Aeronautics & Astronautics

Samuel Gilonis

18th March 2014

Third Year Individual Project

IP Number: 960
IP Title: Low-complexity controller design for an unmanned helicopter
Supervisor: Dr. Zhan Shu
Assessor: XXX

Individual Project Academic Integrity Statement

Declaration,

I, the undersigned, confirm that the material presented in this project report is all my own work. References to, quotations from, and the discussion of work of any other person have been correctly acknowledged/cited within the report in accordance with University of Southampton guidelines on academic integrity.

Name: _____

Signed: _____ Date: _____

Contents

1 Nomenclature

A	Rotor disk area
B	Constant relating drag force to square of angular velocity
C_D	Coefficient of drag
D_i	Drag generated on the i^{th} propeller
e_ϕ	Error in roll angle
e_θ	Error in pitch angle
e_ψ	Error in yaw angle
e_x	Error in x-coordinate
e_y	Error in y-coordinate
e_z	Error in z-coordinate
f	Forces
$\vec{F_D} = \begin{bmatrix} -C_i\dot{x} & -C_j\dot{y} & -C_k\dot{z} \end{bmatrix}$	Drag vector
$\digamma_B = \left\{O_B, \hat{i_B}, \hat{j_B}, \hat{k_B}\right\}$	Body-fixed frame
$\digamma_I = \left\{O_I, \hat{i_I}, \hat{j_I}, \hat{k_I}\right\}$	Earth-fixed inertial frame
H_∞	H-infinity control
I	Inertia matrix
I	Input current for motor
I_0	Current when there is no load on the motor
I_3	3×3 Identity matrix
K	Constant relating thrust to square of angular velocity
K_d	Proportional gain tuning parameter
K_i	Derivative gain tuning parameter
K_p	Integral gain tuning parameter
K_t	Torque proportionality constant
K_v	Proportionality constant relating back EMF to RPM
K_τ	Proportionality constant relating thrust to torque
LQR	Linear Quadratic Regulator
m	Quadcopter mass
PID	Proportional Integral
P	Power consumed by the motor
$\vec{p} = \begin{bmatrix} p_i & p_j & p_k \end{bmatrix}^T$	Arbitrary position vector
R	Rotation matrix
R	Propeller blade length
R_m	Motor resistance
$\vec{T}_B = \begin{bmatrix} 0 & 0 & T_k^B \end{bmatrix}$	Thrust vector in $\digamma_B$
$\vec{T}_I = \begin{bmatrix} T_i^I & T_j^I & T_k^I \end{bmatrix}$	Thrust vector in $\digamma_I$
T_i	Thrust generated by the i^{th} propeller
$u(t)$	Output from controller
UAV	Unmanned Aerial Vehicle

V	Voltage across motor
$\vec{v}^I = \begin{bmatrix} u^I & v^I & q^I \end{bmatrix}^T$	Velocity vector in $\digamma_I$
$\vec{v}^B = \begin{bmatrix} u^B & v^B & w^B \end{bmatrix}^T$	Velocity vector in $\digamma_B$
θ	Pitch angle
$\Theta = [\phi, \theta, \psi]^T$	Orientation vector
ν, v_i	Induced velocity
ρ	Air density
τ	Torques
τ_D	Torque due to drag
τ_m	Motor torque
$\xi = [x\ y\ z]^T$	Position vector
ϕ	Roll angle
ψ	Yaw angle
Ψ	Orientation dynamics rotation matrix
$\omega^B = \begin{bmatrix} p & q & r \end{bmatrix}^T$	Angular velocity vector
Ω	Propeller blade angular velocity
Ω_r	Residual propeller blade angular velocity

List of Figures

List of Tables

2 Introduction

There has been escalating interest in Unmanned Aerial Vehicles (UAVs) in recent years, largely stemming from their controversial military applications but there has also from their potential use in commercial and civil arenas. In the 1990s the need for UAVs was outlined in [1] by Captain B. Tice of the United States Air Force as being for all missions that were "Dirty, Dull or Dangerous". Though he was correct, the Captain may have been unnecessarily narrow in this definition as the UAVs are just as relevant for dirty, dull and dangerous jobs outside of a military context. NASA compiled a survey [2] of possible applications for UAVs in a civil context in which they anticipated that these UAVs, or 'drones', would be used in transport, exploration, scientific research, search and rescue, environmental conservation, policing, surveillance, fire fighting, disaster response and photography. At the time of writing the online retailer Amazon have just announced plans for a delivery service based upon the use of octocopters.

The majority of drones at the moment are fixed wing aircraft but rotorcraft (helicopters) have certain obvious advantages over fixed wing aircraft such as vertical take-off and landing (VTOL) and the ability to hover and fly at low altitudes. These advantages make rotorcraft drones an area of interest as while it would be impossible to inspect a bridge, for example, with a fixed wing drone, with a small unmanned helicopter this would be a trivial exercise.

This should provide some insight into the practical applications of unmanned rotorcraft but it is worth noting that they are of interest in a more academic sense as well. As stated in [3, p.vii], helicopters are high order, highly unstable, underactuated, nonlinear systems with dynamic coupling between the state variables and control inputs and therefore pose a significant challenge from a control perspective.

Helicopter dynamics are complicated enough so that they are still not fully understood and are therefore approximated when constructing a model for control purposes. This project will therefore have three parts: first of all a simple model for the rotorcraft dynamics must be derived; secondly, various low-complexity controllers will be applied to see which offers the greatest stability and performance; thirdly, some kind of graphical representation of the rotorcraft's motion will be created in order to visualize stability and controller efficacy.

3 Aims

The aims of this project are:

- Derive a mathematical model for helicopter dynamics.
- Design a low complexity controller that will keep the helicopter stable and allow it to follow a defined trajectory.
- Test the controller using a simulation in MatLab.
- If possible, to design a 3D visualization of the helicopters motion in MatLab.

