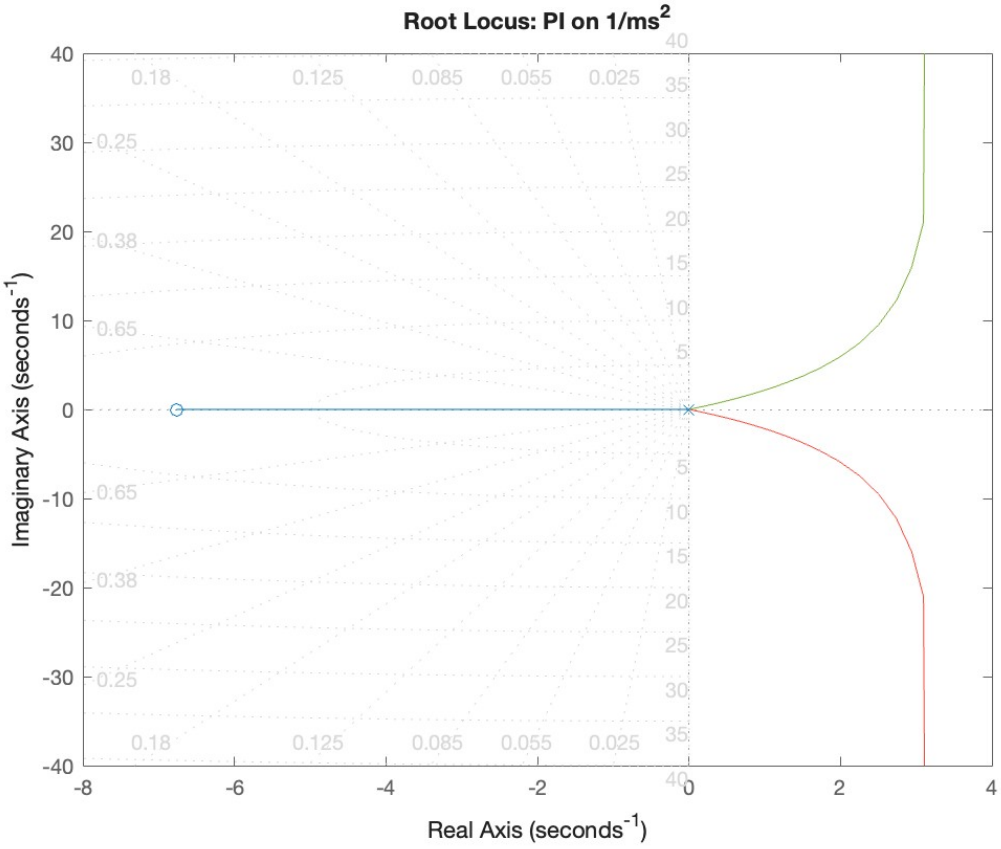


Figure 14: Root locus of the plant $\frac{1}{ms^2}$ under PI control

This root locus closely resembles that shown in Figure 14, with the exception that the branches have shifted slightly off the imaginary axis and into the right-half plane and that there is now the additional zero in the left hand plane. This indicates that, unlike the proportional controller which produced marginal stability for all gain values, the inclusion of integral action has rendered the system asymptotically unstable for any combination of K_P and K_I, confirming the prediction by the Routh–Hurwitz criterion. Consequently, proportional-derivative (PD) control should be tested next to attempt to provide stability.

4.1.3 Proportional and Derivative Control

Considering a PD controller, where $C(s) = K_D s + K_P$, the resulting closed-loop transfer function is:

$$\frac{Y(s)}{R(s)} = \frac{K_P + K_D s}{M s^2 + K_D s + K_P} \tag{12}$$

The corresponding characteristic equation, $Ms^2 + K_D s + K_P$, satisfies the Routh–Hurwitz criterion, indicating the potential for an asymptotically stable system. To determine appropriate values for K_P and K_D, the coefficient matching method was used.

By dividing the characteristic equation by M, it can be expressed in the standard second-order form:

$$s^2 + \frac{K_D}{M} s + \frac{K_P}{M} \tag{13}$$

Matching this to the canonical form $s^2 + 2\zeta\omega_n s + \omega_n^2$ allows the controller gains to be expressed in terms of the damping ratio ζ and natural frequency ω_n:

$$K_P = M\omega_n^2 \tag{14a}$$

$$K_D = 2M\zeta\omega_n \tag{14b}$$

The values of ζ and ω_n have already been selected to meet desired settling time and percentage overshoot specifications and therefore, the gains can be determined.

This tuning procedure was applied to each inner-loop controller, and the resulting gain values are summarised in Table 6.

Table 6: PD Controller Gains for Inner-Loop States

Controller	K_P	K_D
U_1	0.9263	0.8094
U_2	9.304×10^{-4}	0.3249
U_3	9.304×10^{-4}	0.3249
U_4	1.121×10^{-4}	0.0979

PD Controller Stability

Before evaluating the performance of the controllers using the linear and non-linear models, the quality of the stability of the associated transfer functions can be assessed using root locus analysis.

Consider again the open-loop transfer function $C(s)G(s) = \frac{K_D s + K_P}{Ms^2}$. As with the root locus analysis conducted for the PI controller, one of the gains must be factored out to enable variation of a single parameter when generating the root locus plot. Once again, K_P can be factored yielding:

$$C(s)G(s) = K_P \frac{1 + \frac{K_D}{K_P} s}{Ms^2} \tag{15}$$

To proceed, the ratio $\frac{K_D}{K_P}$ must be specified. This time, it can be readily obtained from the design equations in (14), yielding:

$$\frac{K_D}{K_P} = \frac{2\zeta}{\omega_n} \tag{16}$$

With this formulation established, the root locus can be generated for each of the inner-loop transfer functions. However, as all share the same structural form, the resulting root loci will exhibit identical general behaviour. Accordingly, the root locus is presented for the U_1 controller only, as shown in Figure 15, all conclusions drawn from this plot can be readily extended to any of the inner loop transfer functions.

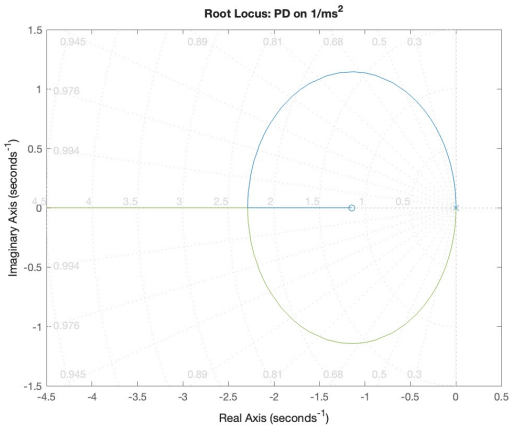


Figure 15: Root locus of the plant $\frac{1}{ms^2}$ under PD control

The root locus plots confirm that for any positive gain K_P, and consequently a positive K_D, the system remains asymptotically stable, consistent with initial predictions. The two branches originate from poles at the origin and migrate into the left-half plane, rejoining along the negative real axis beyond the location of the additional zero at $-\frac{K_P}{K_D} + 0j$. One branch approaches the zero, while the

other continues towards negative infinity as the gain increases. Since no part of the root locus enters the right hand plane, this plant is evidently stable under PD control for any gain value.

To further validate the stability of the controller, a Nyquist plot was generated using the open-loop transfer function with the final tuned gain values. The resulting Nyquist plot is presented in Figure 16.

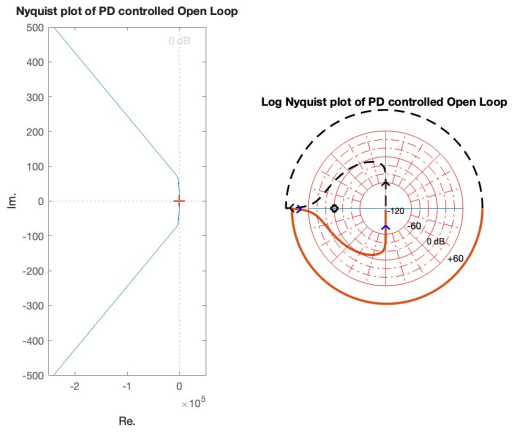


Figure 16: Nyquist plot of the $\frac{1}{ms^2}$ plant under PD control

The standard Nyquist plot (left) does not provide sufficient visual clarity; therefore, a logarithmic Nyquist plot (right) is included to enhance interpretability. These plots reveal that, unlike the uncontrolled system, the Nyquist locus no longer lies along the real axis. It no longer makes contact with the critical point $-1 + 0j$, as evident in the standard plot. As shown more clearly in the logarithmic plot, the locus departs from the origin, moves into the left-half plane, traverses the outer circle corresponding to infinity and returns from negative infinity in the left-half plane. The absence of any encirclement of the critical point, -1, confirms that the system is stable under the selected gain values.

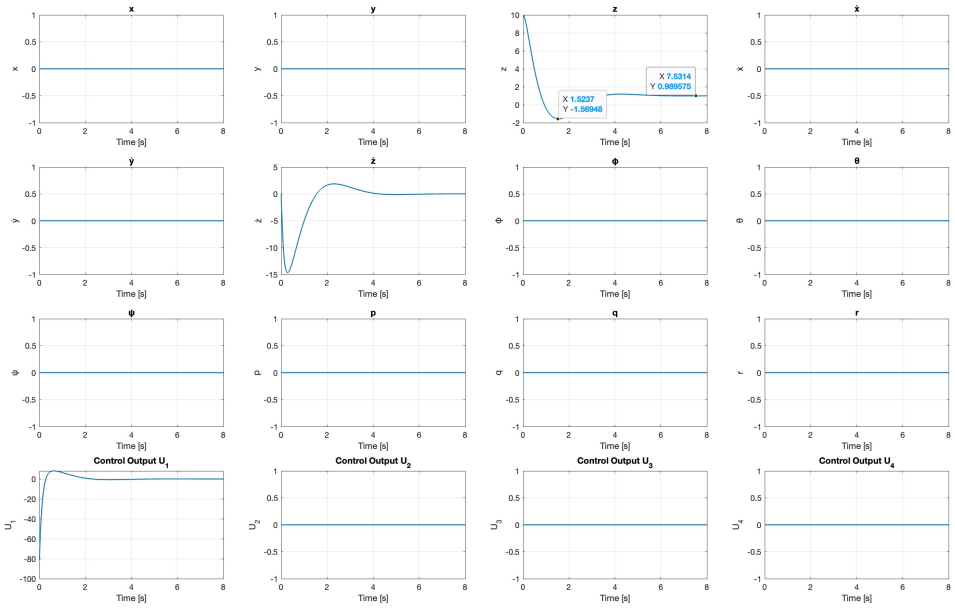
PD Controller Performance

To demonstrate the performance of the controller and thus determine the validity thereof, the responses of both the linear and nonlinear models were evaluated under the three test conditions defined as objectives for this project: free-hand throw, free fall, and inverted (upside-down) initial orientation. The initial conditions corresponding to each of these scenarios can be seen in Table 7 below.

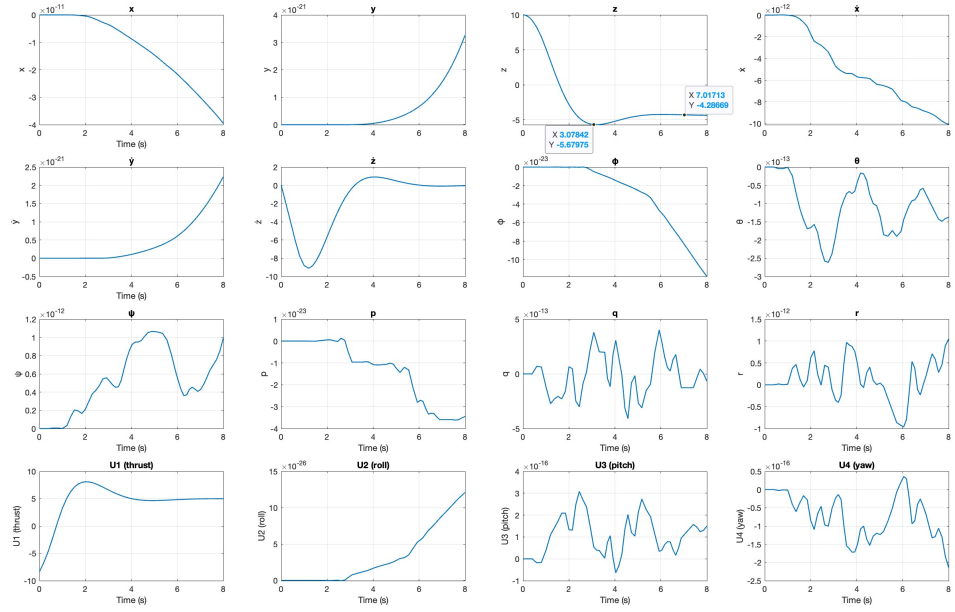
Table 7: Initial Conditions for Different Simulation Scenarios

State Variable	Free Throw	Free Fall	Inverted Orientation
x (m)	0	0	0
y (m)	0	0	0
z (m)	0	10	0
$\dot{x}$ (m/s)	1	0	0
$\dot{y}$ (m/s)	1	0	0
$\dot{z}$ (m/s)	1	0	0
ϕ (radians)	0.52	0	π
θ (radians)	0.52	0	0
ψ (radians)	0.52	0	0
$\dot{\phi}$ (radians/s)	1	0	0
$\dot{\theta}$ (radians/s)	1	0	0
$\dot{\psi}$ (radians/s)	1	0	0

The simulations were conducted using the PD gains derived for the z, ϕ, θ, and ψ control channels, as outlined above and the code used can be found in Appendecies J and G. The drone was commanded to stabilise at a position of $(\phi, \theta, \psi, z) = (0, 0, 0, 1)$. The results of these simulations can be seen in Figures 17, 18 and 19 below.

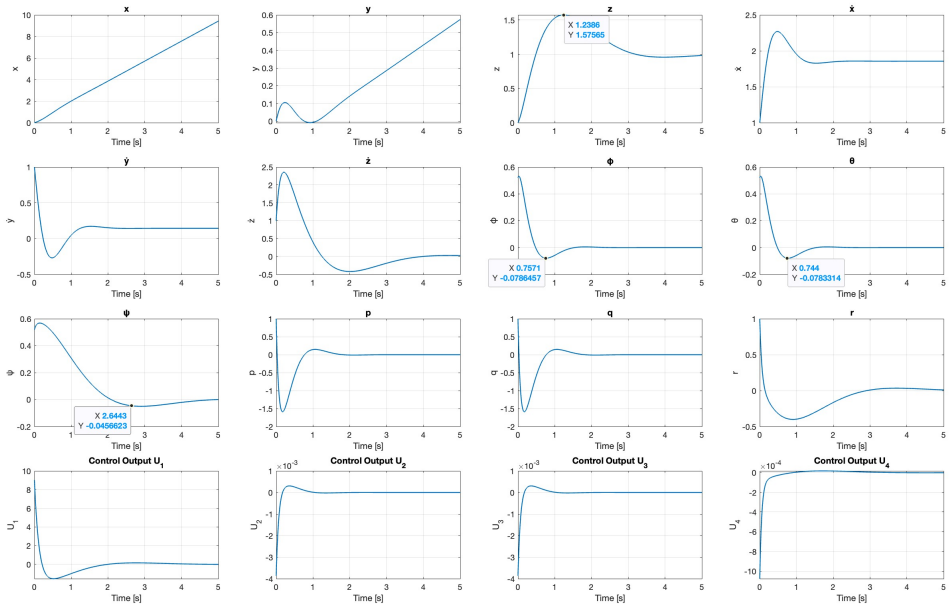


(a)

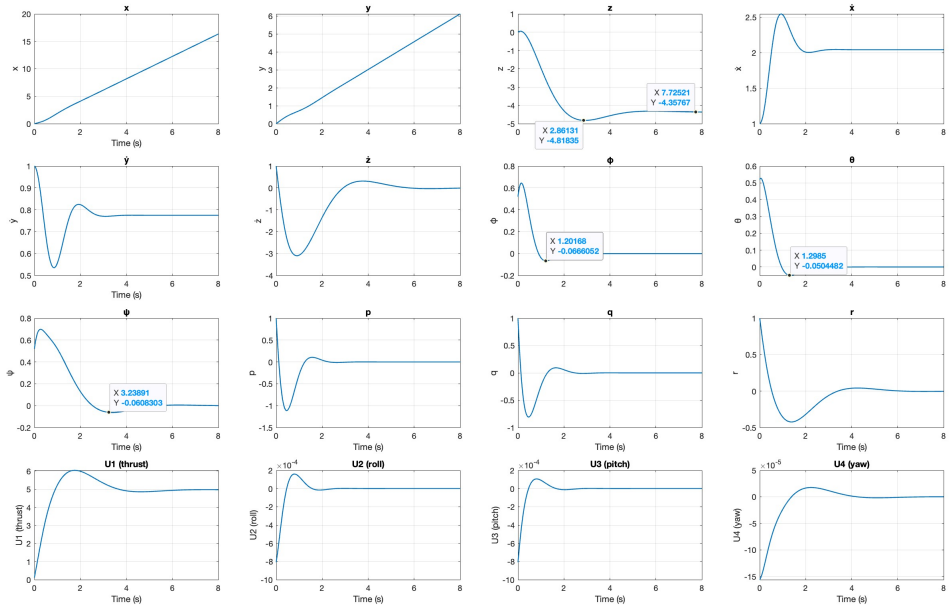


(b)

Figure 17: System Response (a) Linear and (b) Non-Linear to Free Fall for PD Control

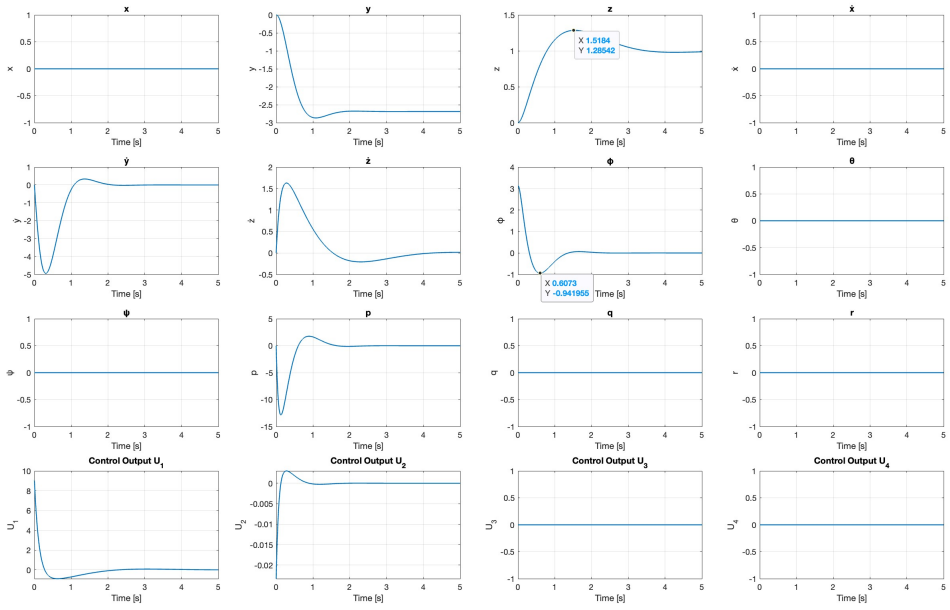


(a)

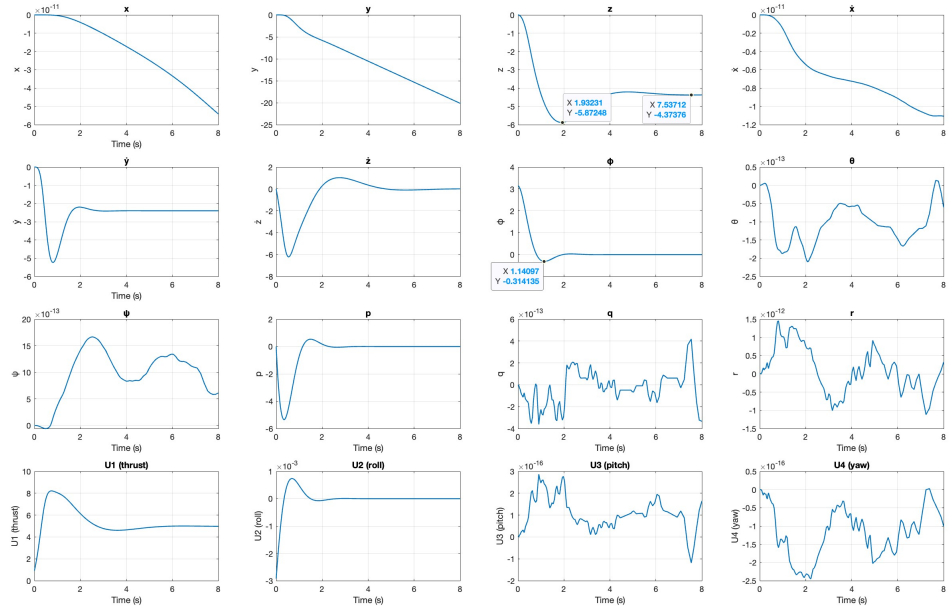


(b)

Figure 18: System Response (a) Linear and (b) Non-Linear to Free Throw for PD Control



(a)



(b)

Figure 19: System Response (a) Linear and (b) Non-Linear to an inverted Orientation for PD Control

The first glaring difference in these two outputs is that the nonlinear free-fall and inverted initial conditions simulations exhibit apparent steady state instability with unsettled perturbations. However, closer examination reveals that these perturbations are on the order of 10^{-12} to 10^{-23}, representing deviations that are negligible in practical terms and would be imperceptible in any physical implementation.