4 Rotor configurations

There is a multiplicity of configurations for helicopters that all seek to solve the problem of torque generated by a lift-generating rotor. The starting point in helicopter design is to use a single rotor to

generate lift but this rotor will exert a torque on the main body of the helicopter (by Newton's Third Law, for every reaction there is an equal and opposite reaction) and therefore cause the helicopter to spin. To counteract this effect, additional rotors can be added that spin in the opposite direction, or a tail rotor can be added that exerts a force perpendicular to that of the main rotor. As these configurations entail different mathematical models, the first decision in this project is to select a 'family' of helicopters to model. The various configurations and the problem of counteracting rotor-torque reaction are described in [4, pp. 3-6]. The rotor configurations can be seen in Figure 1.

4.1 Main rotor and tail rotor

As described above, the traditional helicopter configuration uses a main rotor to generate lift and a tail rotor which exerts a perpendicular force for yaw control and directional stability. In these helicopters there will be a swash-plate mechanism that allows the pitch of the blades to be altered either collectively or cyclically in order to generate vertical propulsion or tilt the thrust vector and cause the helicopter to engage in translational motion.

4.2 Tandem rotors

This configuration positions two counter rotating rotors, one in front of the other. As discussed in [4, p.106], the two rotors rotate at an equal angular velocity and therefore the torque generated by each cancels the other out. The yaw of the helicopter is controlled by increasing the angular velocity of one of the rotors so that it generates a greater torque than the other. This configuration also has swash-plate mechanisms and can therefore tilt the thrust vector by a cyclic control of the blade pitch. Tandem rotors have extremely complicated aerodynamics features as there is a great deal of aerodynamic interaction between the two rotors as well as between the rotors and the fuselage.

4.3 Transverse rotors

Similar to the tandem configuration, the transverse rotor setup uses two counter rotating rotors to neutralise the torque, however, the transverse rotor positions the two rotors side-by-side with each being attached to some form of fixed wing. As with the tandem setup this has the advantage of being able to use all of the power from the engine to generate lift rather than having to divert power to counter the torque.

4.4 Coaxial rotors

The coaxial configuration, discussed in [6, p.101] has two counter rotating rotors with one on top of the other. Though this is a more stable configuration than some others, this comes at a cost given the high mechanical complexity of the hub.

4.5 Intermeshing rotors

This configuration has two counter rotating rotors at angles from the fuselage that are synchronised so that the blades do not collide. Yaw is achieved by increasing the angular velocity of one of the rotors which obviates the need for a tail rotor but the rotors are tilted and therefore not directly producing vertical thrust, which reduces efficiency.

4.6 Multirotor

Rotorcraft of this configuration utilise two or more rotors (typically four, six or eight - called quadcopters, hexacopters and octocopters respectively). Here the same principle of counter rotating rotors is used in order to negate torque. Rather than using variable pitch blades in order to tilt the thrust vector and alter the direction of flight, these multicopters generally alter the lift generated by some of the propellers in order to tilt the rotorcraft and therefore the thrust vector.

Early on in the project the decision was made to focus on a quadcopter configuration because quadcopters, while highly unstable, are mechanically simple, are approximately symmetrical across all three axes, do not have significant rotor-fuselage interaction and do not need variable pitch blades in order to navigate all 6 degrees of freedom. These qualities make the quadcopter simpler to model and simulate while still giving insight into the problems of helicopter control.

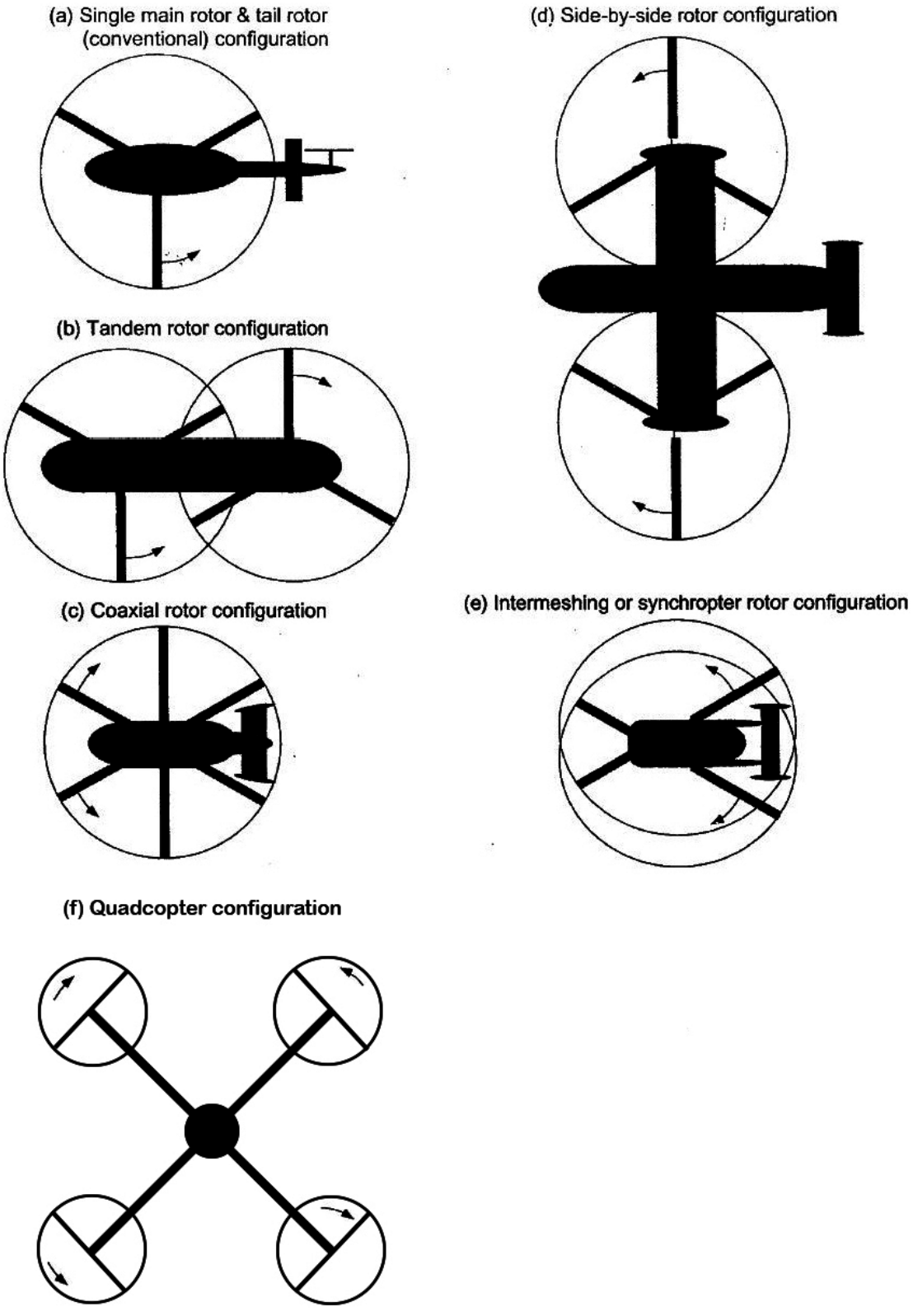


Figure 1: Rotor configurations (a-e taken from [4])

5 Quadcopter Kinematics

5.1 Position and Orientation of the Quadcopter

Numerous models for quadcopter motion have been derived in recent years. The first step is to define two frames, an inertial one that the quadcopter's motion will be measured as being relative to and one that is fixed to the quadcopter body. According to [8], these frames can be described as "a point in space and three orthonormal vectors that form a basis". The first 'inertial' or 'Earth-fixed' frame, F_I, can be described by $\left\{O_I, \hat{i_I}, \hat{j_I}, \hat{k_I}\right\}$ where O_I represents some point that we can call the origin, $\hat{i_I}$represents North, $\hat{j_I}$ represents East and $\hat{k_I}$represents 'up' in the North-East-Down convention. The 'body-fixed' frame, F_B, can be described by $\left\{O_B, \hat{i_B}, \hat{j_B}, \hat{k_B}\right\}$ where O_B represents the intersection of the two quadcopter arms, $\hat{i_B}$ falls along one of the quadcopter arms, $\hat{j_B}$ falls along a quadcopter arm perpendicular to $\hat{i_B}$ and $\hat{k_B}$is normal to $\hat{i_B}$ and $\hat{j_B}$ and therefore is the direction of the thrust vector in F_B when all of the propellers are generating equal thrust.

It is necessary to define these frames so that we can derive a way of mapping vectors from one to the other. For example, while it is trivial to calculate the thrust vector with respect to F_B, it is more difficult to do so with respect to F_I. We need to know the thrust in F_I as this will tell us about the acceleration of the quadcopter with respect to some inertial point rather than with respect to the quadcopter itself which would be meaningless. It is therefore convenient to derive a rotation matrix that would map vectors from F_B to F_I.

First, some method for describing the orientation of F_B must be achieved. This is typically done using the Euler angles: roll, pitch and yaw denoted by ϕ, θ, and ψ respectively. If F_B is at some orientation described by $\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}$ then it will require three rotations of F_B to bring it into alignment with F_I. Each rotation will define a new intermediary frame and we will call these frames F_3, F_2 and F_1. This derivation is similar but not identical to the derivation found in [3, p.25] and aims to provide more detail.

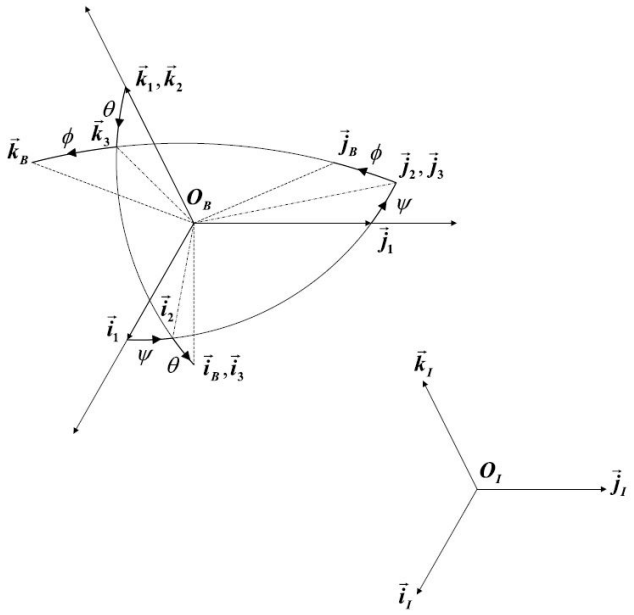


Figure 2: F_I and F_B and the rotations necessary to get between them. [8]

5.2 Position, Velocity, Orientation and Angular Velocity Vectors

It is useful to define the position and velocity vectors with respect to F_I.

The position vector is given by:

$$\vec{\xi} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1}$$

The velocity vector is given by:

$$\vec{v^I} = \begin{bmatrix} u^I \\ v^I \\ w^I \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} \tag{2}$$

The orientation is given by:

$$\Theta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{3}$$

The time derivatives of roll pitch and yaw are given by:

$$\dot{\Theta} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{4}$$

Angular velocity is given by:

$$\omega^B = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{5}$$

5.3 Position Dynamics

5.3.1 Mapping from F_B to F_3

We can posit a position vector $\vec{p}$ that is given by $\vec{p} = \begin{bmatrix} p_i \\ p_j \\ p_k \end{bmatrix}$

This vector could be given with respect to F_I or F_B or any other frame. This position vector will be used to demonstrate how the rotation matrices are obtained.

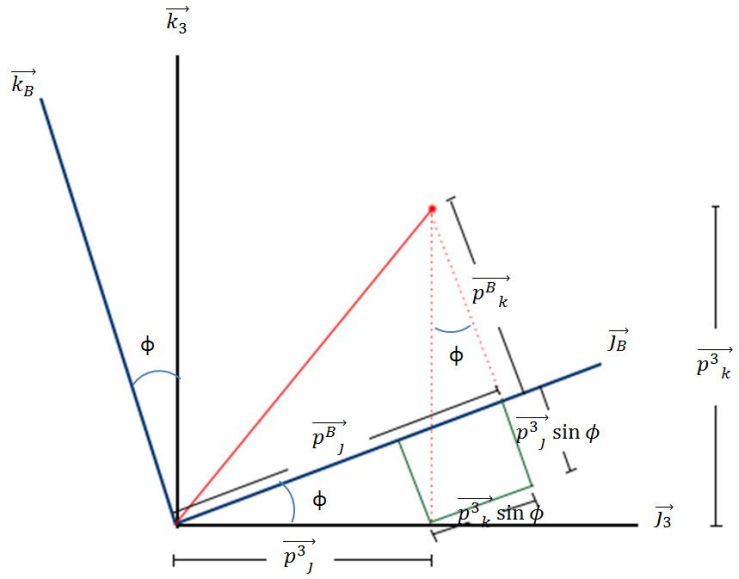


Figure 3: Mapping from F_B to F_3

The first rotation is to rotate F_B about $\hat{i_B}$ by and angle ϕ. This can be seen in Figure 3 where the vectors $\hat{i_B}$ and $\hat{i_3}$are coming out of the page.