To illustrate the insignificance of such variations, consider the following analogy: if a 1 metre rod were rotated through an angle of 10^{-12} radians, the lateral displacement at its tip would be 1 pico metre. This distance lies at the scale of atomic dimensions and is therefore physically negligible.

Despite the negligible magnitude, it remains important to consider the possible origin of these disturbances. One plausible explanation is that they arise from numerical artefacts rather than genuine system behaviour. The nonlinear simulations were performed using MATLAB's variable-step ODE45 solver, which dynamically adjusts the step size to satisfy specified error tolerances - typically on the order of 10^{-6} by default [12]. Even after the system settles into an equilibrium state, finite-precision arithmetic can produce residual numerical noise. This may result in the PID controller issuing vanishingly small corrective commands, which in turn perturb the state by a comparable magnitude.

These perturbations may be amplified or sustained by coupling effects in the nonlinear dynamics - particularly between ϕ and y, θ and x, and also between ψ and the other attitude angles. While the controller would ideally suppress such errors, the persistence of these disturbances may indicate minor sensitivity in the control design.

Furthermore, increasing the solver tolerance did not eliminate the perturbations, suggesting that the issue may not be purely numerical. It is therefore possible that the residual oscillations stem from characteristics of the controller itself. Nevertheless, due to their extremely small magnitude, these deviations are deemed practically insignificant.

Although the signal noise appears to affect only certain states, while others seem unaffected, it is important to note that the states displaying no apparent noise were subjected to initial disturbances. As a result, the scale of their corresponding plots is several orders of magnitude larger than the noise itself, causing the noise to be visually negligible. This reinforces the insignificance of the observed signal noise, as it does not compromise the steady-state stability demonstrated in the states subject to initial conditions.

The linear model does not exhibit this behaviour, as it lacks both the dynamic couplings present in the non-linear model and the numerical sensitivity introduced by variable-step integration.

In terms of the simulation results, it is observed that the percentage overshoot is consistently greater in the nonlinear model compared to the linear model across all test cases. This outcome is expected, as the linear model is based on the assumption of a decoupled hovering plant with small-angle

approximations. The controller gains were designed under these linearised conditions and applied identically in both models.

While the restoring torques generated by the controller are theoretically the same - being determined solely by the gain values - the non-linear model captures the full coupling between the attitude angles and their associated dynamic interactions. These couplings introduce additional complexity and interaction effects that are not accounted for in the linear model, thereby making the system more difficult to control. This behaviour aligns with expectations when transitioning from a simplified linear representation to a more accurate non-linear model, which is expected to be more difficult to control.

Another notable observation is the persistent steady-state error present in all non-linear simulations, with the z-position stabilising at approximately -4.3m. This represents a substantial deviation from the commanded reference of 1m and highlights a significant deficiency in altitude tracking. Despite this, the attitude angles in the nonlinear simulations converge correctly to their respective reference values, it is just the altitude that has this error.

The most pronounced example of this error is observed in Figure 18. In this scenario, the drone begins at a height of 0m with an initial upward velocity. The expected behaviour is for the drone to continue ascending and stabilise at the reference altitude of 1m, as demonstrated in the linear model. However, in the nonlinear simulation, the controller input U_1 applies a corrective action in the downward direction, accelerating the drone such that it ultimately stabilises at approximately -4.3m, the steady stater error here being -5.3m. In contrast, the linear system has no steady state error for any of the responses.

The steady state error is likely attributable to the lack of integral action in the current controller, which may be insufficient to eliminate constant offsets arising from non-linear effects while the PD control is sufficient for the linearised model. The incorporation of integral control will be considered in the following section as a potential solution to address this issue.

Another significant observation is the difference in the initial magnitude of the controller outputs between the linear and nonlinear simulations. In the linear system, the controller outputs are considerably larger at $t = 0$, often by an order of magnitude, before rapidly decreasing. This behaviour is attributed primarily to the derivative component of the controller.

At the instant a step error is introduced such as at $t = 0$, the rate of change of error approaches infinity, leading to a theoretically unbounded response from the derivative term. To prevent this, the derivative component is implemented with a first-order low-pass filter, represented as

$$D(s) = \frac{K_D s}{\tau s + 1} \tag{17}$$

This filter attenuates the derivative spike by limiting the gain at high frequencies. Increasing the value of τ reduces the initial spike; however, as τ increases, the asymptotic gain $\frac{K_D}{\tau}$ falls and the derivative term becomes less influential, thus the controller begins to behave like a pure proportional controller.

The parameter τ was therefore selected through an iterative process to strike a balance: minimising the initial spike in control effort while retaining meaningful derivative behaviour. The final values used for τ can be seen in the code in Appendix J.

4.1.4 Proportional Integral Derivative Control

Although the PD controller offers a degree of stability, the implementation of a full PID controller may provide enhanced stability characteristics, specifically, the integrator helps to eliminate steady state error for step inputs. Therefore, the design of a PID controller is also considered. Following a similar methodology to that used previously, the PID controller is defined as:

$$C(s) = K_P + K_D s + \frac{K_I}{s} \tag{18}$$

This results in the following characteristic equation in the closed-loop system:

$$s^3 + \frac{K_D}{M} s^2 + \frac{K_P}{M} s + \frac{K_I}{M} \tag{19}$$

This expression closely resembles the PD characteristic equation, with the addition of an integrator term that introduces a third pole. To facilitate gain selection, the characteristic equation is matched to a standard third-order form, given by:

$$(s^2 + 2\zeta\omega_n s + \omega_n^2)(s + P) \tag{20}$$

Expanding this expression yields:

$$s^3 + (2\zeta\omega_n + P)s^2 + (2\zeta\omega_n P + \omega_n^2)s + \omega_n^2 P \tag{21}$$

By equating the coefficients of like powers of s in Equations 20 and 21, the controller gains can be derived as follows:

$$K_P = M(2\zeta\omega_n P + \omega_n^2) \tag{22a}$$

$$K_I = M(\omega_n^2 P) \tag{22b}$$

$$K_D = M(2\zeta\omega_n + P) \tag{22c}$$

While the values of ζ and ω_n are determined by performance specifications as was done in the previous controller design, the placement of the additional pole, P, must also be specified. The position of P significantly affects system stability. A negative value of P introduces a pole at $s = |P|$, in the right hand plane, which is unstable. Thus, P must be positive to ensure the additional pole lies in the left-half plane, thereby contributing positively to stability.

The magnitude of P influences the dominance of the pole in the system dynamics. A smaller value places the pole closer to the origin, increasing its influence. Conversely, selecting a larger P reduces its impact but simultaneously increases the magnitude of the controller gains. Excessively large gain values can lead to impractical or unstable implementations. This can amplify measurement noise as well, deleteriously affecting stability.

To balance stability and gain magnitude, P is selected to lie at a distance of $5\omega_n$ from the origin on the negative real axis. This choice ensures the additional pole remains sufficiently fast to minimise its effect on the dominant dynamics, while maintaining reasonable gain values.

As the value of P has been determined, the gain values can be specified purely in terms of ζ and ω_n as:

$$K_P = M\omega_n^2(10\zeta + 1) \tag{23a}$$

$$K_I = M(5\omega_n^3) \tag{23b}$$

$$K_D = M\omega_n(2\zeta + 5) \tag{23c}$$

Based on the system specifications, the gains can now be specified for each of the controllers in the Table 8 below:

Table 8: PID Controller Gains for Inner-Loop States

Controller	K_P	K_I	K_D
U_1	6.4016	6.2663	4.2324
U_2	0.0064	0.0157	0.0017
U_3	0.0064	0.0157	0.0017
U_4	0.0048	0.0119	0.0013

PID Controller Stability

Once again, it is instructive to plot the root locus for this controller. Following a similar process as for the PI and PD controllers, the open loop transfer function $C(s)G(s)$ is factorised to yield:

$$C(s)G(S) = K_p \frac{\frac{K_D}{K_P}s^2 + s + \frac{K_I}{K_P}}{Ms^2} \tag{24}$$

Once again, the value of the ratios $\frac{K_D}{K_P}$ and $\frac{K_I}{K_P}$ can be found directly from the system parameters and the root locus can be plotted by varrying K_P. This results on the following plot see in Figure 20 below. This root locus was plotted for the U_1 controller.

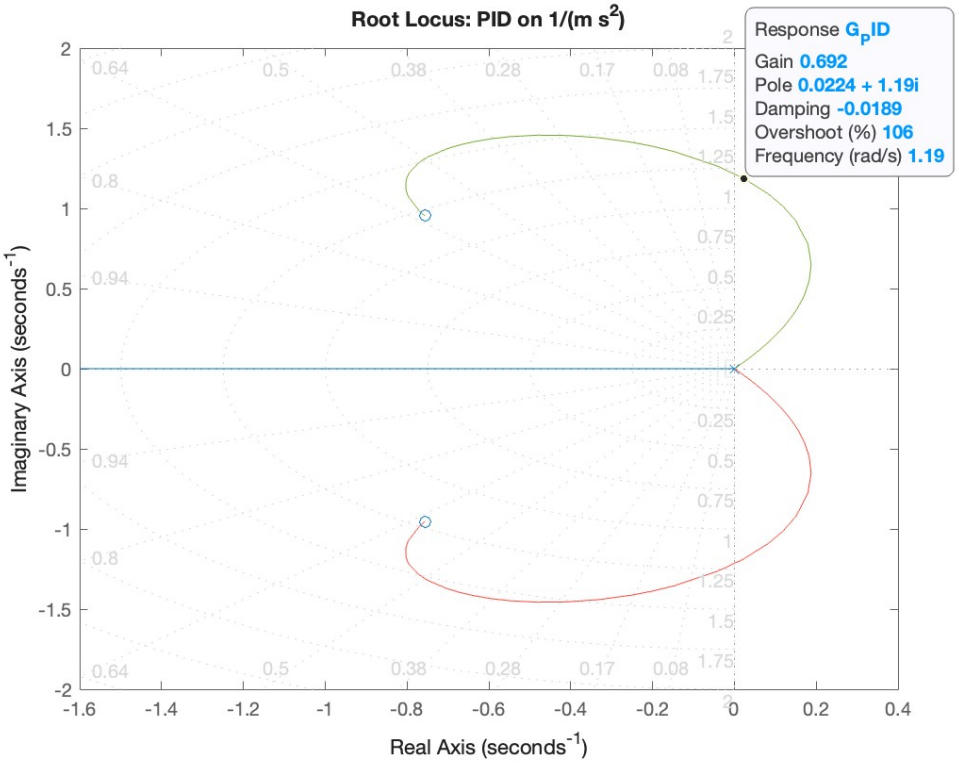


Figure 20: Nyquist plot of the $\frac{1}{ms^2}$ plant under PD control

The root locus plot now exhibits three branches, as expected from the third-order characteristic equation. Two of these branches initially extend into the right-half plane before curving back into the left-half plane. This behaviour indicates that not all combinations of PID controller gains will yield a stable system. Specifically, any design employing a proportional gain below approximately 0.7 - the crossover gain into the left half plane - will result in instability. The controller designed in this case utilises gains significantly above this threshold, thereby confirming system stability. Furthermore, this indicates that the system remains stable across a range of gain values in the vicinity of the selected design point.

Considering the Nyquist plots shown below for the U_1 controller in Figure 21, the response is qualitatively similar to that observed under PD control. The critical point at $-1 + 0j$ is not encircled, indicating closed-loop stability. The overall behaviour of the plot remains consistent; however, the inclusion of integral action results in a noticeably steeper trajectory of the locus as it departs from the origin.

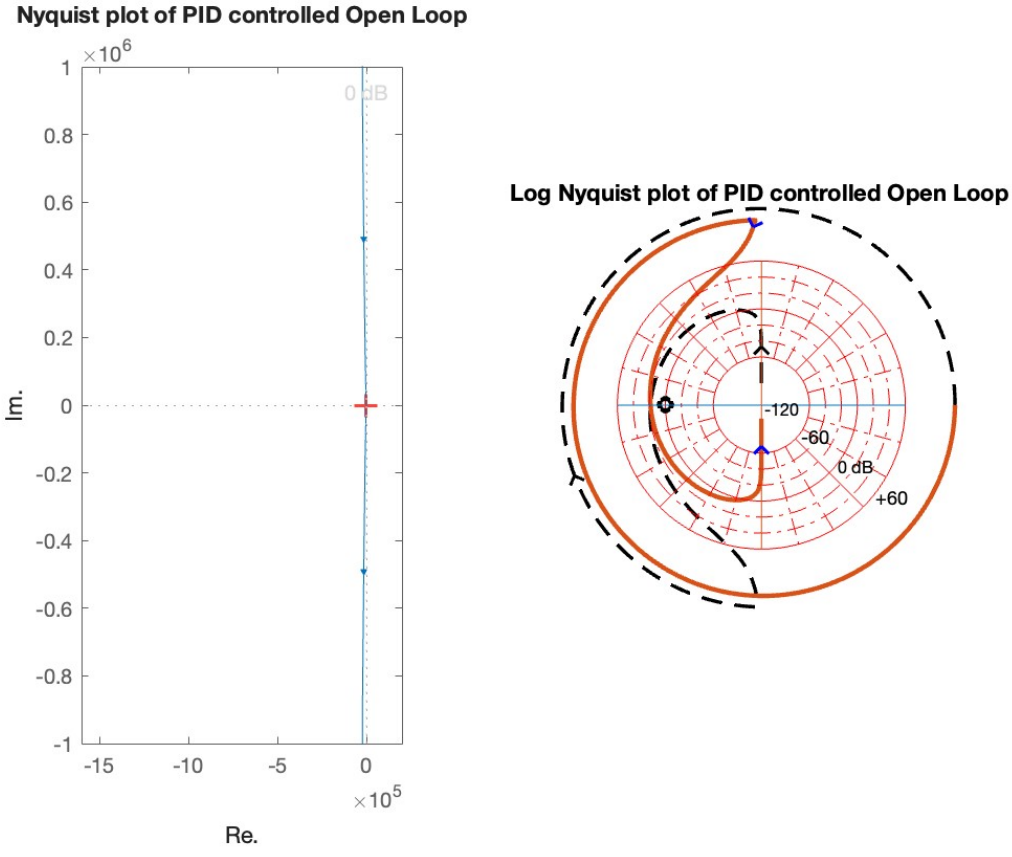
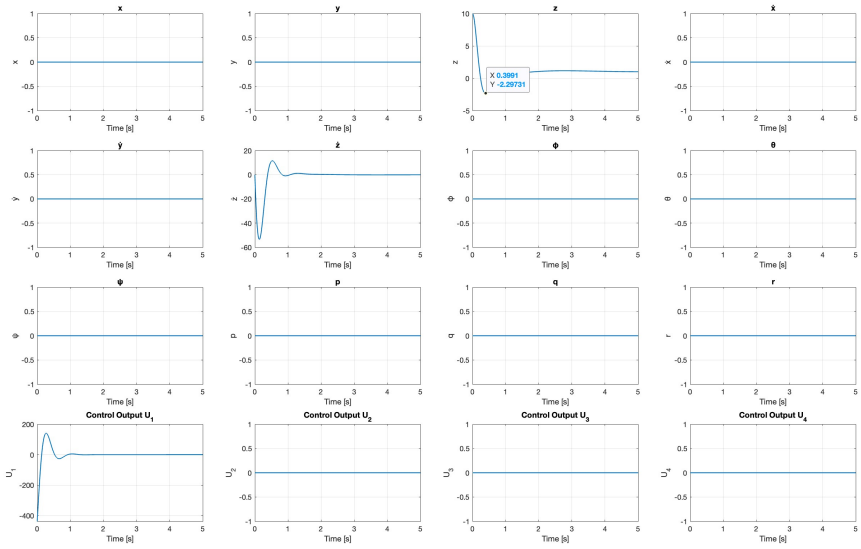


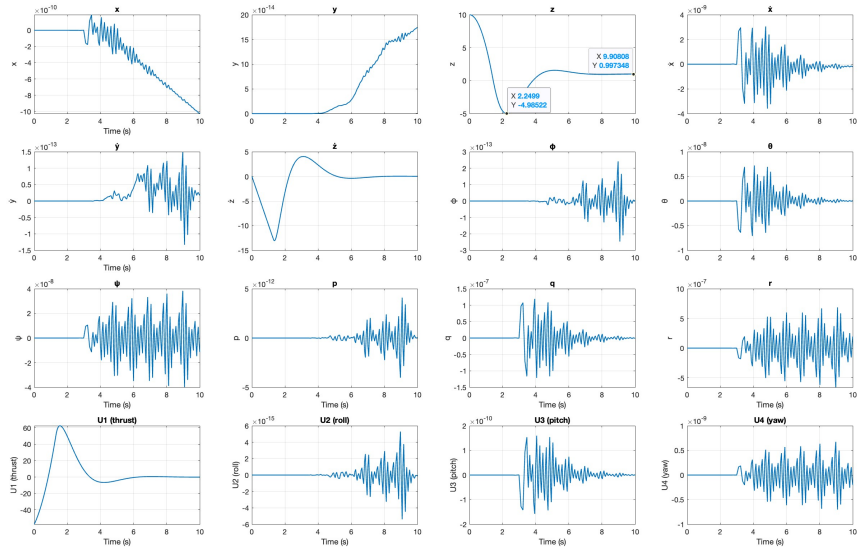
Figure 21: Nyquist plot of the $\frac{1}{ms^2}$ plant under PD control

PID Controller Performance

Using the calculated PID gains, both the linear and nonlinear models were simulated to evaluate the system response under the previously defined initial conditions: free fall, inverted orientation, and free-hand throw. The corresponding responses are presented in the Figures 22, 23 and 24 below.