From Figure 3 it can be made out that:

$$\vec{p_i^B} = \vec{p_i^3} \tag{6}$$

$$\vec{p_j^B} = \vec{p_j^3}\cos\phi + \vec{p_k^3}\sin\phi \tag{7}$$

$$\vec{p_k^B} = \vec{p_k^3}\cos\phi - \vec{p_j^3}\sin\phi \tag{8}$$

Therefore:

$$\begin{bmatrix} \vec{p_i^B} \\ \vec{p_j^B} \\ \vec{p_k^B} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \vec{p_i^3} \\ \vec{p_j^3} \\ \vec{p_k^3} \end{bmatrix} \tag{9}$$

We now have our rotation matrix that will map vectors from $\mathcal{F}_B$ to $\mathcal{F}_3$:

$$\begin{bmatrix} \hat{i_B} \\ \hat{j_B} \\ \hat{k_B} \end{bmatrix} = R^T(\phi) \begin{bmatrix} \hat{i_3} \\ \hat{j_3} \\ \hat{k_3} \end{bmatrix} \tag{10}$$

Where:

$$R^T(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{11}$$

It is worth noting that the matrix obtained has been defined as the transpose of a matrix in order to simplify later calculations.

5.3.2 Mapping from $\mathcal{F}_3$ to $\mathcal{F}_2$

The second rotation is about $\hat{j_3}$ by and angle θ. By the same method as in the previous part we can obtain a rotation matrix to map vectors from $\mathcal{F}_3$ to $\mathcal{F}_2$:

$$\begin{bmatrix} \hat{i_3} \\ \hat{j_3} \\ \hat{k_3} \end{bmatrix} = R^T(\theta) \begin{bmatrix} \hat{i_2} \\ \hat{j_2} \\ \hat{k_2} \end{bmatrix} \tag{12}$$

Where:

$$R^T(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{13}$$

The full working can be seen in Appendix A.

5.3.3 Mapping from $\mathcal{F}_2$ to $\mathcal{F}_1$

The final rotation is about $\hat{k_2}$ by and angle ψ. We can obtain a rotation matrix to map vectors from $\mathcal{F}_2$ to $\mathcal{F}_1$:

$$\begin{bmatrix} \hat{i_2} \\ \hat{j_2} \\ \hat{k_2} \end{bmatrix} = R^T(\psi) \begin{bmatrix} \hat{i_1} \\ \hat{j_1} \\ \hat{k_1} \end{bmatrix} \tag{14}$$

Where:

$$R^T(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{15}$$

The full working can be seen in Appendix A. F_1 is in alignment with F_I therefore if we combine the above rotation matrices we will have obtained a rotation matrix that can map from F_B to F_I. This rotation matrix will be called $R(\Theta)$.

5.3.4 Mapping from F_B to F_I

We can denote the linear velocity vector with respect to F_I of the quadcopter by:

$$\vec{v} = \begin{bmatrix} u^I & v^I & w^I \end{bmatrix} \begin{bmatrix} \hat{i_I} \\ \hat{j_I} \\ \hat{k_I} \end{bmatrix} \tag{16}$$

We can also write the velocity vector with respect to F_B:

$$\vec{v} = \begin{bmatrix} u^B & v^B & w^B \end{bmatrix} \begin{bmatrix} \hat{i_B} \\ \hat{j_B} \\ \hat{k_B} \end{bmatrix} \tag{17}$$

Now using the rotation matrices:

$$\vec{v} = \begin{bmatrix} u^B & v^B & w^B \end{bmatrix} R^T(\phi) \begin{bmatrix} \hat{i_3} \\ \hat{j_3} \\ \hat{k_3} \end{bmatrix} \tag{18}$$

$$\vec{v} = \begin{bmatrix} u^B & v^B & w^B \end{bmatrix} R^T(\phi)R^T(\theta) \begin{bmatrix} \hat{i_2} \\ \hat{j_2} \\ \hat{k_2} \end{bmatrix} \tag{19}$$

$$\vec{v} = \begin{bmatrix} u^B & v^B & w^B \end{bmatrix} R^T(\phi)R^T(\theta)R^T(\psi) \begin{bmatrix} \hat{i_1} \\ \hat{j_1} \\ \hat{1_2} \end{bmatrix} \tag{20}$$

Now equate the right hand side of equations (20) and (16):

$$\begin{bmatrix} u^B & v^B & w^B \end{bmatrix} R^T(\phi)R^T(\theta)R^T(\psi) \begin{bmatrix} \hat{i_1} \\ \hat{j_1} \\ \hat{1_2} \end{bmatrix} = \begin{bmatrix} u^I & v^I & w^I \end{bmatrix} \begin{bmatrix} \hat{i_I} \\ \hat{j_I} \\ \hat{k_I} \end{bmatrix} \tag{21}$$

Therefore:

$$\begin{bmatrix} u^B & v^B & w^B \end{bmatrix} R^T(\phi)R^T(\theta)R^T(\psi) = \begin{bmatrix} u^I & v^I & w^I \end{bmatrix} \tag{22}$$

Now taking the transpose of each side (Recalling that $(AB)^T = B^T A^T$):

$$\begin{bmatrix} u^B \\ v^B \\ w^B \end{bmatrix} = R(\psi)R(\theta)R(\phi) \begin{bmatrix} u^B \\ v^B \\ w^B \end{bmatrix} \tag{23}$$

We will therefore define our overall rotation matrix $R(\Theta)$ as being equal to$R(\psi)R(\theta)R(\phi)$, therefore:

$$R(\Theta) = \left(R^T(\phi)R^T(\theta)R^T(\psi)\right)^T \tag{24}$$

$$R(\Theta) = \left(\begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}^T \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}^T \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}^T \right)^T \tag{25}$$

Giving:

$$R(\Theta) = \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\phi\sin\theta - \cos\phi\sin\psi & \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{26}$$

For a given vector $\vec{v}$ in $\digamma_B$, the corresponding vector in $\digamma_I$ is given by $R(\Theta)\vec{v}$

It is now simple to create a function in MatLab that will map vectors from $\digamma_B$ to $\digamma_I$. This code can be viewed in Appendix B, Section 1.

5.4 Orientation Dynamics

We now need the corresponding transfer matrix in order to convert from the angular velocity of the quadcopter to the time derivatives of the Euler angles. A less detailed version of this derivation can be found in [3, pp.27-28] and [6, p.100]:

During an infinitesimal time interval dt the quadcopter undergoes three infinitesimal rotations $d\psi$, $d\theta$ and $d\phi$ therefore the quadcopters orientation can now be given by: $\psi + d\psi, \theta + d\theta, \phi + d\phi$. According to [3], "finite rotations cannot be treated as vectors, infinitesimal rotations may be treated as such", and therefore according to [6]:

$$\hat{n} = d\phi\hat{i}_B + d\theta\hat{j}_2 + d\psi\hat{k}_1 \tag{27}$$

Therefore:

$$\overrightarrow{\omega} = \frac{d\hat{n}}{dt} = \dot{\phi}\hat{i}_B + \dot{\theta}\hat{j}_2 + \dot{\psi}\hat{k}_1 \tag{28}$$

We can also define the angular velocity vector with respect to F_B:

$$\omega^B = p\hat{i}_B + q\hat{j}_B + r\hat{k}_B \tag{29}$$

Therefore:

$$\dot{\phi}\hat{i}_B + \dot{\theta}\hat{j}_2 + \dot{\psi}\hat{k}_1 = p\hat{i}_B + q\hat{j}_B + r\hat{k}_B \tag{30}$$

We can now use the separate rotation matrices to relate $\begin{bmatrix} p \\ q \\ r \end{bmatrix}$ and $\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R^T(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R^T(\phi)R^T(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{31}$$

This give us:

$$\omega^B = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{32}$$

The interstitial steps may be seen in Appendix C.

This has given us a way of converting from the time derivatives of the Euler angles to angular velocity but we would like to convert the other way around:

$$\dot{\Theta} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix}^{-1} \omega^B \tag{33}$$

Therefore:

$$\dot{\Theta} = \Psi(\Theta)\omega^B \tag{34}$$

Where:

$$\Psi(\Theta) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & \sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \tag{35}$$

This can now be made into a function in MatLab to convert from angular velocities to the time derivatives of the Euler angles which can be seen in Appendix B, Section 2.

We can now describe the position, velocity and angular velocity of the quadcopter with respect to F_I.

6 Forces and Moments Acting on the Quadcopter

Now that we can describe quadcopter motion, the next step towards a complete model of the quadcopter dynamics is to assess which forces and moments are acting on the quadcopter. An analysis of the forces and moments upon quadcopters is undertaken in [5, pp. 15-24] from which the list of forces and moments was obtained but the derivations of the equations for these forces and moments are original unless otherwise stated. Some forces, such as the 'hub forces' (defined in [5] as "the resultant of the horizontal forces acting on all the blade elements") and other aerodynamic effects such as the tendency of the advancing blade in a propeller to generate more lift than the retreating blade, have been neglected for the sake of simplifying the model. They may be included in the model at a later date.

6.1 Forces in the $\hat{k_I}$ direction

6.1.1 Weight of the Quadcopter

The force due to gravity is mg. As a vector with respect to F_I this is given by:

$$\begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{36}$$

6.1.2 Actuator Action

The thrust vector with respect to F_B can be expressed as $\vec{T_B}$ where $\vec{T_B} = \begin{bmatrix} 0 \\ 0 \\ T_k^B \end{bmatrix}$ as in F_B the thrust vector only has a 'vertical' component.
The thrust vector with respect to F_I is therefore given by:

$$\vec{T_I} = \begin{bmatrix} T_i^I \\ T_j^I \\ T_k^I \end{bmatrix} = R(\Theta) \begin{bmatrix} 0 \\ 0 \\ T_k^B \end{bmatrix} \tag{37}$$

$$\vec{T_I} = \begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\phi\sin\theta - \cos\phi\sin\psi & \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ T_k^B \end{bmatrix} \tag{38}$$

Therefore the component of thrust from the propellers in the $\hat{k_B}$ direction is given by:

$$T_k^I = \cos\phi\cos\theta T_k^B = \cos\phi\cos\theta\vec{T_B} = \cos\phi\cos\theta\sum_{i=1}^{4} T_i \tag{39}$$

Where i denotes the number of the propeller and T_i denotes the thrust generated by that propeller.

6.1.3 Friction

The frictional force on the quadcopter is modelled as being simply:

$$\vec{F}_D = \begin{bmatrix} -C_i\dot{x} \\ -C_j\dot{y} \\ -C_k\dot{z} \end{bmatrix} \tag{40}$$

Where C_i, C_j and C_k are constants that relate force to linear velocity. The model of friction used is a very primitive one but as it is very similar to the ones used in other successful projects such as [5] and [7] it appears to be sufficient. Neither of these projects offers an explanation of why this model may be appropriate so an outline of one has been attempted here. One method of expressing drag force is by use of the drag equation:

$$F_D = \frac{1}{2}\rho v^2 C_D A \tag{41}$$

Where F_D is the drag,ρ is the air density, v is the velocity, C_D is a dimensionless drag coefficient and A is the area of object experiencing drag. It is important to note that C_D cannot always be taken as a constant and at low Reynold's numbers there is a relationship between Reynold's number and C_D such that drag can be taken as approximately proportional to velocity. At higher Reynolds numbers this relationship changes and the drag is taken to be proportional to the square of the velocity. It is therefore assumed that the quadcopter will be operating at low Reynold's numbers and therefore frictional forces can be related to linear velocities by the constants C_x, C_y and C_z which incorporate drag coefficient, the density of air, the factor of one half and the area of the quadcopter generating drag.