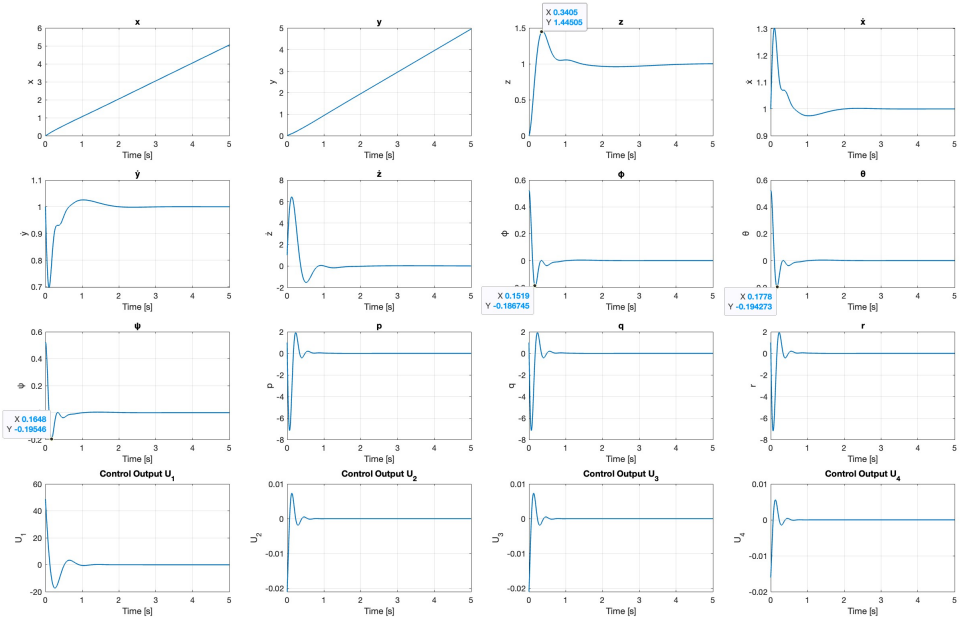


(a)

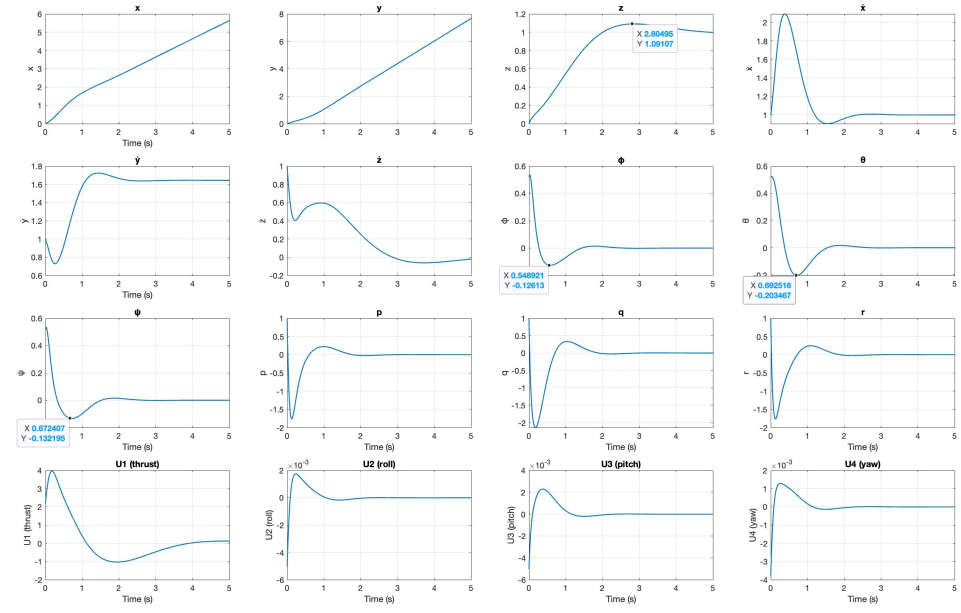


(b)

Figure 22: System Response (a) Linear and (b) Non-Linear to Free Fall for PID Control

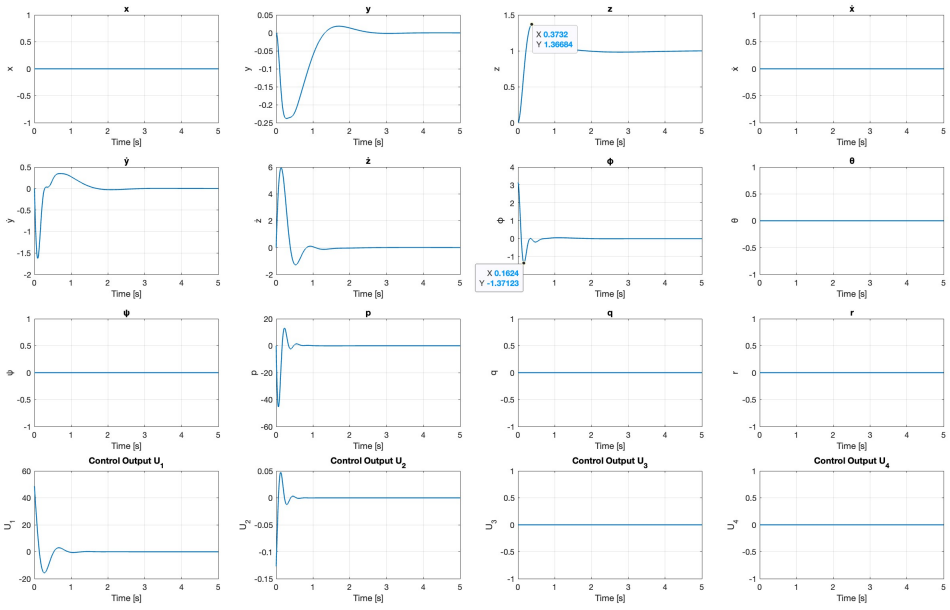


(a)

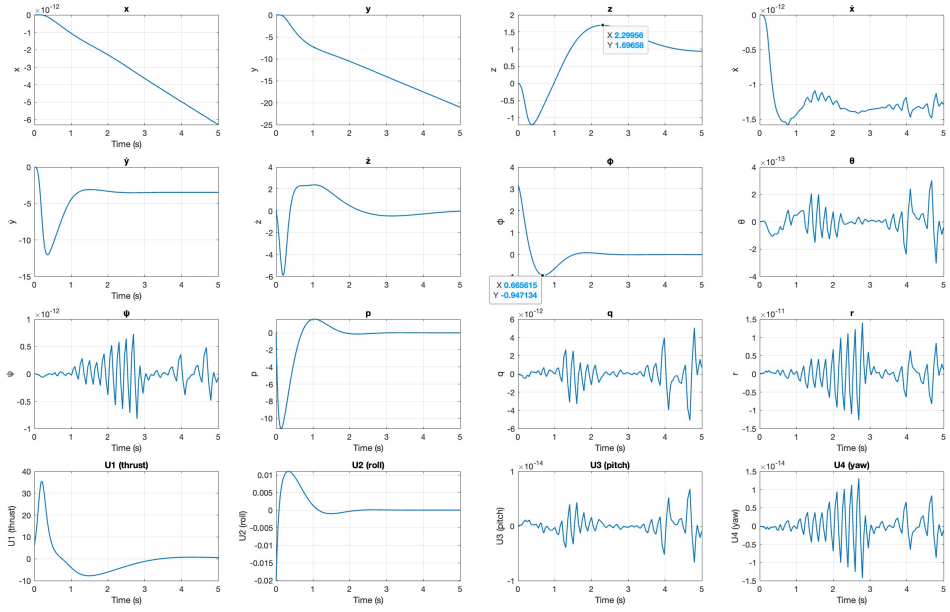


(b)

Figure 23: System Response (a) Linear and (b) Non-Linear to Free Throw for PID Control



(a)



(b)

Figure 24: System Response (a) Linear and (b) Non-Linear to Inverted Orientation for PID Control

Analysis of the results presented above reveals the most prominent improvement: the elimination of steady-state error in the non-linear model. This was a principal motivation for the introduction of integral action and clearly demonstrates the effect that incorporating an integral gain can have on system performance.

From a theoretical perspective, the integral term contributes infinite DC gain - the gain at zero frequency - which enables it to eliminate constant steady-state errors by continuously amplifying any residual offset. Under the influence of full PID control, both the linear and non-linear models now exhibit stable behaviour.

However, the inclusion of integral action has not mitigated the high-frequency perturbations observed in the non-linear response. In some instances, the magnitude of these perturbations has increased by over three orders of magnitude. A likely explanation is that the overall controller gain values in the PID implementation are significantly larger - by around an order of magnitude — which amplifies not only the control effort but also any numerical or dynamic noise present in the system.

This increase in gain may also account for the faster transient response observed in the linear model following the introduction of the integrator. While integrators typically reduce system responsiveness due to their low-frequency emphasis, the increased proportional and derivative gain values in the PID design more than compensate for this effect, resulting in an overall faster response.

Between the two control strategies evaluated, the PID controller demonstrably provides superior performance in terms of steady-state accuracy and overall stability. Consequently, PID control is selected as the preferred method for this system. Some tuning of the gain values may be required to reduce high-frequency noise and further optimise the transient response in the non-linear model, but the PID structure offers a more robust and effective control solution overall.

In Table 9 below, the requirements for the implementation of this control system are outlined.

Table 9: Required measurements and corresponding instrumentation for full controller implementation

Measurement	**Instrument**
Roll rate p, Pitch rate q, Yaw rate r	3-axis gyroscope (IMU)
Roll angle ϕ, Pitch angle θ	3-axis accelerometer & gyroscope (IMU)
Yaw angle ψ	3-axis magnetometer
Position x, y	GPS (outdoors) or motion-capture system (indoors)
Velocity $\dot{x}, \dot{y}$	Derived from position or optical-flow sensor
Altitude z	Barometric altimeter or LiDAR/ultrasonic altimeter
Vertical velocity $\dot{z}$	Derived from altitude or dedicated vertical-speed sensor
Rotor speeds ω_i	ESC telemetry or optical/tachometer sensor
Unmeasurable or unreliable states	State observer (EKF/UKF or Luenberger observer)

5 Trajectory Control and Hardware Deployment

5.1 Simulating Complex Trajectory - Adding x and y Control

To enable the drone to execute complex trajectories, control must be extended to the x and y translational directions in addition to the four previously stabilised states. This requires the implementation of two additional PID controllers dedicated to position control in the horizontal plane.

However, the same controller design methodology used for the inner-loop dynamics cannot be directly applied here, as the open-loop transfer functions for x and y do not conform to the form $G(s) = \frac{1}{Ms^2}$. Instead, these axes exhibit a quadruple integrator structure, rendering the previous design approach inapplicable. Therefore, consider the block diagram shown in Figure 25 below of the outer loop.

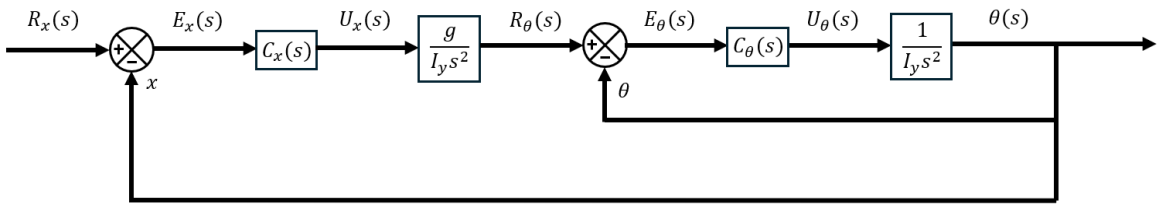


Figure 25: Transfer Function of outer loop.

One proposed method for addressing this is to assume a time-scale separation between the inner and outer control loops—specifically, that the inner loops governing ϕ and θ respond significantly

faster than the outer x and y loops. Under this assumption, the inner loops may be approximated as instantaneous, reducing the outer-loop dynamics to a double integrator form and permitting application of the standard tuning procedure. However, in practice, this method did not produce stable results. A likely explanation is that the time-scale separation was insufficient; the x and y position controllers were assigned settling times of 10 seconds, while the ϕ and θ controllers had settling times of 5 seconds—resulting in a coupling effect too strong to neglect.

As a result, the PID gains for the x and y controllers were manually tuned in conjunction with retuning of the existing PID parameters to ensure stable and coordinated performance across all six states as shown in Table 10.

Table 10: PID Controller Gains for Motion in a Complex Trajectory

Controller	K_P	K_I	K_D
U_1	8.6	11.18	5.16
U_2	0.009	0.0003	0.001
U_3	0.009	0.0003	0.001
U_4	0.003	0.0004	0.0018
U_x	5	2	3.8
U_y	5	2	3.8

This approach successfully produced a fully stabilised drone capable of controlled motion in three-dimensional space. To demonstrate this capability, the drone was tasked with navigating a server room environment.

The trajectory required the drone to move around two rows of server racks, pause at designated points to simulate scanning, and return to a docking station located in the corner. The complete trajectory, along with the corresponding plots of x, y, z and ϕ, θ, ψ, is presented in Figures 26 and 27 below.

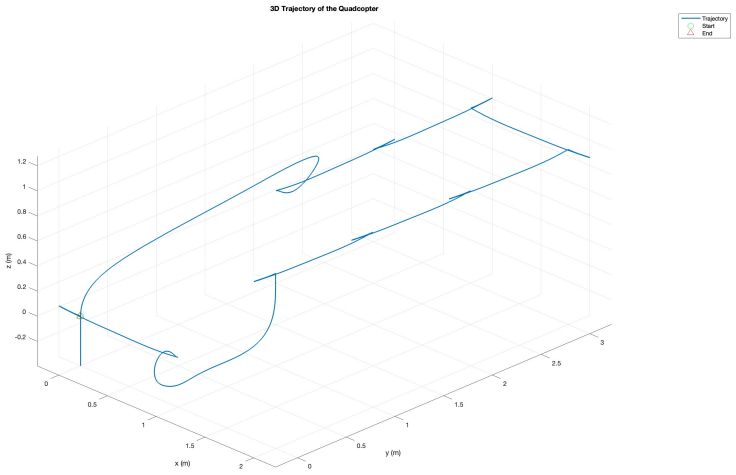


Figure 26: Complex trajectory modelled in MatLab in 3D from code in Appendix **??**

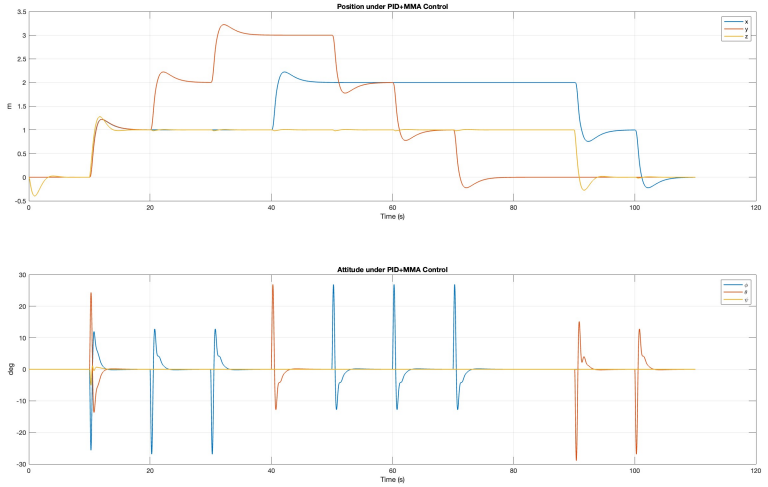


Figure 27: Plot of complex trajectory.

5.2 Hardware Implementation

The final part of the assignment required deploying the control algorithm, developed in simulation as described in Section 5.1, onto the Parrot Mambo Minidrone hardware. This deployment was arranged through a collaboration between our lecturer, Dr. A. Panday from the University of the Witwatersrand, and MathWorks associates. The flight session was scheduled for 19 May at 12:30 in the NWE Lab mezzanine area. In preparation, one of our group members, Rune, emailed Dr. Panday the PID gain values on the same day for the live flight demonstration.

However, during the scheduled session, the drone was not operational. As a result, an alternate assessment was offered to all groups in attendance, including ours. This involved submitting an animation file of the drone following the intended trajectory in a virtual environment. Unfortunately, this alternative could also not be completed due to a large-scale MathWorks server outage, which began on 18 May. This outage affected numerous users worldwide and prevented access to critical MATLAB functionalities.

Specifically, we were unable to install the required Minidrone Support Package due to persistent errors such as “no healthy upstream” when attempting to sign in to MATLAB, download add-ons, install new MATLAB versions, or manually integrate the support package (even when shared by classmates).

All relevant documentation supporting this issue is included in the final appendix of this report. This includes the original flight demonstration session email, the follow-up communication offering the revised assessment, correspondence with Dr. Panday detailing our efforts and the impact of the outage, user complaints from the official MATLAB subreddit, and a screenshot from the MathWorks server status page confirming the service disruption during the critical period.

6 Comments and Conclusions

A cascaded PID control system was successfully designed and validated for a quadrotor drone operating in a constrained indoor environment. The controller achieved full six-degree-of-freedom stability and enabled precise trajectory tracking in simulation. Comparative analysis demonstrated that PID control outperforms PD control, particularly in eliminating steady-state errors and handling nonlinear dynamics. The system meets the requirements for stable, reliable flight and forms a strong foundation for future real-world implementation.

References

[1] Server Racks Online, *Overhead cable management*, Accessed: 2025-05-21, 2025. [Online]. Available: https://www.server-rack-online.com/overhead-cable-management/.

[2] Sysracks. 'How are data centers cooled?' Accessed: 2025-05-20. (n.d.), [Online]. Available: https://sysracks.com/blog/how-are-data-centers-cooled/.

[3] Server Room Environments. 'Thermal camera surveys for critical data centre systems.' Accessed: 2025-05-20. (n.d.), [Online]. Available: https://www.serverroomenvironments.co.uk/blog/thermal-camera-surveys-for-critical-data-centre-systems.

[4] Black Box. 'Server cabinet cooling.' Accessed: 2025-05-20. (n.d.), [Online]. Available: https://www.blackbox.co.uk/gb-gb/page/25703/Resources/Technical-Resources/Black-Box-Explains/Cabinets/Server-Cabinet-Cooling.

[5] X. Zhou, J. Liu, W. Zhang and M. Li. 'Simulation of a temperature adaptive control strategy for an iwse economizer in a data center.' Accessed: 2025-05-20. (2014), [Online]. Available: https://www.researchgate.net/publication/264982820_Simulation_of_a_temperature_adaptive_control_strategy_for_an_IWSE_economizer_in_a_data_center.

[6] J. O. Pedro, M. Dangor and P. J. Kala, 'Differential evolution-based pid control of a quadrotor system for hovering application,' in *2016 IEEE Congress on Evolutionary Computation (CEC)*, Accessed: 2025-05-20, IEEE, 2016, pp. 2791–2798. DOI: 10.1109/CEC.2016.7744262.

[7] M. Belkheiri, A. Rabhi, A. E. Hajjaji and C. Pégard, 'Different linearization control techniques for a quadrotor system,' *ResearchGate*, 2014, Accessed: 2025-05-20. DOI: 10.13140/2.1.1596.2400. [Online]. Available: https://www.researchgate.net/publication/259780946_Different_linearization_control_techniques_for_a_quadrotor_system.

[8] M. Okasha, J. Kralev and M. Islam, 'Design and experimental comparison of pid, lqr and mpc stabilizing controllers for parrot mambo mini-drone,' *Aerospace*, vol. 9, no. 6, p. 298, 2022, Accessed: 2025-05-20. DOI: 10.3390/aerospace9060298. [Online]. Available: https://www.mdpi.com/2226-4310/9/6/298.

[9] H. Suresh, A. Sulficar and V. Desai, 'Hovering control of a quadcopter using linear and non-linear techniques,' *International Journal of Modelling, Identification and Control*, 2020, Accessed: 2025-05-20. [Online]. Available: https://www.researchgate.net/publication/327431213_Hovering_control_of_a_quadcopter_using_linear_and_nonlinear_techniques.

[10] Swarthmore College, *Transfer function to state space*, https://lpsa.swarthmore.edu/Representations/SysRepTransformations/TF2SS.html, Accessed: 2024-05-21, 2024.

[11] G. Delgado-Reyes, J. S. Valdez-Martínez, P. Guevara-López and M. A. Hernández-Pérez, 'Hover flight improvement of a quadrotor unmanned aerial vehicle using pid controllers with an integral effect based on the riemann–liouville fractional-order operator: A deterministic approach,' *Fractal and Fractional*, vol. 8, no. 11, p. 634, 2024. DOI: 10.3390/fractalfract8110634. [Online]. Available: https://www.mdpi.com/2504-3110/8/11/634.

[12] The MathWorks, Inc., *Ode45 – solve nonstiff differential equations – medium order method*, https://www.mathworks.com/help/matlab/ref/ode45.html, Accessed: 2024-05-21, 2024.

A Appendix A: Simulating the linearised drone response to step, sin and ramp inputs of U1, U2, U3 and U4

```
function simulateDroneLinear_UResponses_12States()

    % Computes Step, Ramp, and Sinusoid responses for a chosen input
       channel
    % on the 12-state linearized quadrotor model and plots each state
       on its
    % own subplot with the input overlaid in light grey.

    %% 1) Parameters & state-space matrices
    m  = 0.506;
    g  = 9.81;
    Ix = 8.11858e-5;
    Iy = 8.11858e-5;
    Iz = 6.12223e-5;

    A = zeros(12);
    A(1,4)=1;   A(2,5)=1;   A(3,6)=1;
    A(4,8)= g;  A(5,7)=-g;
    A(7,10)=1;  A(8,11)=1;  A(9,12)=1;

    B = zeros(12,4);
    B(6,1)=1/m;
    B(10,2)=1/Ix;
    B(11,3)=1/Iy;
    B(12,4)=1/Iz;

    C = eye(12);
    D = zeros(12,4);

    sys = ss(A, B, C, D);

    %% 2) Select which input channel to test and input amplitude
    inputChannel = 4;          % change to 1  4   as desired
    Ainp = 3.06115e-5;         % choose input size

    %% 3) Time vector
    tspan = 0:0.01:20;

    %% 4) Define test signals

```

```
tests = { ...
  struct('name','Step',     'u',@(t) Ainp*ones(size(t))), ...
  struct('name','Ramp',     'u',@(t) Ainp*t), ...
  struct('name','Sinusoid', 'u',@(t) Ainp*sin(2*pi*1*t)) ...
};

%% 5) Loop through each test, simulate, and plot

for k = 1:numel(tests)
    test = tests{k};
    Uvec = test.u(tspan);
    U = zeros(length(tspan),4);
    U(:,inputChannel) = Uvec;

    % simulate linear response of all states
    [Y, T] = lsim(sys, U, tspan);

    % create figure
    figure('Name',[test.name ' Linear Response     U_' num2str(
        inputChannel)] , ...
            'NumberTitle','off');

    % plot all 12 states each on its own subplot
    stateNames = {'x','y','z','   ','   ','  ','  ','  ','  ','p',
        'q','r'};
    for i = 1:12
        subplot(4,3,i)
        plot(T, Y(:,i), 'LineWidth',1.2); hold on
        plot(T, Uvec,   'Color',[0.8 0.8 0.8],'LineWidth',1);
        hold off
        grid on; axis square
        title([stateNames{i} ' Response'])
        xlabel('Time [s]')
        if i <= 3
            ylabel('Position [m]')
        elseif i <= 6
            ylabel('Velocity [m/s]')
        elseif i <= 9
            ylabel('Angle [rad]')
        else
            ylabel('Rate [rad/s]')
        end
        legend(stateNames{i},['U_' num2str(inputChannel) ' in'],'
           Location','best')
```

```
        end
    end
end
```

B Simulating the NON LINEAR drone response to step, sin and ramp inputs of U1, U2, U3 and U4

```
function simulateDroneNonlinear_UResponses_12States()
    %% 1) Load vehicle parameters and compute mixer inverse
    params      = loadQuadParams();
    kF          = params.kF;
    l           = params.l;
    B           = params.B;
    m           = params.m;
    g           = params.g;

    M = [  kF,     kF,     kF,    kF;
           0,  -kF*l,      0,  kF*l;
        -kF*l,     0,   kF*l,     0;
          -B,      B,     -B,     B ];
    Minv = inv(M);

    %% 2) Select which input channel to test
    inputChannel = 4;          % change to 1  4   as desired
    A = 5*1.624e-6;            % choose input size

    %% 3) Initial state and time vector
    x0     = zeros(12,1);              % [x;y;z;u;v;w;phi;theta;psi;p;q;r
       ]
    tspan = 0:0.01:20;                 % 0   5s

    %% 4) Define test signals

    tests = { ...
      struct('name','Step',     'u',@(t) A*ones(size(t))), ...
      struct('name','Ramp',     'u',@(t) A*t), ...
      struct('name','Sinusoid', 'u',@(t) A*sin(2*pi*1*t)) ...
    };

    %% 5) Loop through each test, simulate, and plot
    for k = 1:numel(tests)
```

```
34          test = tests{k};
35          Uvec = test.u(tspan);
36
37          % simulate open-loop nonlinear dynamics
38          [T, X] = ode45(@(t,x) nlDyn(t, x, test.u), tspan, x0);
39
40          % create figure
41          figure('Name',[test.name ' Non-Linear Response     U_' num2str
                (inputChannel)], ...
42              'NumberTitle','off');
43
44          % plot all 12 states each on its own subplot
45          stateNames = {'x','y','z','   ','   ','  ','  ','  ','  ','p',
                'q','r'};
46          for i = 1:12
47              subplot(4,3,i)
48              plot(T, X(:,i), 'LineWidth',1.2); hold on
49              plot(T, Uvec,  'Color',[0.8 0.8 0.8],'LineWidth',1);
50              hold off
51              grid on; axis square
52              title([stateNames{i} ' Response'])
53              xlabel('Time [s]')
54              if i <= 3
55                  ylabel('Position [m]')
56              elseif i <= 6
57                  ylabel('Velocity [m/s]')
58              elseif i <= 9
59                  ylabel('Angle [rad]')
60              else
61                  ylabel('Rate [rad/s]')
62              end
63              legend(stateNames{i},['U_' num2str(inputChannel) ' in'],'
                    Location','best')
64          end
65      end
66
67      %% Nested dynamics function
68      function dx = nlDyn(t, x, uFcn)
69          % baseline hover thrust to cancel weight
70          U = [m*g; 0; 0; 0];
71          % add test signal on selected channel
72          u = uFcn(t);
73          U(inputChannel) = U(inputChannel) + u;
74          % mixer     rotor speeds
```

```
        f            = Minv * U;
        params.omega = sqrt(max(f,0));
        % compute state derivatives
        dx = quadNonlinearDynamics(t, x, params);
    end
end
```

C Load Quad Params. Loads the parameters of the quadcopter. Used as a helper function throughout the coding process.

```
% loadQuadParams.m
% ------------------------------------------------
% Helper function to load quadcopter parameters

function params = loadQuadParams()
    params.m   = 0.506;
    params.g   = 9.81;
    params.I_x = 8.11858e-5;
    params.I_y = 8.11858e-5;
    params.I_z = 6.12223e-5;
    params.l   = 0.235;
    params.kF  = 3.13e-5;
    params.B   = 75e-7;
end
```

D Appendix 1 new - simulates uncontrolled non linear drone

```
%% Simulate Non Linear Drone

params.m    = 0.506;     % mass [kg]
params.g    = 9.81;      % gravitational acceleration [m/s^2]
params.I_x  = 8.11858e-5;   % moment of inertia about x-axis [kg m^2]
params.I_y  = 8.11858e-5;   % moment of inertia about y-axis [kg m^2]
params.I_z  = 6.12223e-5;   % moment of inertia about z-axis [kg m^2]
params.l    = 0.235;         % two times arm length [m]

%% Rotor constants:
params.kF   = 3.13e-5;   % Lift constant [N/(rad/s)^2]
params.B    = 75e-7*0;   % Drag constant [N m/(rad/s)^2]
```

```
13
14 %% Compute ideal rotor speed for hover
15 omega_hover = sqrt(params.m*params.g/(4*params.kF));
16 params.omega = repmat(omega_hover,4,1);
17
18 %% Initial condition: ['x','y','z','   ','   ','  ','  ','  ','  ','p
       ','q','r']
19 x0 = zeros(12,1);
20 x0(1) = 1;
21       x0(1) = 1;
22     x0(2) = 1;
23     x0(3) = 1;
24     x0(4) = 0;
25     x0(5) = 0;
26     x0(6) = 0;
27     x0(7) = 0;
28     x0(8) = 0;
29     x0(9) = 30*pi/180;
30     x0(10) = 0;
31     x0(11) = 0;
32     x0(12) = 0;
33 %% Time span
34 tspan = [0 20];
35
36 %% Simulate open-loop hover
37 [t, X] = ode45(@(t,x) quadNonlinearDynamics(t, x, params), tspan, x0);
38
39 %% Plot all 12 states in a 4x3 grid
40 stateNames = {'x','y','z','   ','   ','  ','  ','  ','  ','p','q','r'
       };
41 yLabels    = {'Position (m)','Position (m)','Position (m)', ...
42               'Velocity (m/s)','Velocity (m/s)','Velocity (m/s)', ...
43               'Angle (rad)','Angle (rad)','Angle (rad)', ...
44               'Rate (rad/s)','Rate (rad/s)','Rate (rad/s)'};
45
46 figure('Name','Hover Response: 12 States','NumberTitle','off');
47 for i = 1:12
48     subplot(4,3,i)
49     if i>=7 && i<=9
50         % Euler angles: plot in radians
51         plot(t, X(:,i), 'LineWidth',1.2);
52     else
53         plot(t, X(:,i), 'LineWidth',1.2);
54     end
```

```
    title(stateNames{i});
    xlabel('Time (s)');
    ylabel(yLabels{i});
    grid on
    axis tight
end
```

E Appendix 2 new - simulates uncontrolled linear drone

```
%% 1) Define symbolic state and input variables
syms x y z u v w phi theta psi p q r real
syms omega1 omega2 omega3 omega4 real

X = [x; y; z; u; v; w; phi; theta; psi; p; q; r];
U = [omega1; omega2; omega3; omega4];

%% 2) Parameters & equilibrium
m   = 0.506;    g = 9.81;
I_x = 8.11858e-5; I_y = 8.11858e-5; I_z = 6.12223e-5;
l   = 0.235;    kF = 3.13e-5;    B = 0;

omega_hover = sqrt(m*g/(4*kF));
U_eq = repmat(omega_hover,4,1);
X_eq = zeros(12,1);

%% 3) Symbolic nonlinear dynamics f(X,U)
f = sym(zeros(12,1));
f(1:3) = [u; v; w];
T1 = kF*omega1^2; T2 = kF*omega2^2; T3 = kF*omega3^2; T4 = kF*omega4
    ^2;
F = T1+T2+T3+T4;
f(4) = (F/m)*(cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi));
f(5) = (F/m)*(cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi));
f(6) = (F/m)*cos(phi)*cos(theta) - g;
f(7) = p + q*sin(phi)*tan(theta) + r*cos(phi)*tan(theta);
f(8) = q*cos(phi) - r*sin(phi);
f(9) = q*sin(phi)/cos(theta) + r*cos(phi)/cos(theta);
tau_phi   = l*(T4-T2);
tau_theta = l*(T3-T1);
tau_psi   = B*(-omega1^2+omega2^2-omega3^2+omega4^2);
f(10) = (tau_phi   - (I_y-I_z)*q*r)/I_x;
f(11) = (tau_theta - (I_z-I_x)*p*r)/I_y;
```

```
f(12) = (tau_psi   - (I_x-I_y)*p*q)/I_z;

%% 4) Jacobians
A_sym = jacobian(f, X);
B_sym = jacobian(f, U);

%% 5) Numeric linearization at hover
subsList = [X; U];
eqList   = [X_eq; U_eq];
A_lin = double(subs(A_sym, subsList, eqList));
B_lin = double(subs(B_sym, subsList, eqList));

disp('Linearized A at hover:'); disp(A_lin);
disp('Linearized B at hover:'); disp(B_lin);
save('linearModel.mat','A_lin','B_lin');

%% 6) Simulate the linear system     = A_lin * x
tspan = [0 10];
IC = input('Enter 12 1  initial state [x y z u v w phi theta psi p q r
    ]: ');
if numel(IC)~=12, error('Must supply 12  element   vector'); end
[T, X] = ode45(@(t,x) A_lin*x, tspan, IC(:));

%% 7) Plot the 12 states in a 3 4  grid
stateNames = {'x','y','z','u','v','w','\phi','\theta','\psi','p','q','
    r'};
yLabels = {'Position (m)','Position (m)','Position (m)', ...
           'Velocity (m/s)','Velocity (m/s)','Velocity (m/s)', ...
           'Angle (rad)','Angle (rad)','Angle (rad)', ...
           'Rate (rad/s)','Rate (rad/s)','Rate (rad/s)'};

figure('Name','Linear Hover Response','NumberTitle','off');
for i = 1:12
    subplot(3,4,i);
    plot(T, X(:,i), 'LineWidth',1.2);
    title(stateNames{i});
    xlabel('Time (s)');
    ylabel(yLabels{i});
    grid on; axis tight;
end
```

F Function that gets called within Appendix 1 to solve for DE of non linear system

```
function dx = quadNonlinearDynamics(~, x, params)
% quadNonlinearDynamics computes the state derivatives for a quadcopter
% using a detailed nonlinear model that calculates the thrusts and torques
% based on individual rotor speeds.
%
% The state vector x is:
%   x = [x; y; z; u; v; w; phi; theta; psi; p; q; r]
% where:
%   - (x, y, z) are the positions in the inertial frame,
%   - (u, v, w) are the linear velocities,
%   - (phi, theta, psi) are the Euler angles (roll, pitch, yaw),
%   - (p, q, r) are the angular velocities about the body axes.
%
% The model computes:
%   - For each rotor: T_i = kF * omega_i^2
%   - Total thrust: F_total = T1 + T2 + T3 + T4
%   - Roll torque:   tau_phi   = l * (T4 - T2)
%   - Pitch torque:  tau_theta = l * (T3 - T1)
%   - Yaw torque:    tau_psi   = kM * (omega1^2 - omega2^2 + omega3^2 - omega4^2)
%
% These are used to compute both the translational dynamics (by rotating the thrust
% into the inertial frame) and the rotational dynamics (using Euler's equations).

% Unpack state variables
pos   = x(1:3);      % Position [x; y; z]
vel   = x(4:6);      % Linear velocity [u; v; w]
phi   = x(7);        % Roll angle
theta = x(8);        % Pitch angle
psi   = x(9);        % Yaw angle
p     = x(10);       % Roll rate
q     = x(11);       % Pitch rate
r     = x(12);       % Yaw rate

% Unpack parameters
m   = params.m;
```

```
g    = params.g;
I_x = params.I_x;
I_y = params.I_y;
I_z = params.I_z;
l    = params.l;
kF   = params.kF;
B   = params.B;

% Compute individual rotor thrusts (T_i = kF * omega_i^2)
T1 = kF * (params.omega(1)^2);
T2 = kF * (params.omega(2)^2);
T3 = kF * (params.omega(3)^2);
T4 = kF * (params.omega(4)^2);

% Total thrust produced by all four rotors
F_total = T1 + T2 + T3 + T4;

% Compute torques:
% Roll torque: difference in thrust between right and left rotors.
tau_phi   = l * (T4 - T2);
% Pitch torque: difference in thrust between the front and back rotors
    .
tau_theta = l * (T3 - T1);
% Yaw torque: due to reactive drag torques generated by the rotors (
    with appropriate sign).
tau_psi   = B * (-params.omega(1)^2 + params.omega(2)^2 - params.omega
    (3)^2 + params.omega(4)^2);

% Preallocate state derivative vector
dx = zeros(12,1);

% ------------------------------
% Kinematics: Position derivatives
% ------------------------------
dx(1:3) = vel;  % The derivative of position is the velocity

% ------------------------------
% Translational Dynamics:
% The thrust is applied along the body-fixed z-axis.
% Use the Euler angles to project F_total into the inertial frame.
% ------------------------------
dx(4) = (F_total/m) * (cos(phi)*sin(theta)*cos(psi) + sin(phi)*sin(psi
    ));
```

```
dx(5) = (F_total/m) * (cos(phi)*sin(theta)*sin(psi) - sin(phi)*cos(psi
    ));
dx(6) = (F_total/m) * (cos(phi)*cos(theta)) - g;

% ------------------------------
% Euler Angle Kinematics:
% Assuming a Z-Y-X (yaw-pitch-roll) sequence.
% ------------------------------
dx(7) = p + q*sin(phi)*tan(theta) + r*cos(phi)*tan(theta);
dx(8) = q*cos(phi) - r*sin(phi);
dx(9) = q*sin(phi)/cos(theta) + r*cos(phi)/cos(theta);

% ------------------------------
% Rotational Dynamics (Euler Equations)
% ------------------------------
dx(10) = (tau_phi - (I_y - I_z)*q*r) / I_x;    % Roll rate derivative (
    p_dot)
dx(11) = (tau_theta - (I_z - I_x)*p*r) / I_y;  % Pitch rate derivative
    (q_dot)
dx(12) = (tau_psi - (I_x - I_y)*p*q) / I_z;       % Yaw rate derivative
    (r_dot)

end
```

G APPENDIX-NONLINEAR-PID: Calculates the response of the non-linearised system under the affect of PID

```
function NON_LINEAR_NOXY_12STATES()

    % NOTE:
    % Make sure quadNonlinearDynamics.m and loadQuadParams.m are on
       your path


    %% 1) Load parameters & hover speed
    params      = loadQuadParams();
    kF          = params.kF;
    l           = params.l;
    B           = params.B;
    m           = params.m;
    g           = params.g;
```

```
    omega_hover = sqrt(m*g/(4*kF));

    %% 2) PID gains
    Kp_z     =  6.4016 ;  Ki_z     = 6.2663;  Kd_z    = 4.2324;
    Kp_phi   = 0.0064;  Ki_phi   =  0.0157 ;  Kd_phi = 0.0017;
    Kp_theta = 0.0064;  Ki_theta =  0.0157 ;  Kd_theta = 0.0017;
    Kp_psi   = 0.0048;  Ki_psi   = 0.0119;  Kd_psi = 0.0013;

    %% 3) References
    z_ref     = 1.0;     % [m]
    phi_ref   = 0.0;     % [rad]
    theta_ref = 0.0;     % [rad]
    psi_ref   = 0.0;     % [rad]

    %% 4) Precompute mixer inverse
    M = [  kF,     kF,     kF,    kF;
           0,  -kF*l,      0,  kF*l;
       -kF*l,     0,   kF*l,      0;
         -B,      B,     -B,      B ];
    Minv = inv(M);

    %% 5) Initial condition (12 states + 4 integrators)
    x0        = zeros(12,1);

x0 = zeros(12,1);
x0(1) = 0;                    % x
x0(2)= 0;                     % y
x0(3) = 0;                    % z
x0(4) = 0;                    % u
x0(5) = 0;                    % v
x0(6) = 0;                    % w
x0(7) = pi;                   % phi
x0(8) = 0;                    % theta
x0(9) = 0;                    % psi
x0(10) = 0;                   % p
x0(11) = 0;                   % q
x0(12) = 0;                   % r

    I0        = zeros(4,1);        % initial integrals [iz; ip; it; ips
       ]
    X0_ext    = [x0; I0];

    %% 6) Simulate
    tspan = [0 8];
```

```
    [t, Xext] = ode45(@(t, X) combinedDynExt(...
        t, X, params, omega_hover, Minv, ...
        Kp_z,Ki_z,Kd_z, ...
        Kp_phi,Ki_phi,Kd_phi, ...
        Kp_theta,Ki_theta,Kd_theta, ...
        Kp_psi,Ki_psi,Kd_psi, ...
        z_ref,phi_ref,theta_ref,psi_ref), ...
        tspan, X0_ext);

    %% 7) Extract states and recompute inputs
    X = Xext(:,1:12);
    U = zeros(length(t),4);
    iz=0; ip=0; it=0; ips=0;
    dt = t(2)-t(1);
    for k=1:length(t)
        xs = X(k,:)';
        z    = xs(3); w    = xs(6);
        phi  = xs(7); p    = xs(10);
        theta= xs(8); q    = xs(11);
        psi  = xs(9); r    = xs(12);
        % Altitude
        ez = z_ref - z; iz = iz + ez*dt; dez = -w;
        U1 = Kp_z*ez + Ki_z*iz + Kd_z*dez;
        % Roll
        e_phi = phi_ref - phi; ip = ip + e_phi*dt; dep = -p;
        U2 = Kp_phi*e_phi + Ki_phi*ip + Kd_phi*dep;
        % Pitch
        e_th = theta_ref - theta; it = it + e_th*dt; det = -q;
        U3 = Kp_theta*e_th + Ki_theta*it + Kd_theta*det;
        % Yaw
        e_ps = psi_ref - psi; ips = ips + e_ps*dt; deps = -r;
        U4 = Kp_psi*e_ps + Ki_psi*ips + Kd_psi*deps;
        U(k,:) = [U1 U2 U3 U4];
    end

    %% 8) Plot each state and input in a 4x4 grid
    figure;
    stateNames = {'x','y','z','  ','  ','  ','  ','  ','  ','p','q',
        'r'};
    for i=1:12
        subplot(4,4,i);
        plot(t, X(:,i), 'LineWidth',1.2);
        ylabel(stateNames{i});
        title(stateNames{i});
```

```
        if i>8 || mod(i,4)==1, xlabel('Time (s)'); end
        grid on;
    end
    inputNames = {'U1 (thrust)','U2 (roll)','U3 (pitch)','U4 (yaw)'};
    for j=1:4
        subplot(4,4,12+j);
        plot(t, U(:,j), 'LineWidth',1.2);
        ylabel(inputNames{j}); xlabel('Time (s)'); grid on;
        title(inputNames{j});
    end
end

function dXext = combinedDynExt(t, Xext, params, omega_hover, Minv,
    ...
                                Kp_z,Ki_z,Kd_z, ...
                                Kp_phi,Ki_phi,Kd_phi, ...
                                Kp_theta,Ki_theta,Kd_theta, ...
                                Kp_psi,Ki_psi,Kd_psi, ...
                                z_ref,phi_ref,theta_ref,psi_ref)
    x     = Xext(1:12);
    iz    = Xext(13); ip    = Xext(14);
    it    = Xext(15); ips   = Xext(16);
    dt = 0.01;
    z     = x(3); w     = x(6);
    phi   = x(7); p     = x(10);
    theta = x(8); q     = x(11);
    psi   = x(9); r     = x(12);
    % PID computations
    ez  = z_ref - z;    iz  = iz + ez*dt; dez  = -w;    U1  = Kp_z*ez
        + Ki_z*iz + Kd_z*dez;
    e_phi = phi_ref - phi; ip = ip + e_phi*dt; dep = -p; U2 = Kp_phi*
        e_phi + Ki_phi*ip + Kd_phi*dep;
    e_th  = theta_ref - theta; it = it + e_th*dt; det = -q; U3 =
        Kp_theta*e_th + Ki_theta*it + Kd_theta*det;
    e_ps  = psi_ref - psi; ips = ips + e_ps*dt; deps = -r; U4 = Kp_psi
        *e_ps + Ki_psi*ips + Kd_psi*deps;
    U  = [U1;U2;U3;U4]; f = Minv*U; f = max(f,0); omega = sqrt(f);
    params.omega = omega;
    dx = quadNonlinearDynamics(t, x, params);
    dXext = [dx; ez; e_phi; e_th; e_ps];
end
```