The component of drag in the $\hat{k_B}$ direction is therefore:

$$-C_z\dot{z} \tag{42}$$

6.2 Forces in the $\hat{j_I}$ direction

6.2.1 Actuator Action

As with the component of thrust in the $\hat{k_B}$ direction, from equation (38) we can see that the component of thrust from the propellers in the $\hat{j_B}$ direction is given by:

$$T_j^I = (\cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi)\sum_{i=1}^{4} T_i \tag{43}$$

6.2.2 Hub Force

As stated in [5, p.20]: "the hub force is the resultant of the horizontal forces acting on all the blade elements". The horizontal forces are those acting in the plane defined by the orthonormal vectors $\hat{i_B}$ and $\hat{j_B}$ in F_B. As of yet this force has not been modelled for this project.

6.2.3 Friction

From equation (40) the frictional force in the $\hat{j_B}$ direction is given by:

$$-C_y\dot{y} \tag{44}$$

6.3 Forces in the $\hat{i_I}$ direction

6.3.1 Actuator Action

From equation (38) we can see that the component of thrust from the propellers in the $\hat{i_B}$ direction is given by:

$$T_j^I = (\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta)\sum_{i=1}^{4} T_i \tag{45}$$

6.3.2 Hub Force

As of yet this force has not been modelled for this project.

6.3.3 Friction

From equation (40) the frictional force in the $\hat{i_B}$ direction is given by:

$$-C_x\dot{x} \tag{46}$$

6.4 Rolling moments

Here we must arbitrarily define which propellers contribute to rolling moments and which contribute to pitching moments.

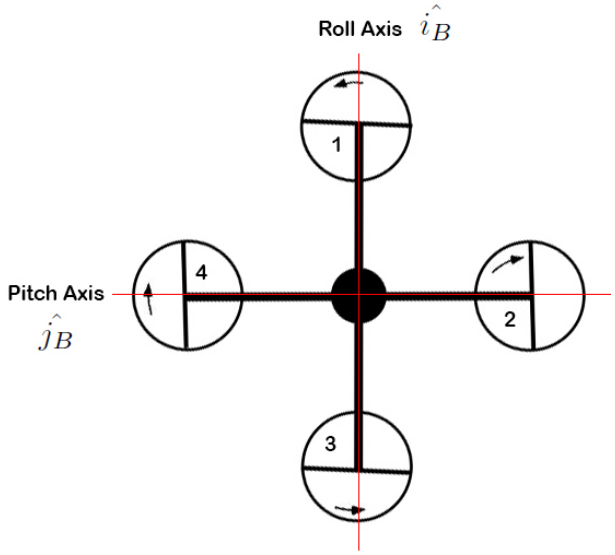


Figure 4: Assigning propellers to roll and pitch motion.

As can be seen in Figure 4, propellers 1 and 3 will be contributing to the pitching moment and propellers 2 and 4 will be contributing to the rolling moment.

6.4.1 Actuator Action

If we take the distance from the centre of the quadcopter to the centre of each propeller to be l then the rolling moment given by the propellers can be written as:

$$l(T_2 - T_4) \tag{47}$$

6.4.2 Gyroscopic Effect from the Quadcopter Body

The Newton-Euler formalism defines the translational and rotational dynamics of a rigid body:

$$\begin{bmatrix} f \\ \tau \end{bmatrix} = \begin{bmatrix} mI_3 & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} + \begin{bmatrix} \omega \times m \\ \omega \times I\omega \end{bmatrix} \tag{48}$$

Where I_3 is a 3×3 identity matrix, m is the mass of the body and I is the moment of inertia of the body about the centre of gravity. From this we can obtain Euler's equations for rigid body dynamics in vector form:

$$\tau = I\dot{\omega} + \omega \times (I\omega) \tag{49}$$

Giving us:

$$\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} I_{xx}\dot{p} + (I_{zz} - I_{yy})\, qr \\ I_{yy}\dot{q} + (I_{xx} - I_{zz})\, pr \\ I_{zz}\dot{r} + (I_{yy} - I_{xx})\, pq \end{bmatrix} \tag{50}$$

$$\begin{bmatrix} I_{xx}\dot{p} \\ I_{xx}\dot{q} \\ I_{xx}\dot{r} \end{bmatrix} = \begin{bmatrix} \tau_\phi + (I_{yy} - I_{zz})\, qr \\ \tau_\theta + (I_{zz} - I_{xx})\, pr \\ \tau_\psi + (I_{xx} - I_{yy})\, pq \end{bmatrix} \tag{51}$$

Where $\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix}$ are the external torques. We can therefore see that the rolling gyroscopic effect from the quadcopter body is given by:

$$(I_{yy} - I_{zz})\, qr \tag{52}$$

Even when hub force has been modelled, this moment may be neglected for the sake of simplicity as the hub force and the distance h will both be small.

6.5 Pitching moments

6.5.1 Actuator Action

If we take the distance from the centre of the hub to the centre of each propeller to be l then the rolling moment given by the propellers can be written as:

$$l(T_1 - T_3) \tag{53}$$

6.5.2 Gyroscopic Effect from the Quadcopter Body

From equation (51) the pitching gyroscopic effect from the quadcopter body is given by:

$$(I_{zz} - I_{xx})\, pr \tag{54}$$

6.6 Yawing moments

6.6.1 Gyroscopic Effect from the Quadcopter Body

From equation (51) the pitching gyroscopic effect from the quadcopter body is given by:

$$(I_{xx} - I_{yy})\, pq \tag{55}$$

6.6.2 Inertial Counter Torque from the Propellers

Any change in rotor speed will result in a torque given by:

$$I_R \dot{\Omega}_r$$

Where I_R is the rotor inertia and $\dot{\Omega}_r$ is the overall residual propeller angular acceleration (therefore if all of the propellers are accelerating uniformly then $\dot{\Omega}_r$ will equal zero. $\dot{\Omega}_r$ is only nonzero if there is a disparity if there is a net angular acceleration when the angular acceleration of all of the propellers are summed).