H Appendix-NYquist : Generates Nyquist Plots

```
% NOTE: This code uses the function nyqlog. To run the code, download
    the
% function from MatLab directly.

%% Define Variables:
s  = tf('s');

m  = 0.506;                % mass [kg]
Ix = 8.11858e-5;           % roll-axis inertia [kg m^2]
Iy = 8.11858e-5;           % pitch-axis inertia [kg m^2]
Iz = 6.12223e-5;           % yaw-axis inertia [kg m^2]

K_p = 6.4016;
K_d = 4.2324;
K_i = 6.2663;

% Define Laplace variable
s = tf('s');

% Transfer functions
Gz      = 1/m*s^2;
Gx = 1/(Iy*s^4);
Gy = -1/(Ix*s^4);
Gpd = (K_p + K_d*s+K_i/s)/(m*s^2);

%% FOR TWO SIDE BY SIDE PLOTS

figure;

 % Left plot
subplot(1,2,1);
nyquist(Gpd);
grid on;
title('Nyquist plot of PID controlled Open Loop');
%ylim(1e-3*[-1 1]);                          % Add for pure integrator
    plots
%xlim([-5 1]);
xlabel('Re.');
ylabel('Im.');

 % Right plot
subplot(1,2,2);
```

```
nyqlog(Gpd);
grid on;
title('Log Nyquist plot of PID controlled Open Loop');
xlabel('Re.');
ylabel('Im.');

%% FOR A SINGLE PLOT

%figure;
%     nyquist(Gpd);
%    grid on;
%    title('Nyquist plot');
%     xlabel('Re');
%     ylabel('Im');
```

I Appendix-RootLocusPlot : Generates Root Locus Plots

```
%% Root Locus For Design of U1

m = 0.506;
s = tf('s');

%% 1) P
figure;
G_P = 1/(m*s^2);          % plant
rlocus(G_P)               % loci of poles as K varies
title('Root Locus: P on 1/ms^2')
grid on

%% 2) PD

m = 0.506;
s = tf('s');


figure;
Ts_z      = 5;
zeta_z    = 0.5911;
omega_n_z= 4/(Ts_z*zeta_z);
P_z       = (2*zeta_z)/omega_n_z;

G_D = (1 + s*P_z)/(m*s^2);
```

```
rlocus(G_D)
title('Root Locus: PD on 1/ms^2')
grid on

%% 3) PI

figure;
G_I = (s + P_z)/(m*s^3);
rlocus(G_I)
title('Root Locus: PI on 1/ms^2')
grid on

%% 4) PID
s = tf('s');
m = 0.506;          % [kg m^2]

Ts_z      = 5;
zeta_z    = 0.5911;
omega_n_z= 4/(Ts_z*zeta_z);
P_z       = 5*omega_n_z;

figure;
G_PID =  (0.6611*s^2 + s + 0.9789)/(m*s^3);
rlocus(G_PID)
title('Root Locus: PID on 1/(m s^2)')
grid on
```

J APPENDIX-LINEAR PID: Calculates the response of the linearised system under the affect of PID

```
%% 1) Build the hover linearised state space model
m  = 0.506;                % mass [kg]
g  = 9.81;                 % gravity [m/s^2]
Ix = 8.11858e-5;           % [kg m^2]
Iy = 8.11858e-5;           % [kg m^2]
Iz = 6.12223e-5;           % [kg m^2]

A = zeros(12);APPENDIX_LINEAR_PID
A(1,4)=1;  A(2,5)=1;  A(3,6)=1;
A(4,8)= g; A(5,7)=-g;
A(7,10)=1; A(8,11)=1; A(9,12)=1;
```

```
12
13 B = zeros(12,4);
14 B(6,1)=1/m;    B(10,2)=1/Ix;
15 B(11,3)=1/Iy;  B(12,4)=1/Iz;
16
17 C = zeros(4,12);
18 C(1,3)=1;   C(2,7)=1;
19 C(3,8)=1;   C(4,9)=1;
20 D = zeros(4,4);
21
22 drone_sys = ss(A, B, C, D);
23
24 %% 2) Define Laplace variable and derivative filter
25 s      = tf('s');
26 %% 3) PID GAINS
27
28 %% Z Gains
29
30 tau_d_z = 0.1;
31
32 Kpz      = 0.9263;
33 Ki_z     = 0;
34 Kdz      =  0.8094;
35
36 %% Phi Gains
37
38 tau_d_phi = 0.05;
39
40 Kp_phi    = 0.0009304;
41 Ki_phi     = 0;
42 Kd_phi      = 0.0003249;
43
44 %% Theta GAINS
45
46 tau_d_theta = 0.05;
47
48 Kp_theta    = 0.0009304;
49 Ki_theta    = 0;
50 Kd_theta     = 0.0003249;
51
52 %% Psi Gains
53
54 tau_d_psi = 0.05;
55
```

```
Kp_psi   = 0.0001121;
Ki_psi   = 0;
Kd_psi  = 0.0000979;



%% 4) CONTROLLERS:

Cz     = Kpz + Ki_z/s    + (Kdz    * s)/(tau_d_z*s + 1);
Cphi   = Kp_phi + Ki_phi/s + (Kd_phi * s)/(tau_d_phi*s + 1);
Ctheta = Kp_theta + Ki_theta/s + (Kd_theta*s)/(tau_d_theta*s + 1);
Cpsi   = Kp_psi + Ki_psi/s + (Kd_psi * s)/(tau_d_psi*s + 1);



%% 5) Convert each to  s t a t e space
Cz_ss     = ss(Cz);
Cphi_ss   = ss(Cphi);
Ctheta_ss = ss(Ctheta);
Cpsi_ss   = ss(Cpsi);


%% 6) Build  b l o c k diagonal  controller
C_all = blkdiag(Cz_ss, Cphi_ss, Ctheta_ss, Cpsi_ss);

%% 7) Close the full MIMO loop
drone_closed = feedback(drone_sys * C_all, eye(4));

%% 8) Simulate controlled performance

dt = 1e-4;                    % sampling period [s]
t  = 0:dt:5;                  % time vector


%% 9) SET INITIAL CONDITIONS
nx = size(drone_closed.A,1);
x0 = zeros(nx,1);
x0(1) = 0;                    % x
x0(2)= 0;                     % y
x0(3) = 0;                    % z
x0(4) = 0;                    % u
x0(5) = 0;                    % v
x0(6) = 0;                    % w
x0(7) = pi;                   % phi
```

```
x0(8) = 0;                      % theta
x0(9) = 0;                      % psi
x0(10) = 0;                     % p
x0(11) = 0;                     % q
x0(12) = 0;                     % r

%% 10) step in thrust, zero torque commands
u = [ 1*ones(length(t),1), zeros(length(t),3) ];

[y, t_out, x] = lsim(drone_closed, u, t, x0);

%% 11) Plot all 12 states and the 4 controller outputs in subplots
% Compute the tracking error and actual controller outputs
% Compute the tracking error and actual controller outputs
e      = u - y;                          % error = reference
    response
u_ctrl = lsim(C_all, e, t_out);        % controllers output signals

figure('Name','States and Control Signals','Position',[200 200 1200
    800]);

% Names for the 12 states
stateNames = {'x','y','z','   ','   ','   ','   ','   ','   ','p','q','r'
    };

% Plot the 12 states in a 4 4  grid (positions 1 to 12)
for i = 1:12
    subplot(4,4,i)
    plot(t_out, x_plot(:,i), 'LineWidth', 1.2)
    grid on
    title(stateNames{i})
    xlabel('Time [s]')
    ylabel(stateNames{i})
end

% Plot the 4 controller outputs (   U   U   ) in the last row (
    positions 13 to 16)
for j = 1:4
    subplot(4,4,12 + j)
    plot(t_out, u_ctrl_plot(:,j), 'LineWidth', 1.2)
    grid on
    title(sprintf('Control Output U_{%d}', j))
    xlabel('Time [s]')
    ylabel(sprintf('U_{%d}', j))
```

```
end
```

K Complex Trajectory Code

```
function hoverMMAControlXYPID()
    % hoverMMAControlXYPID     Full PID+MMA hover control following a waypoint path
    %
    % The quadrotor will hover at 0,0,0 then follow:
    %   [1,1,1]     [1,2,1]     [1,3,1]     [2,3,1]     [2,2,1]     [2,1,1],
    % holding each waypoint for 5 s .

    %% 1) Load params & hover speed
    params      = loadQuadParams();
    kF          = params.kF;
    l           = params.l;
    B           = params.B;
    m           = params.m;
    g           = params.g;
    omega_hover = sqrt(m * g / (4*kF));

    %% 2) PID gains
    Kp_z     =  8.6;   Ki_z     = 8;    Kd_z     = 5.16;    % altitude U1
    Kp_phi   = 0.009;  Ki_phi   =0.0003;    Kd_phi   =0.001;   % roll U2
    Kp_theta = 0.009;  Ki_theta =0.0003;    Kd_theta =0.001;   % pitch U3
    Kp_psi   = 0.003;  Ki_psi   =0.0004;    Kd_psi   =1.8e-3; % yaw U4

    Kp_x   = 5;  Ki_x   = 2;  Kd_x   = 3.8;  % outer x
    Kp_y   = Kp_x;  Ki_y   = Ki_x;  Kd_y   = Kd_x;  % outer y

    %% 3) Precompute mixer inverse
    M = [  kF,     kF,     kF,    kF;
           0,  -kF*l,      0,  kF*l;
       -kF*l,     0,   kF*l,      0;
         -B,      B,     -B,      B ];
    Minv = inv(M);

```

```
%% 4) Define waypoints and timing
waypts = [ 0 0 0;
           1 1 1;
           1 2 1;
           1 3 1;
           2 3 1;
           2 2 1;
           2 1 1;
           2 0 1;
           2 0 1;
           1 0 0;
           0 0 0;
           0 0 0];

holdTime = 10;                                    % seconds per
    waypoint
t_ref    = (0:size(waypts,1)-1) * holdTime;      % [0,5,10,...,30]
totalT   = t_ref(end);
tspan    = [0 totalT];

%% 5) Initial conditions
x0       = zeros(12,1);
x0(1:3)  = waypts(1,:)';    % start at [0;0;0]
x0(7:9)  = [0;0;0];         % zero attitudes
I0       = zeros(6,1);
X0_ext   = [x0; I0];

%% 6) Simulate over full trajectory
odefun = @(t,X) combinedDynXYPID( ...
    t, X, params, omega_hover, Minv, ...
    Kp_z,Ki_z,Kd_z, ...
    Kp_phi,Ki_phi,Kd_phi, ...
    Kp_theta,Ki_theta,Kd_theta, ...
    Kp_psi,Ki_psi,Kd_psi, ...
    Kp_x,Ki_x,Kd_x, ...
    Kp_y,Ki_y,Kd_y, ...
    interp1(t_ref, waypts(:,1), t, 'previous'), ...
    interp1(t_ref, waypts(:,2), t, 'previous'), ...
    interp1(t_ref, waypts(:,3), t, 'previous'), ...
    0, g);

[t, Xext] = ode45(odefun, tspan, X0_ext);

%% 7) Plot physical states
```

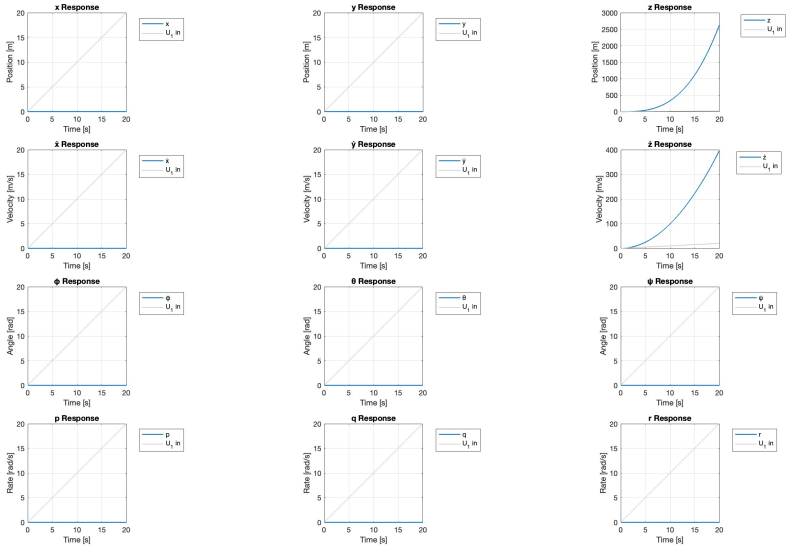
```
    X = Xext(:,1:12);

    figure;
    subplot(2,1,1);
    plot(t, X(:,1:3), 'LineWidth',1.2);
    legend('x','y','z');
    title('Position under PID+MMA Control');
    xlabel('Time (s)'); ylabel('m'); grid on;

    subplot(2,1,2);
    plot(t, rad2deg(X(:,7:9)), 'LineWidth',1.2);
    legend('\phi','\theta','\psi');
    title('Attitude under PID+MMA Control');
    xlabel('Time (s)'); ylabel('deg'); grid on;

    % 3D trajectory
    x_traj = X(:,1); y_traj = X(:,2); z_traj = X(:,3);
    figure;
    plot3(x_traj, y_traj, z_traj, 'LineWidth',1.5);
    grid on; axis equal;
    xlabel('x (m)'); ylabel('y (m)'); zlabel('z (m)');
    title('3D Trajectory of the Quadcopter');
    view(45,25);
    hold on;
    plot3(x_traj(1), y_traj(1), z_traj(1), 'go', 'MarkerSize',10, '
       DisplayName','Start');
    plot3(x_traj(end), y_traj(end), z_traj(end), 'r^', 'MarkerSize'
       ,10, 'DisplayName','End');
    legend('Trajectory','Start','End');
end

function dXext = combinedDynXYPID(t, Xext, params, omega_hover, Minv,
   ...
                                  Kp_z,Ki_z,Kd_z, ...
                                  Kp_phi,Ki_phi,Kd_phi, ...
                                  Kp_theta,Ki_theta,Kd_theta, ...
                                  Kp_psi,Ki_psi,Kd_psi, ...
                                  Kp_x,Ki_x,Kd_x, ...
                                  Kp_y,Ki_y,Kd_y, ...
                                  x_ref,y_ref, z_ref, psi_ref, g)
    % Split extended state
    xstate = Xext(1:12);
    iz     = Xext(13); ip = Xext(14); it = Xext(15);
    ips    = Xext(16); ix = Xext(17); iy = Xext(18);
```

```
117
118     dt = 0.01;  % PID update interval
119
120     % Unpack vehicle state
121     x     = xstate(1);  u = xstate(4);
122     y     = xstate(2);  v = xstate(5);
123     z     = xstate(3);  w = xstate(6);
124     phi   = xstate(7);  p = xstate(10);
125     theta = xstate(8);  q = xstate(11);
126     psi   = xstate(9);  r = xstate(12);
127
128     %% Outer loops     attitude refs
129     ex        = x_ref - x;    ix = ix + ex*dt;    dex = -u;
130     Ux        = Kp_x*ex + Ki_x*ix + Kd_x*dex;
131     theta_ref= Ux / g;
132
133     ey        = y_ref - y;    iy = iy + ey*dt;    dey = -v;
134     Uy        = Kp_y*ey + Ki_y*iy + Kd_y*dey;
135     phi_ref   = -Uy / g;
136
137     %% Inner loops      U 1 U4
138     ez    = z_ref - z;    iz = iz + ez*dt;    dez = -w;
139     U1    = Kp_z*ez + Ki_z*iz + Kd_z*dez;
140
141     e_phi = phi_ref - phi;    ip = ip + e_phi*dt;    dep = -p;
142     U2    = Kp_phi*e_phi + Ki_phi*ip + Kd_phi*dep;
143
144     e_th  = theta_ref - theta;    it = it + e_th*dt;    det = -q;
145     U3    = Kp_theta*e_th + Ki_theta*it + Kd_theta*det;
146
147     e_ps  = psi_ref - psi;    ips = ips + e_ps*dt;    deps = -r;
148     U4    = Kp_psi*e_ps + Ki_psi*ips + Kd_psi*deps;
149
150     %% Motor Mixing
151     U    = [U1;U2;U3;U4];
152     f    = Minv * U;   f = max(f, 0);
153     params.omega = sqrt(f);
154
155     % Compute dynamics
156     dx        = quadNonlinearDynamics(t, xstate, params);
157
158     % Integrator derivatives
159     diz  = ez;  dip  = e_phi;  dit  = e_th;
160     dips = e_ps; dix  = ex;     diy  = ey;
```

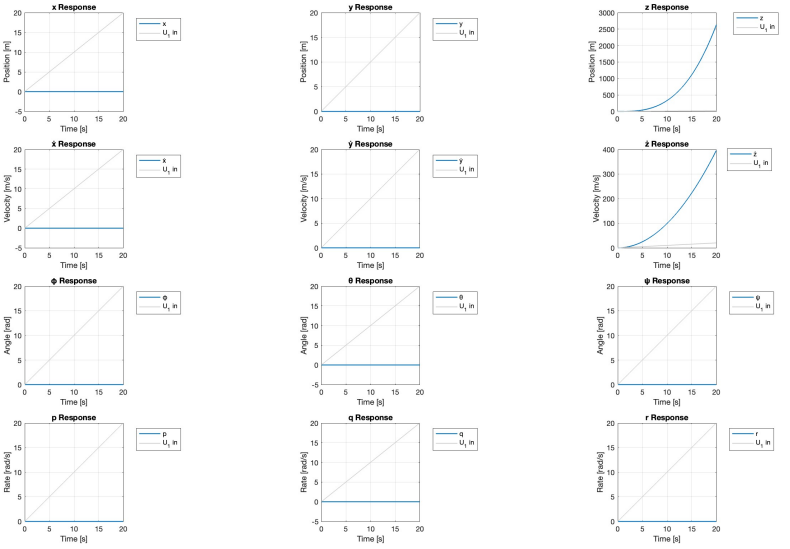
```
161
162     % Pack extended derivative
163     dXext = [dx; diz; dip; dit; dips; dix; diy];
164 end
```