6.6.3 Yaw Due to Drag

If D_i is the torque due to drag on the i^{th}propeller then the total yaw due to drag is given by:

$$\sum_{i=1}^{4} (-1)^{i+1} D_i \tag{56}$$

7 Equations of Motion

The above forces and torques can be written as:

$$f_I = m\vec{\ddot{\xi}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} (\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta)\sum_{i=1}^{4} T_i - C_x\dot{x} \\ (\cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi)\sum_{i=1}^{4} T_i - C_y\dot{y} \\ -mg + \cos\phi\cos\theta\sum_{i=1}^{4} T_i - C_z\dot{z} \end{bmatrix} \tag{57}$$

$$\tau_B = I\dot{\omega} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} l(T_2 - T_4) + (I_{yy} - I_{zz})\, qr \\ l(T_1 - T_3) + (I_{zz} - I_{xx})\, pr \\ \sum_{i=1}^{4} (-1)^{i+1} D_i + (I_{xx} - I_{yy})\, pq \end{bmatrix} \tag{58}$$

Giving us:

$$\ddot{\vec{\xi}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\left(\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta\right)\sum_{i=1}^{4} T_i - \frac{1}{m}C_x\dot{x} \\ \frac{1}{m}\left(\cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi\right)\sum_{i=1}^{4} T_i - \frac{1}{m}C_y\dot{y} \\ -mg + \cos\phi\cos\theta\sum_{i=1}^{4} T_i - C_z\dot{z} \end{bmatrix} \tag{59}$$

Therefore if we know the orientation, linear velocity, thrust of each propeller and hub forces then we know how the quadcopter will accelerate in three dimensional space.

$$\dot{\omega} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{I_{xx}} l(T_2 - T_4) + \frac{1}{I_{xx}}\left(I_{yy} - I_{zz}\right) qr \\ \frac{1}{I_{yy}} l(T_1 - T_3) + \frac{1}{I_{yy}}\left(I_{zz} - I_{xx}\right) pr \\ \frac{1}{I_{zz}}\sum_{i=1}^{4}(-1)^{i+1} D_i + \frac{1}{I_{zz}}\left(I_{xx} - I_{yy}\right) pq + I_R\dot{\Omega}_r \end{bmatrix}$$

Therefore if we know the thrusts of each propeller and the drag force created by the propeller blades and the angular velocity of the quadcopter then we can calculate the angular acceleration.

As shown in [7] we can now crudely relate the thrust and the drag fores on the propellers to the angular velocities of the rotors using Momentum Theory. For a richer model Blade Element Theory should be used but for reasons of simplicity it has not been included at this time.

7.1 Relating Thrust to Propeller Angular Velocity

The motor torque can be given by:

$$\tau_m = K_t(I - I_0) \tag{60}$$

Where τ_m is the rotor torque, K_t is the torque proportionality constant, I is the input current and I_0 is the current when there is no load on the motor.

The voltage across the motor can be expressed as the sum of back EMF and resistive losses:

$$V = IR_m + K_v\Omega \tag{61}$$

Where V is the voltage, I is the input current, R_m is the motor's resistance, K_v is the proportionality constant that indicates back EMF per RPM and Ω is the propeller blade angular velocity.

Power consumed is given by:

$$P = IV \tag{62}$$

From equation (60):

$$I = \frac{\tau_m}{K_t} + I_0 \tag{63}$$

Therefore:

$$P = \left(\frac{\tau_m}{K_t} + I_0\right)\left[\left(\frac{\tau_m}{K_t} + I_0\right)R_m + K_v K_t \Omega\right] \tag{64}$$

$$P = \frac{(\tau_m + I_0 K_t)(\tau_m R_m + I_0 K_t R_m + K_v \Omega)}{K_t^2} \tag{65}$$

Now assume that the motor's resistance and the current when there is no load on the motor are both negligible. These assumptions are made in [7] and will yield:

$$P \simeq \frac{K_v}{K_t}\tau_m \Omega \tag{66}$$

The motor's torque is proportional to the thrust produced therefore we can rewrite equation (66) as:

$$P \simeq \frac{K_v K_\tau}{K_t} T\Omega \tag{67}$$

Where K_τ is a proportionality constant that relates thrust to torque.

As can be seen in Appendix D, from Momentum Theory we get that the induced velocity of a propeller is given by:

$$\nu = \sqrt{\frac{T}{2\rho A}} \tag{68}$$

Where ν is the induced velocity, ρ is air density and A is the area swept through by the propeller. This relationship is derived in Appendix C.

$$P = T\nu \tag{69}$$

Therefore:

$$\frac{K_v K_\tau}{K_t} T\Omega = \frac{T^{\frac{3}{2}}}{\sqrt{2\rho A}} \tag{70}$$

$$T = \left[2\rho A\left(\frac{K_v K_\tau}{K_t}\right)\right]\Omega^2 \tag{71}$$

Therefore thrust is proportional to the square of the propellers angular velocity:

$$T = K\Omega^2 \tag{72}$$

We can therefore calculate some value for K, or calculate one based upon some existing quadcopter and then relate thrust to the square of the propellers angular velocity in our model of quadcopter dynamics.

This relationship rests upon Momentum Theory which is known to be an inaccurate method of analysing propellers. For the sake of simplicity Momentum Theory has been used here but a more complete analysis should incorporate Blade Element Analysis to better calculate the thrust generated by the propellers.

7.2 Relating Drag Force on the Propeller to Propeller Angular Velocity

Using the drag equation we can express the frictional force on the propeller blades as:

$$F_D = \frac{1}{2}\rho A v^2 C_D \tag{73}$$

Where $F_D$is the frictional force on the blades, ρ is the air density, v is the blade tip velocity and $C_D$is the dimensionless drag coefficient.