L Additional Analysis Images

Responses to U_1

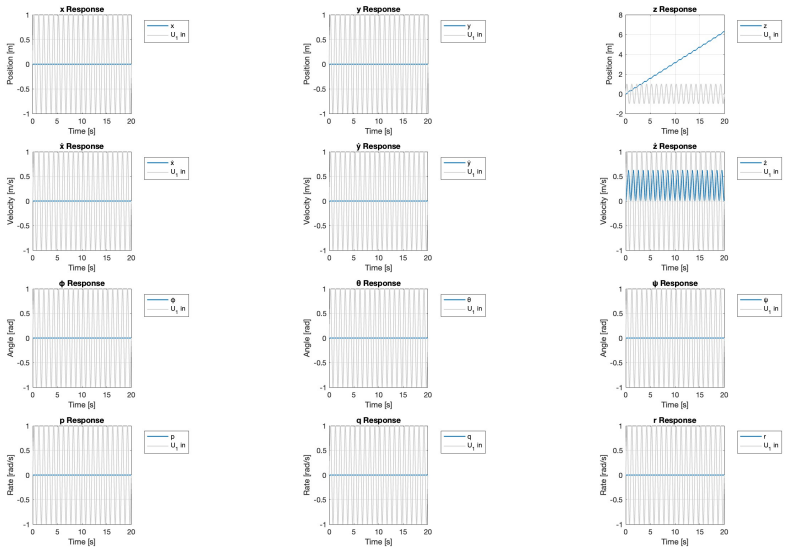


(a)

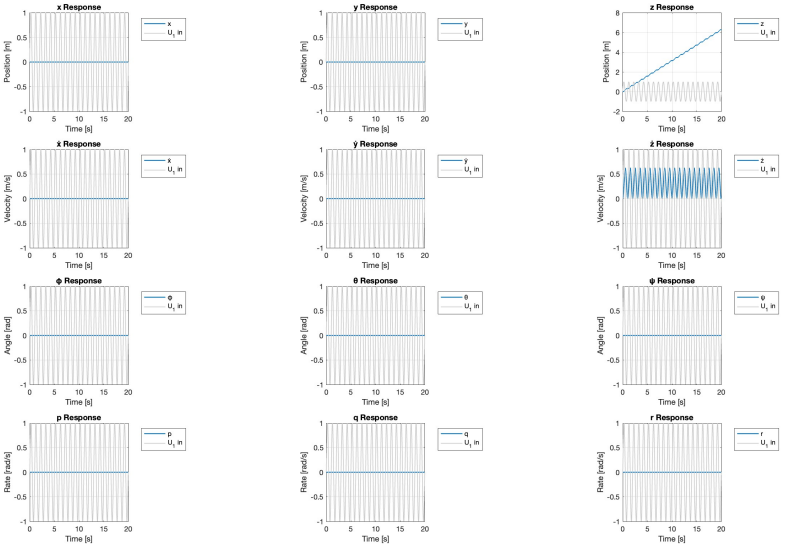


(b)

Figure 28: Linear (a) and Non Linear (b) response to ramp in U_1



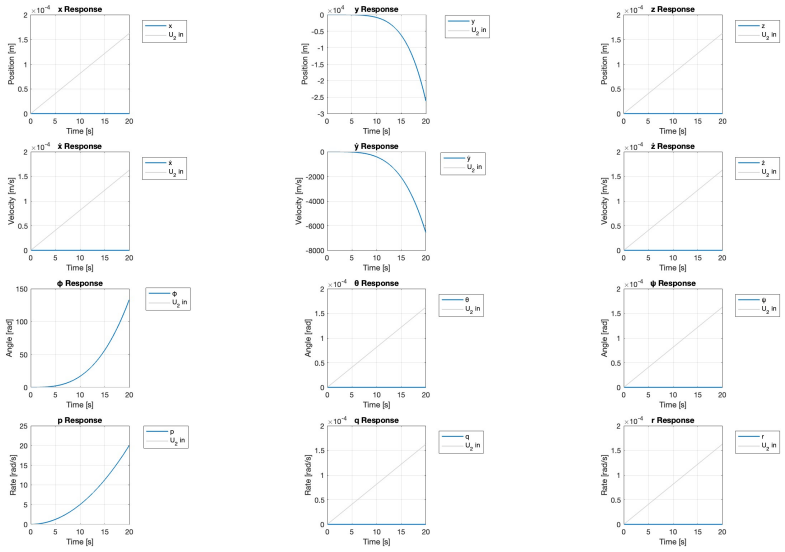
(a)



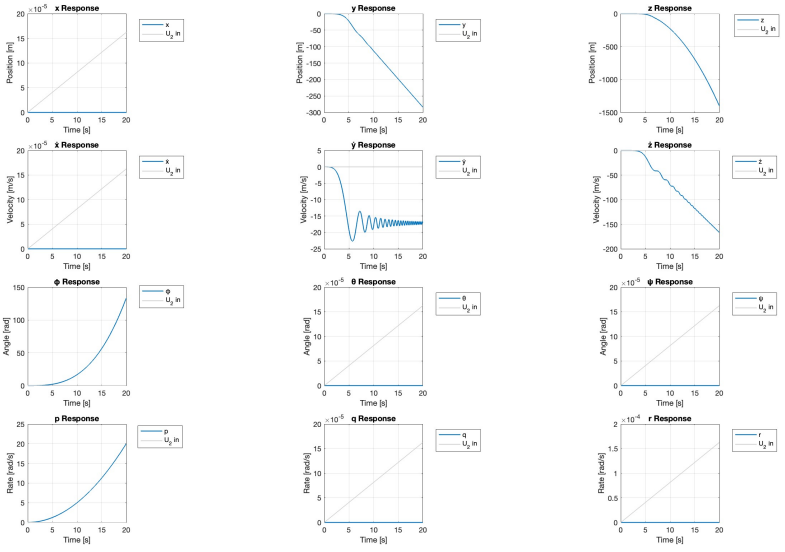
(b)

Figure 29: Linear (a) and Non Linear (b) response to sin in U_1

Responses to U_2



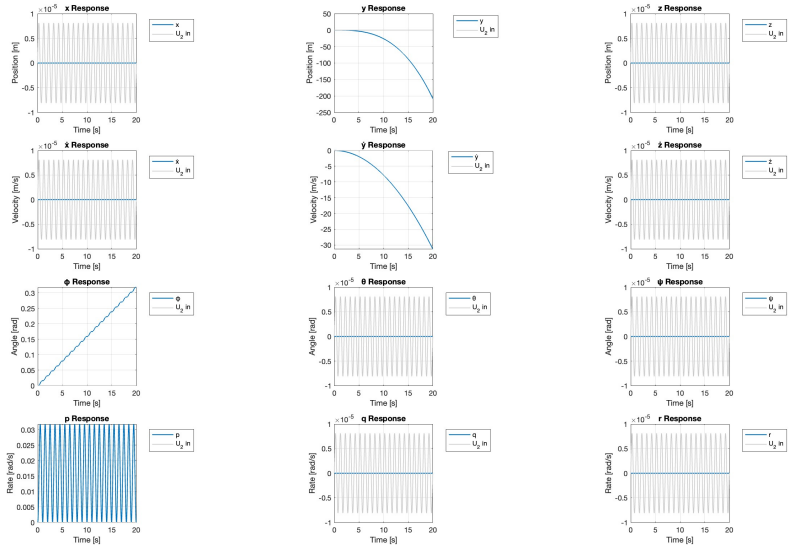
(a)



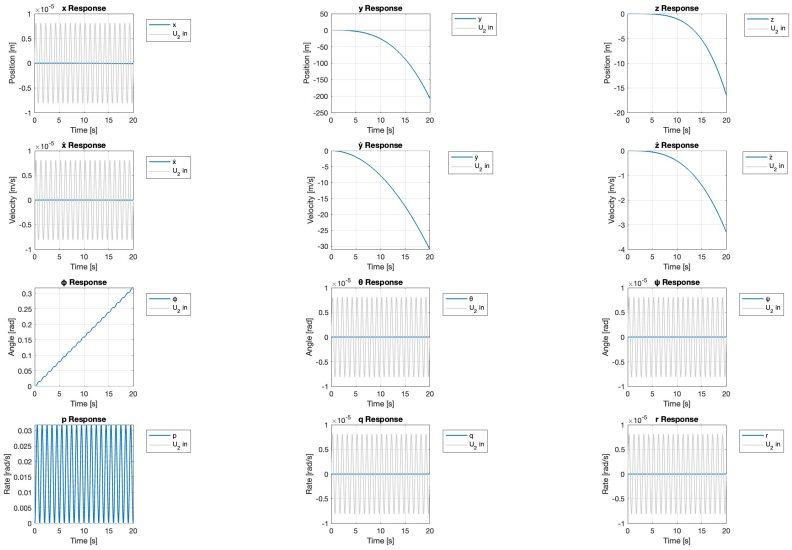
(b)

Figure 30: Linear (a) and Non Linear (b) response to ramp in U_2

Responses to U_4

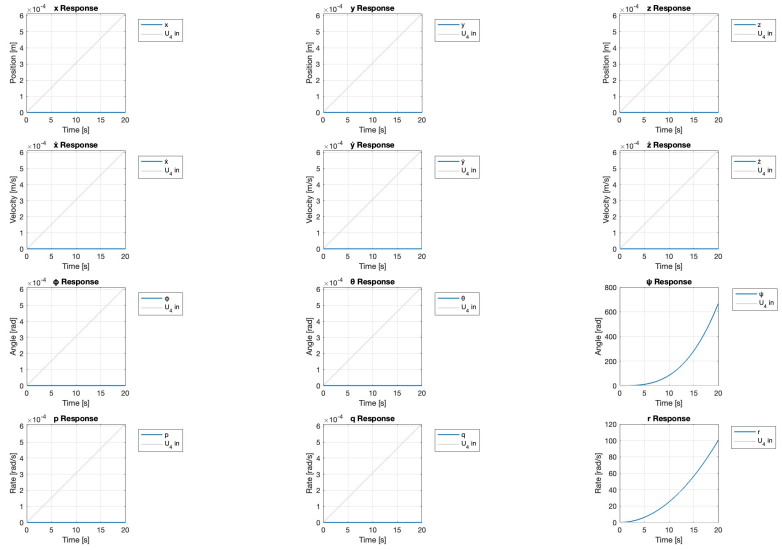


(a)

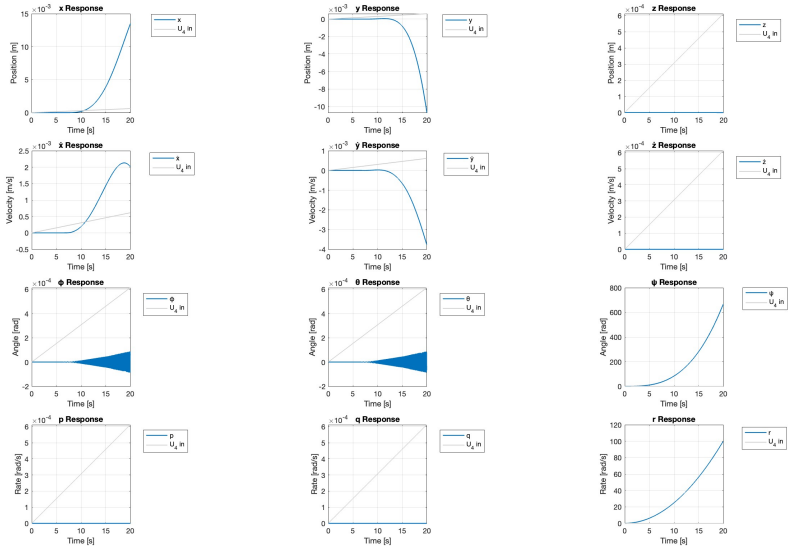


(b)

Figure 31: Linear (a) and Non Linear (b) response to sin in U_2

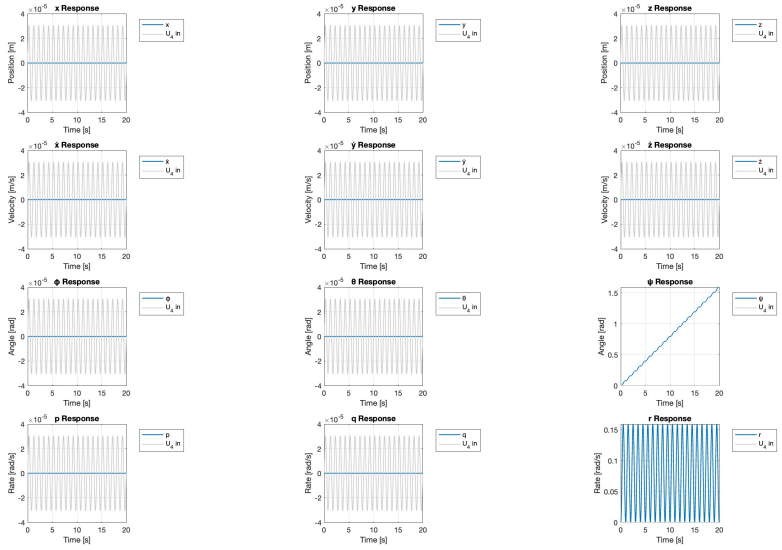


(a)

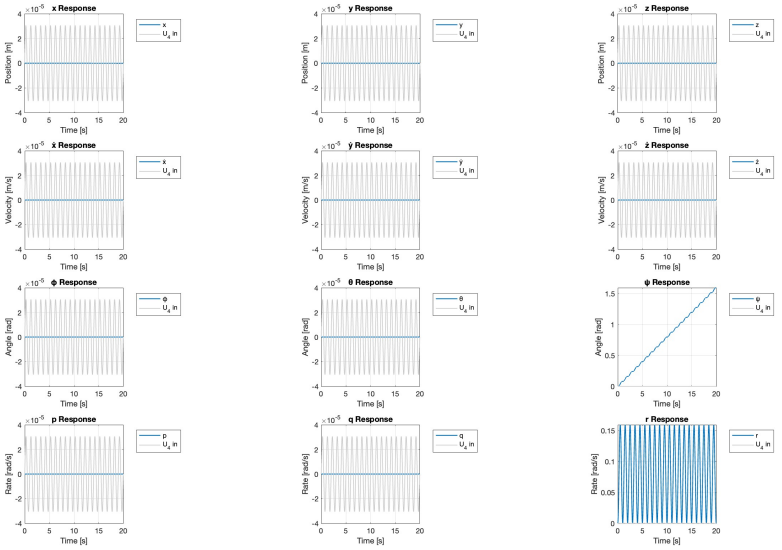


(b)

Figure 32: Linear (a) and Non Linear (b) response to ramp in U_4



(a)



(b)

Figure 33: Linear (a) and Non Linear (b) response to sin in U_4

Appendix M: Hardware Implementation Issues

WITS UNIVERSITY

Tiaan Geldenhuys <2850040@students.wits.ac.za>

Flying the Mambo Drone - 19 May 2025: MECN4029A - Mechatronics II-2025-SM1

1 message

MECN4029A - Mechatronics II-2025-SM1 <notifications@instructure.com> 19 May 2025 at 09:26
Reply-To: reply+ab27cfbbfc61153d-19417~222733048-1747639602@eu.notifications.canvaslms.com
To: 2850040@students.wits.ac.za

Dear Students

Those groups who are flying the drone today - please come to the NWE Lab mezzanine area at 12:30.

Please wear your safety boots.

There will be some set up time, so we may not start exactly at 12:30, but please be there at 12:30.

Also, please bring a laptop + flash drive with you - with the following

- MATLAB/Simulink Installed
- Must include the following toolboxes:
  - Simulink
  - MATLAB Support Package for Parrot Minidrone
  - Simulink Coder
  - Embedded Coder
- Parrot Minidrone Support Package Setup
  - Use matlab.addons.installer or install from Add-Ons > Get Hardware Support Packages.
  - Includes firmware flashing and communication setup.

Your control law needs to be designed already - meaning that your P, I, D gains should be tuned already.

Thank you and see you at 12:30

AP

Replies to this email will be posted as a reply to the announcement, which will be seen by everyone in the course.

CANVAS

WITS UNIVERSITY

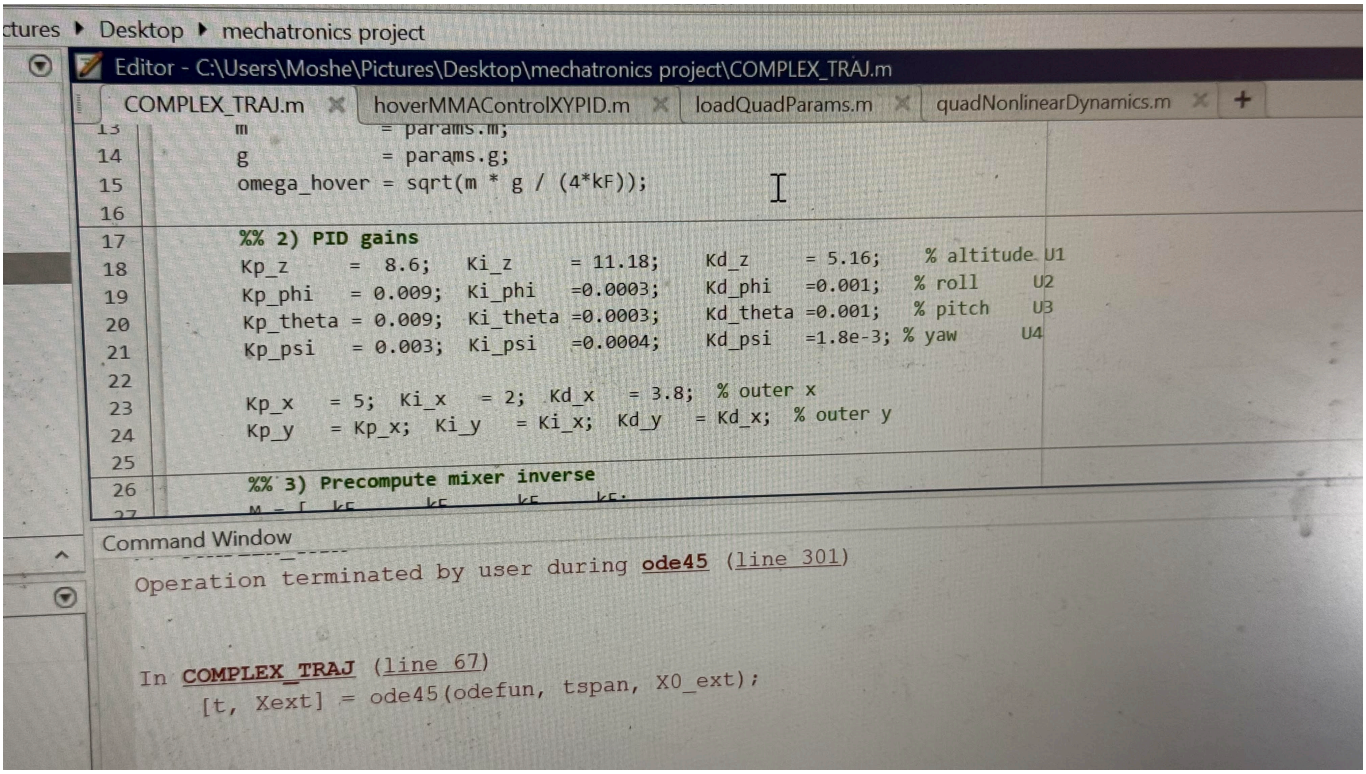
Tiaan Geldenhuys <2850040@students.wits.ac.za>

Fwd: Group 28 PID Values

1 message

Runé Edeling <2400293@students.wits.ac.za> 21 May 2025 at 21:19
To: Tiaan Geldenhuys <2850040@students.wits.ac.za>

---------- Forwarded message ---------
From: **Runé Edeling** <2400293@students.wits.ac.za>
Date: Mon, 19 May 2025 at 14:08
Subject: Group 28 PID Values
To: Aarti Panday <aarti.panday@wits.ac.za>



Flying the Mambo Drone

1 message

Aarti Panday <aarti.panday@wits.ac.za> 19 May 2025 at 20:20
To: Willbes Banda <2109005@students.wits.ac.za>, Tumiso Mokoena <2320283@students.wits.ac.za>, Gumani Mphaga <2129552@students.wits.ac.za>, "2208912@students.wits.ac.za" <2208912@students.wits.ac.za>, Feyitunmise Odedina <2626783@students.wits.ac.za>, Afshan Sabadia <2537452@students.wits.ac.za>, Umar Moola <2538622@students.wits.ac.za>, "2429308@students.wits.ac.za" <2429308@students.wits.ac.za>, Suwilanji Banda <2614685@students.wits.ac.za>, Philani Dlamini <2135214@students.wits.ac.za>, "2540337@students.wits.ac.za" <2540337@students.wits.ac.za>, "2362608@students.wits.ac.za" <2362608@students.wits.ac.za>, Gavriel Dirmeik <2126237@students.wits.ac.za>, Runé Edeling <2400293@students.wits.ac.za>, Tiaan Geldenhuys <2850040@students.wits.ac.za>, Moshe Jacobson <2461271@students.wits.ac.za>

Dear Students

Thank you for coming through today for flying the drone.

Unfortunately, we only managed to get the rotors moving, but did not achieve flight. The drone kept disconnecting, even when fully charged.

So we agreed that for all of you who attended the session with intention to fly the drone (Groups 2, 3, 6 and 28), you can submit the animation file – I am told by some groups that you can fly the bonus scenario in the virtual environment and generate an animation which you can upload in addition to all the other files required.

If any of you saw on Suwilanji's tablet, that what he was doing when we were trying to connect the drone.

Your report needs to include a section that addresses the bonus section in any case, so you can give the details of the animation in the virtual environment.

This is different from showing x,y,z 3D plot of the complex trajectory in the normal simulation.

I am allowing your groups to liaise with each other if any is unsure of the virtual environment and generating the animation.

Thank you!

AP

Dr. Aarti Panday (PhD)

Lecturer

School of Mechanical, Industrial & Aeronautical Engineering

University of the Witwatersrand

South West Engineering Room 110D

011 717 7308

Aarti.Panday@wits.ac.za

WITS UNIVERSITY

Tiaan Geldenhuys <2850040@students.wits.ac.za>

Matlab Outage Group 28

6 messages

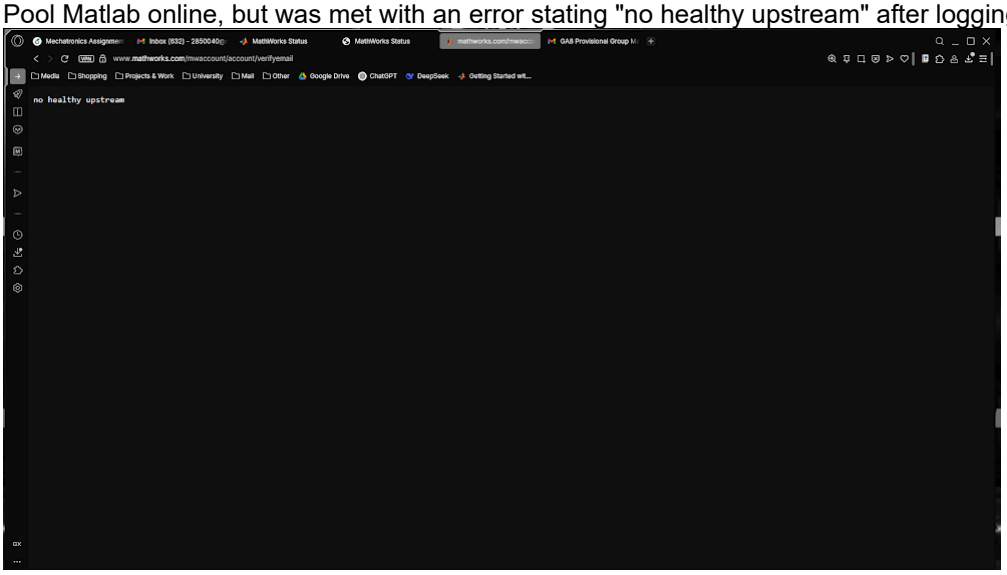
Tiaan Geldenhuys <2850040@students.wits.ac.za> 21 May 2025 at 20:31
To: Aarti Panday <aarti.panday@wits.ac.za>

Good day Dr. Panday

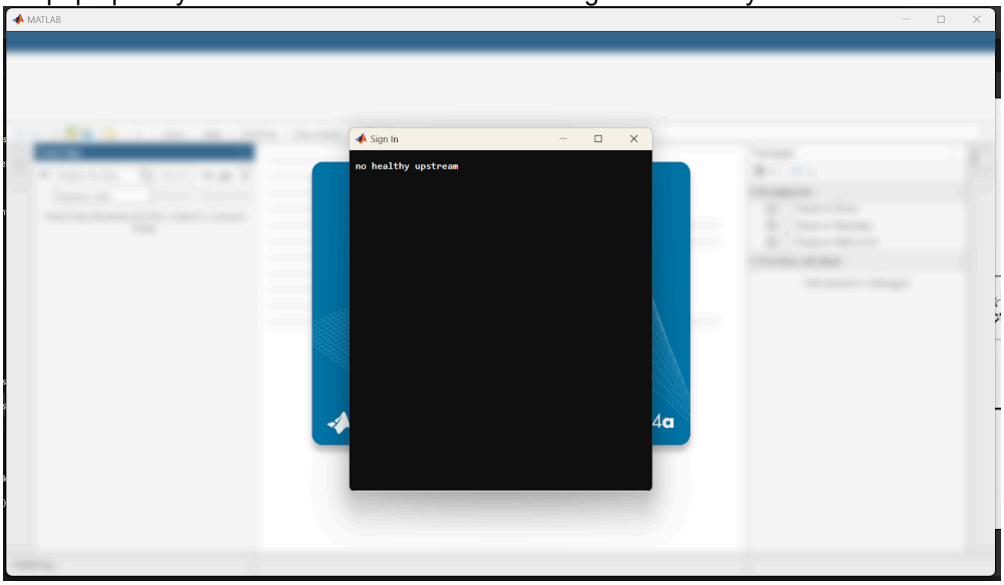
I am writing this email in response to the email you sent Gavi requesting the situation to be documented in detail.

Matlab servers are currently facing a large outage and therefore we cannot generate the video file of the drone being controlled. I have attached a printed pdf view from the official mathworks server status website (labelled "Mathworks Status") where it shows several of its segments are offline, which prevents login, updates and downloads of add-ons. This outage started the 18th of May and has yet to be restored as I am writing this email.

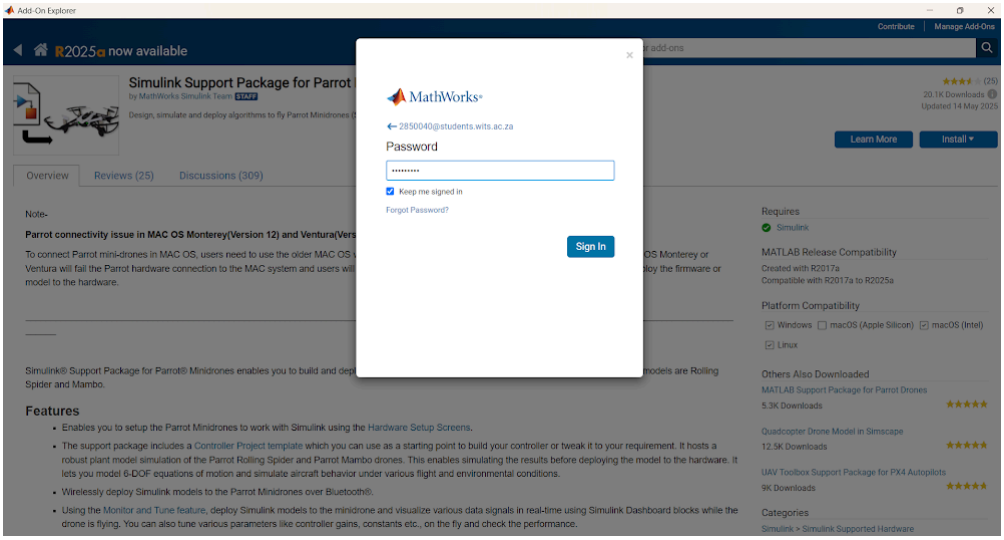
This outage has affected many students worldwide and in our university as students cannot log into matlab online in the PC Pool either. Our group visited the PC pools today as well around 12:30 to 14:00 where we tried to use the PC Pool Matlab online, but was met with an error stating "no healthy upstream" after logging in.

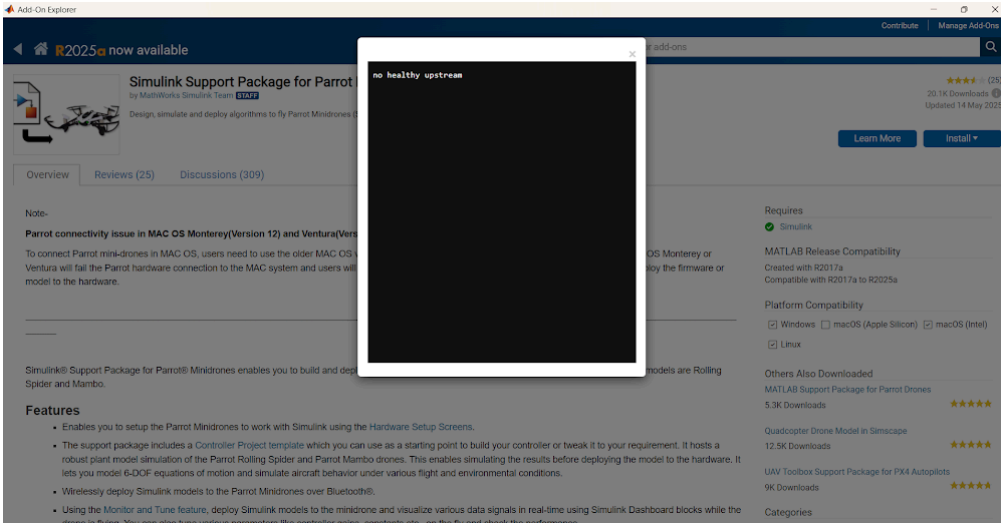


When using Matlab on our personal computer, Matlab would give the same error in a pop up message, where closing the pop up okay would result in the software closing automatically.

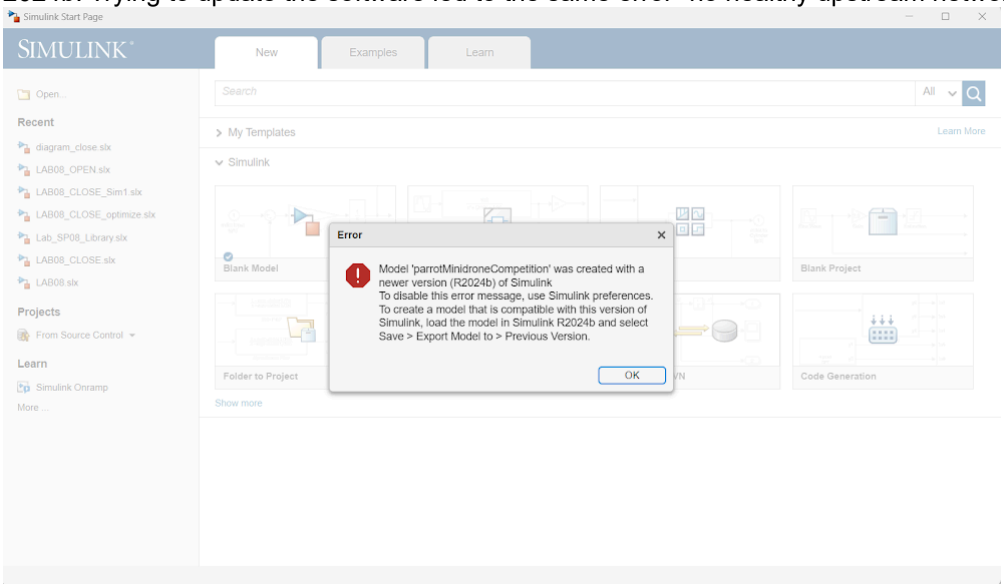


However, switching network off allows one to go into the software, but we could not download or install any of the add ons necessary due matlab requiring sign in to add add ons

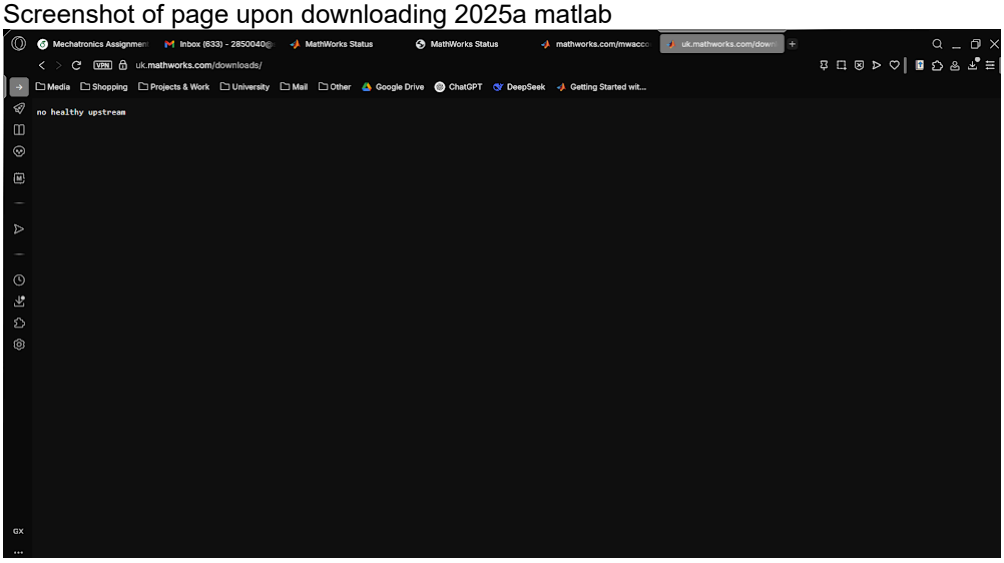




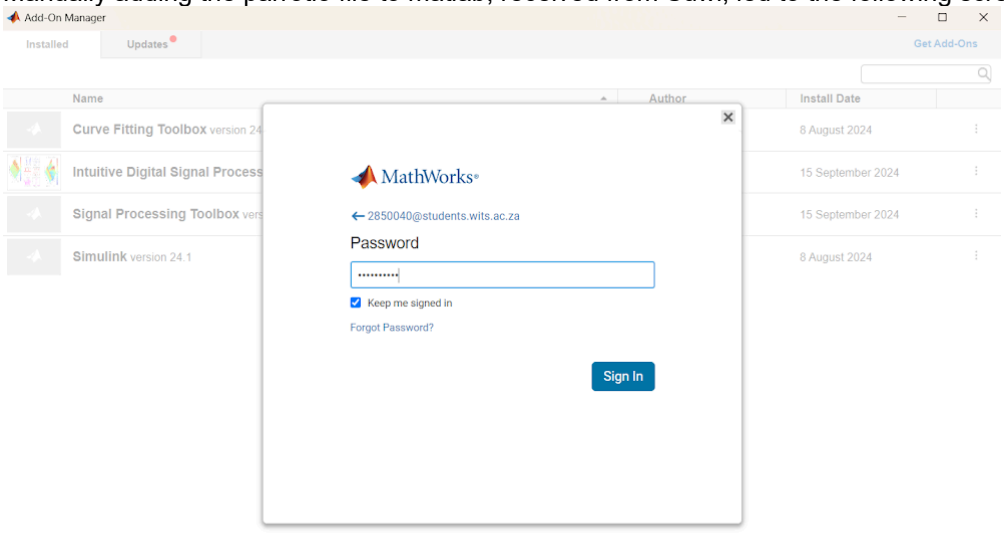
We got the file for the simulink setup from Suwi, but could not open the file due to requiring an update on our matlab software. Only one member had simulink installed but could not open the file since his matlab was 2024a and not 2024b. Trying to update the software led to the same error "no healthy upstream network"

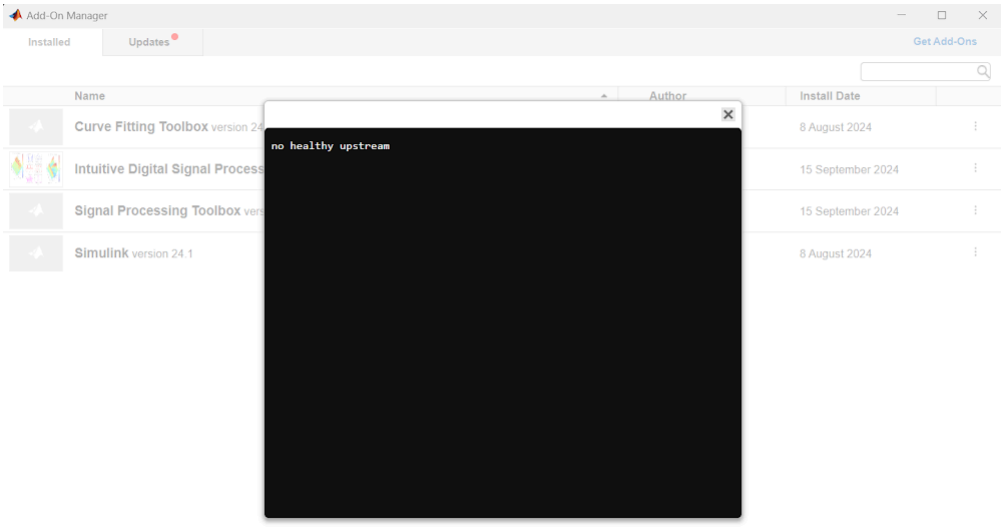


Screenshot of page upon downloading 2025a matlab

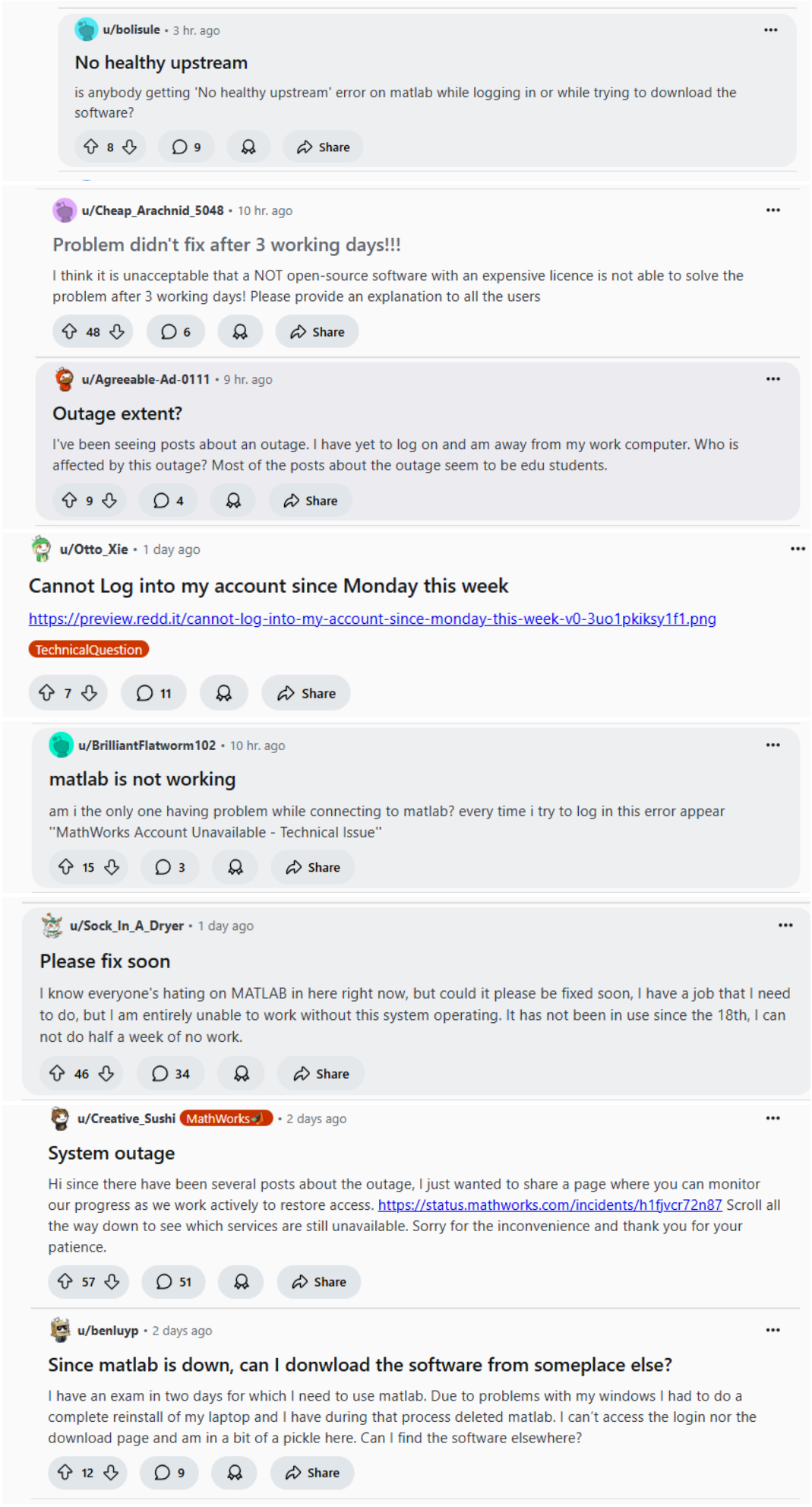


Manually adding the parrotio file to matlab, received from Suwi, led to the following screens





Please see below several people on reddit (https://www.reddit.com/r/matlab/) having the same issues:



With this, I hope that you understand the current situation we are facing and that we cannot do anything about it, therefore we ask for an extension to submit the video file once the servers are up and running again.

Kind regards
Tiaan Geldenhuys

2 attachments

MathWorks Status.pdf
199K

MATLAB programming & numeric computing platform.pdf
3299K

Aarti Panday <aarti.panday@wits.ac.za> 21 May 2025 at 20:38
To: Tiaan Geldenhuys <2850040@students.wits.ac.za>

Hi Tiaan

Just another clarifying question before I give you a way forward…

For your bonus mark case, I saw your group's 3D plot on the demo day on the 19th.

Are you able to generate this plot again for your report?

Please let me know as soon as you can.

Thanks

AP

[Quoted text hidden]

Tiaan Geldenhuys <2850040@students.wits.ac.za> 21 May 2025 at 20:40
To: Aarti Panday <aarti.panday@wits.ac.za>

Yes and it will be in the report, along with the gains for all of the controllers.
[Quoted text hidden]

Aarti Panday <aarti.panday@wits.ac.za> 21 May 2025 at 20:50
To: Tiaan Geldenhuys <2850040@students.wits.ac.za>

Okay great!

Since it's too late in the evening now to confirm an extension on your animation video and you are the only group who has this issue, I would suggest the following to wrap up your submission.

In your report itself in the section where you are presenting your results, you can include a sub-chapter to address the flying (3.19 of the assignment brief).

You can outline all the points – that group 28 was present for the demo on the 19th, Rune sent the lecturer the PID gains on email on the 19th for the flight demo (include Rune's email to the report).

You can add my email of Monday 19 May in this report section to indicate the alternate I offered all groups who attended the flight demo.

Then include everything you have sent me here – the attachments and this email thread.

Put all of this in the report so that we have the complete picture in one place.

From there, I can use that to confirm with the School if I can offer your group the animation extension that you have requested.

[Quoted text hidden]

Tiaan Geldenhuys <2850040@students.wits.ac.za> 21 May 2025 at 21:24
To: Aarti Panday <aarti.panday@wits.ac.za>

Thank you for your understanding, we will do that!
Kind regards
Tiaan
[Quoted text hidden]

Aarti Panday <aarti.panday@wits.ac.za> 21 May 2025 at 21:27
To: Tiaan Geldenhuys <2850040@students.wits.ac.za>

Great stuff

I will also need to check in with Mathworks to get more details.

So you may not get immediate feedback from me while I wait to hear the other details.

Thanks!

[Quoted text hidden]

MathWorks®

Application Status

Contact support

This page displays real-time status and planned maintenance updates for MathWorks applications. If you are experiencing an outage that is not listed, please submit a support request.

MathWorks is currently working on an issue that has affected segments of our online systems.

We will provide further updates as soon as we can.

May 21, 2025 - 13:48 EDT

Issue with multiple applications

Subscribe

Update - Adding ThingSpeak to the list of affected applications.
May 20, 2025 - 10:18 EDT

Update - We are continuing to investigate this issue.
May 19, 2025 - 17:05 EDT

Update - We are continuing to investigate this issue.
May 19, 2025 - 09:48 EDT

Update - We are continuing to investigate this issue.
May 19, 2025 - 09:46 EDT

Update - We are continuing to investigate this issue.
May 19, 2025 - 08:54 EDT

Update - We are continuing to investigate this issue.
May 19, 2025 - 08:28 EDT

Update - We are continuing to investigate this ongoing issue. We will provide updates as they become available.
May 18, 2025 - 23:41 EDT

Update - We are continuing to investigate this issue.
May 18, 2025 - 20:51 EDT

Update - We are continuing to investigate this issue.
May 18, 2025 - 19:54 EDT

Investigating - There is an issue with multiple applications. We are assessing impact and investigating the cause.
May 18, 2025 - 19:54 EDT

Add-On Explorer	⚠ Degradation
Careers at MathWorks	✖ Outage
Cloud Center	✔ Operating normally
Cody	✖ Outage
Downloads	✖ Outage
File Exchange	✖ Outage
License Center	✖ Outage
Online Licensing	✔ Operating normally
MathWorks.com	⚠ Degradation
MathWorks Store	✖ Outage
Online Courses App	✔ Operating normally
MATLAB and Simulink Course Schedule	✖ Outage
MATLAB Answers	⚠ Degradation
MATLAB Copilot	✔ Operating normally
MATLAB Drive	✔ Operating normally
MATLAB Grader	✖ Outage
MATLAB Grader - LTI® Service	✔ Operating normally
MATLAB Mobile	✔ Operating normally
MATLAB Online	⚠ Degradation
ThingSpeak	⚠ Degradation

Past Incidents

May 20, 2025

Unresolved incident: Issue with multiple applications.

May 19, 2025

May 18, 2025

← Incident History

Powered by Atlassian Statuspage

reddit

r/matlab Search in r/matlab

Home
Popular
Explore
All

CUSTOM FEEDS
Create a custom feed

RECENT
r/matlab
r/JurassicWorldApp
r/AppleWatch
r/functionalprint

COMMUNITIES
Create a community
r/announcements
r/AppleWatch
r/Dinosaurs
r/discordapp

r/matlab

Create Post | Join

Best

Community highlights

Submitting Homework questions? Read this
191 votes • 26 comments

If you paste ChatGPT output into posts or comments, please say it's from...
96 votes • 0 comments

u/Cheap_Arachnid_5048 • 10 hr. ago

Problem didn't fix after 3 working days!!!

I think it is unacceptable that a NOT open-source software with an expensive licence is not able to solve the problem after 3 working days! Please provide an explanation to all the users

48 | 6 | Share

u/Agreeable-Ad-0111 • 9 hr. ago

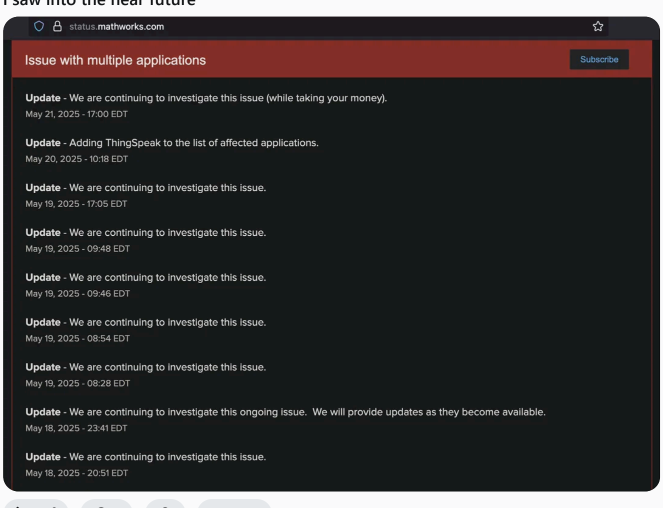
Outage extent?

I've been seeing posts about an outage. I have yet to log on and am away from my work computer. Who is affected by this outage? Most of the posts about the outage seem to be edu students.

9 | 4 | Share

u/Weary_Primary3410 • 3 hr. ago

I saw into the near future



8 | 2 | Share

u/Round_Historian_6262 • 2 days ago

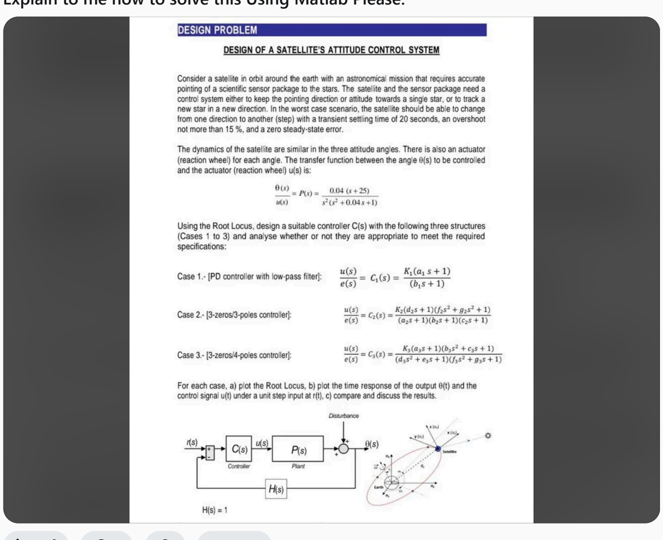
Is MatLab Reliable?

I've only been trying to start teaching myself MatLab in the past 24 hours, but because of the outage that started yesterday, I am unable to. I noticed that it had the same outage on May 15th, how often does MatLab crash and is it a reliable platform?

8 | 19 | Share

u/Small-momo • 22 min. ago

Explain to me how to solve this Using Matlab Please.



2 | 0 | Share

u/NoGeologist3815 • 6 hr. ago

Workaround for Current MathWorks Outage

If you currently face problems starting Matlab, this might help you. Disconnect from the internet and kill the "mathwokrs service hosts taks" in task manager. Then try start Matlab again. For me it started without problems.

14 | 1 | Share

u/mikkej • 15 hr. ago

Matlab 2025a is now out with Web UI

https://www.mathworks.com/products/new_products/latest_features.html

8 | 9 | Share

u/bolisule • 3 hr. ago

No healthy upstream

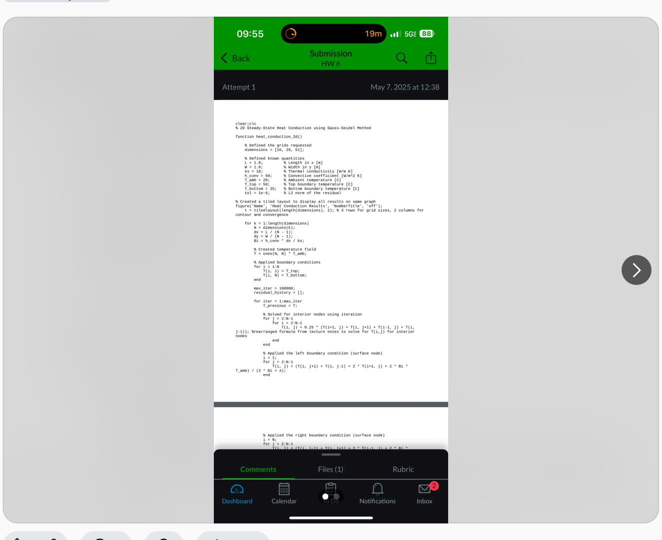
is anybody getting 'No healthy upstream' error on matlab while logging in or while trying to download the software?

8 | 9 | Share

u/DeathKnight05 • 5 days ago

Code Error

HomeworkQuestion

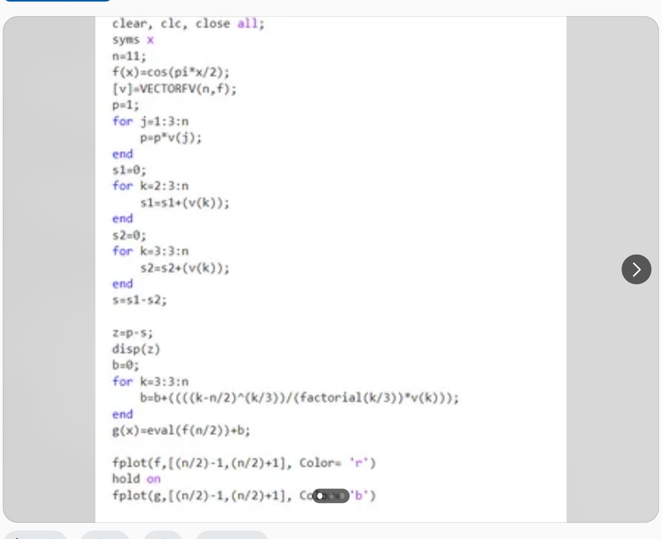


7 | 7 | Share

u/zamil1000 • 51 min. ago

I don't understand what error it shows

HomeworkQuestion



2 | 1 | Share

u/Otto_Xie • 1 day ago

Cannot Log into my account since Monday this week

https://preview.redd.it/cannot-log-into-my-account-since-monday-this-week-v0-3uo1pkiksy1f1.png

TechnicalQuestion

7 | 11 | Share

u/HaveUSaidThankU • 2 days ago

Can I use personal license to develop and publish free application, then buy another commercial license before releasing a paid version

I am developing a MATLAB application. I plan to release it as a free version initially, then make another paid version if I can accumulate enough users. I haven't registered my company yet. So I assume it would be ok if I publish the free version when I'm still using the home license. I plan to register a company and buy a standard or startup license later if I get positive feedbacks.

Misc

6 | 1 | Share

u/BrilliantFlatworm102 • 10 hr. ago

matlab is not working

am i the only one having problem while connecting to matlab? every time i try to log in this error appear ''MathWorks Account Unavailable - Technical Issue''

15 | 3 | Share

u/EngineEngine • 2 hr. ago

Trying to average every four rows. It requires me to average the datetime column separately, and I can't add it back into the the newly-created tabled of averages

CodeShare

2 | 3 | Share

u/Hot_Active_6521 • 6 days ago

traffic simulation at red light junction

hELLO I'm trying to do this but I'm having a hard time as a matlab beginner. I imagine there are several ways to get there.

4 | 1 | Share

u/Adept-Gene-4661 • 1 hr. ago

a funny way to access matlab document

I'm not sure which service is you need, but I can access the helper document by my pad now. Although search functhion still not well. So I use a very very very funny way to continue my work： I search the specific helper page on bing by my pad，and copy useful info to my computer🤡 If you have urgency you can imitate 🤡 just like me.

1 | 0 | Share

u/Ok_Weekend_3637 • 1 day ago

Creating a graph like this for glmm in Matlab?

Hi, I am usually an R-user, but apparently fitting a GLMM with maximum pseudolikelihood is exclusive to Matlab (my PI's language). While it is relatively easy to plot the model predictions in R, it is proving to be hellish in Matlab, and I am finding minimum documentation to help me with this. Even AI is proving pretty unhelpful, but I am sure that someone has done this before. What I am looking for is a graph with the response as the y-axis, one of the predictors as the x-axis, and two sets of lines (one for each level). Basically I am looking for this: https://preview.redd.it/creating-a-graph-like-this-for-glmm-in-matlab-v0-f8mtn1svzx1f1.png I have already spe...

3 | 2 | Share

u/DueGrocery1152 • 2 days ago

POLYNOMIAL FITTING

TechnicalQuestion

3 | 3 | Share

u/Ready_Ask1771 • 5 hr. ago

pso-flc hybrid mppt for pv system

hi does anyone here is particularly good in this topic or matlab overall? i need help for my project

1 | 0 | Share

u/Sock_In_A_Dryer • 1 day ago

Please fix soon

I know everyone's hating on MATLAB in here right now, but could it please be fixed soon, I have a job that I need to do, but I am entirely unable to work without this system operating. It has not been in use since the 18th, I can not do half a week of no work.

46 | 34 | Share

u/AirlineStunning4896 • 2 days ago

Advice Needed: Best Practice for Generating Realistic Synthetic Biomedical Data in MATLAB (rand vs randi)

Hi all, I'm generating a synthetic dataset in MATLAB for a biomedical MLP classifier (200 samples, 4 features: age, heart rate, systolic BP, cholesterol). Should I use `rand()` (scaled) or `randi()` for generating values in realistic clinical ranges? I want the data to look plausible—e.g., cholesterol = 174.5, not just integers. Would `randn()` with bounding be better to simulate physiological variability? Thanks for any advice!

3 | 3 | Share

u/OneKaleidoscope9858 • 6 hr. ago

Matlab code homework

Hello! I am studying bioengineering and I had to take an IT class first semester. Due to health issues I was unable to take the exam. I am now in fourth year, my thesis is due next week and I just got an email from administration that I have two days to close my IT debt. I have to submit some assignments and I feel so lost and I just don't have time to do it all. I really really need help cause I don't know where to start, some guidance would be sooo appreciated. this is what the professor sent: Write a Matlab program that:

1. converts a number of radix $r \in \{2, 3, 4, \ldots, 16\}$ to the decimal number....

HomeworkQuestion

1 | 2 | Share

u/hdiyad • 6 days ago

PWM signal goes to zero when using PID controller after MPPT (P&O) in PV system

2 | 19 | Share

u/Bread_is_life_17 • 21 hr. ago

Workaround for "no healthy upstream" error when validating licence

Hello everyone, On Sunday, my local copy of matlab requested to validate the licence. Since the outage, I was getting the error 'no healthy upstream' (very bad timing for us people that use matlab for work). One workaround that work for me to avoid this problem was: Disconnect from the internet -> Open matlab session (matlab did not ask for licence validation and it just works) -> Connect to the internet again. Hopefully the tip it is useful for someone

21 | 3 | Share

u/Creative_Sushi MathWorks • 2 days ago

System outage

Hi since there have been several posts about the outage, I just wanted to share a page where you can monitor our progress as we work actively to restore access. https://status.mathworks.com/incidents/h1fjvcr72n87 Scroll all the way down to see which services are still unavailable. Sorry for the inconvenience and thank you for your patience.

57 | 51 | Share

u/benluyp • 2 days ago

Since matlab is down, can I donwload the software from someplace else?

I have an exam in two days for which I need to use matlab. Due to problems with my windows I had to do a complete reinstall of my laptop and I have during that process deleted matlab. I can't access the login nor the download page and am in a bit of a pickle here. Can I find the software elsewhere?

12 | 9 | Share

u/hcarlsso • 1 day ago

How to disable Navigation Keyboard shortcuts in 2025a?

The problem is that I am using the emacs keyboard shortcuts, but everytime I press the Alt key (for copying) this toolbar appears: https://preview.redd.it/how-to-disable-navigation-keyboard-shortcuts-in-2025a-v0-0hs3rap5fy1f1.png I have tried to disable it with the command: com.mathworks.desktop.mnemonics.MnemonicsManagers.get.disable But does not work. So how do I disable these Navigation Keyboard shortcuts? https://preview.redd.it/how-to-disable-navigation-keyboard-shortcuts-in-2025a-v0-yh6o5ygzey1f1.png

2 | 2 | Share

u/Acceptable_Pin_9111 • 3 days ago

is anyonehaving trouble logging in due to a tech error? when trying to do other things I get the unhealthy upstream message. Should I just revisit mathworks tomorrow?

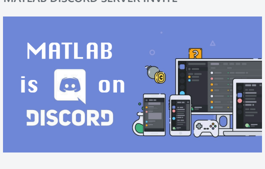
33 | 31 | Share

MATLAB programming & numeric computing platform
Official MATLAB subreddit
Created Aug 15, 2009
Public

68K coders | 36 logged in | Top 3% Rank by size

USER FLAIR
Virtual-Wonder9287

MATLAB DISCORD SERVER INVITE



MODERATORS
Message Mods
u/virga Solves for zero
u/Mimshot L = @(r) x.*log(r)-r
u/Weed_O_Whirler +5
u/chartporn r/MATLAB