Here we can assume that all of this force acts at the blade's tip which is an erroneous assumption but irrelevant since we only desire to derive some relationship between the drag moment and propeller angular velocity.

$$\tau_D = \frac{1}{2}\rho A v^2 C_D R \tag{74}$$

Where R is the propeller radius. Given that:

$$v = \Omega R \tag{75}$$

We can rewrite equation (74) as:

$$\tau_D = \frac{1}{2}\rho A C_D R^3 \Omega^2 \tag{76}$$

Therefore the torque due to drag is proportional to the square of the angular velocity:

$$\tau_D = B\Omega^2 \tag{77}$$

7.3 Updated Equations of Motion

We can now rewrite the equations of motion:

$$f_I = m\vec{\ddot{\xi}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} (\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta)\, K\sum_{i=1}^{4}\Omega_i^2 - C_x\dot{x} \\ (\cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi)\, K\sum_{i=1}^{4}\Omega_i^2 - C_y\dot{y} \\ -mg + \cos\phi\cos\theta K\sum_{i=1}^{4}\Omega_i^2 - C_z\dot{z} \end{bmatrix} \tag{78}$$

$$\tau_B = I\dot{\omega} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} lK(\Omega_2^2 - \Omega_4^2) + (I_{yy} - I_{zz})\, qr \\ lK(\Omega_1^2 - \Omega_3^2) + (I_{zz} - I_{xx})\, pr \\ B\sum_{i=1}^{4}(-1)^{i+1}\Omega_i^2 + (I_{xx} - I_{yy})\, pq \end{bmatrix} \tag{79}$$

Therefore:

$$\ddot{\vec{\xi}} = \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\left(\sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta\right) K\sum_{i=1}^{4}\Omega_i^2 - \frac{1}{m}C_x\dot{x} \\ \frac{1}{m}\left(\cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi\right) K\sum_{i=1}^{4}\Omega_i^2 - \frac{1}{m}C_y\dot{y} \\ -g + \cos\phi\cos\theta K\sum_{i=1}^{4}\Omega_i^2 - \frac{1}{m}C_z\dot{z} \end{bmatrix} \tag{80}$$

$$\dot{\omega} = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{I_{xx}}lK(\Omega_2^2 - \Omega_4^2) + \frac{1}{I_{xx}}\left(I_{yy} - I_{zz}\right)qr \\ \frac{1}{I_{yy}}lK(\Omega_1^2 - \Omega_3^2) + \frac{1}{I_{yy}}\left(I_{zz} - I_{xx}\right)pr \\ \frac{1}{I_{zz}}B\sum_{i=1}^{4}(-1)^{i+1}\Omega_i^2 + \frac{1}{I_{zz}}\left(I_{xx} - I_{yy}\right)pq \end{bmatrix} \tag{81}$$

MatLab functions can now be written that compute the quadcopter's translational and rotational acceleration. These functions can be found in Appendix B.

8 Rudimentary PD Controller

Proportional-Integral-Derivative (PID) control is perhaps the most widely used control mechanism and has been used since its conception almost one hundred years ago when it was developed by Nicolas Minorsky as a means to automatically steer ships for the US Navy as documented in [9, pp.145-146]. Although this technique is very old it is still used today and given the simplicity of designing PID controllers, it will serve well to test the model of quadcopter dynamics derived in Sections 5-7.

As outlined in [10, pp.211-217], PID control or 'three-term' control calculates the error between a desired set point and the measured state as well as the rate of change of this error and the integral of this error. The output of the controller is a signal that is proportional to the sum of these errors, each multiplied by some constant. The derivative, or 'D', term serves to reduce overshoot of the system and the integral, 'I', term serves to eliminate steady-state errors.

The derivative or integral terms may be left out to create P,PI or PD controllers.

To control the quadcopter using PID or PD control, six single-input, single-output (SISO) controllers are necessary to govern roll, pitch, yaw (heading), and motion in the $\hat{i_I}, \hat{j_I}, \hat{k_I}$ directions.

The first step is to stabilise the roll pitch and yaw. The errors can be defined thusly:

$$e_\phi = \phi_{desired} - \phi_{measured} \tag{82}$$

$$e_\theta = \theta_{desired} - \theta_{measured} \tag{83}$$

$$e_\psi = \psi_{desired} - \psi_{measured} \tag{84}$$

The outputs of the PD controllers will therefore be:

$$K_p^\phi e_\phi + K_d^\phi \frac{de_\phi(t)}{dt} \tag{85}$$

$$K_p^\theta e_\theta + K_d^\theta \frac{de_\theta(t)}{dt} \tag{86}$$

$$K_p^{\psi} e_{\psi} + K_d^{\psi} \frac{de_{\psi}(t)}{dt} \tag{87}$$

Where K_p denotes the proportional gain tuning parameter and K_d denotes the derivative gain tuning parameter and the superscripts ϕ, θ and ψ denoted which action the tuning parameters apply to. Given that the roll and pitch movements are identical but about different axes, we can use the same values for K_p^{ϕ} and K_p^{θ} and for K_d^{ϕ} and K_d^{θ}. As a starting point the same values will also be used for K_p^{ψ} and K_d^{ψ} although these may have to be altered. We may also assume for the sake of this test controller that $\phi_{desired}$ and $\theta_{desired}$ are zero so that controller always stabilises but we would like to be able to set the heading $\psi_{desired}$.

The torques are related to the orientation of the quadcopter by the relationship:

$$\tau = I\ddot{\Theta} \tag{88}$$

Where I is the inertia matrix for the quadcopter. We therefore want the output of our PD controller to be related to the torques by the relationship as stated in [7]:

$$\tau = Iu(t) \tag{89}$$

$$\tau = \begin{bmatrix} \tau_{\phi} \\ \tau_{\theta} \\ \tau_{\psi} \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} K_p^{\phi} e_{\phi} + K_d^{\phi} \frac{de_{\phi}(t)}{dt} \\ K_p^{\theta} e_{\theta} + K_d^{\theta} \frac{de_{\theta}(t)}{dt} \\ K_p^{\psi} e_{\psi} + K_d^{\psi} \frac{de_{\psi}(t)}{dt} \end{bmatrix} \tag{90}$$

Given that we are assuming $\phi_{desired}$ and $\theta_{desired}$ are zero, this means that:

$$e_{\phi} = -\phi_{measured} = -\phi \tag{91}$$

$$e_{\theta} = -\theta_{measured} = -\theta \tag{92}$$

$$e_{\psi} = \psi_{desired} - \psi_{measured} \tag{93}$$

Therefore:

$$\begin{bmatrix} \tau_{\phi} \\ \tau_{\theta} \\ \tau_{\psi} \end{bmatrix} = \begin{bmatrix} -I_{xx} \left(K_p^{\phi} \phi + K_d^{\phi} \dot{\phi} \right) \\ -I_{yy} \left(K_p^{\theta} \theta + K_d^{\theta} \dot{\theta} \right) \\ -I_{zz} \left(K_p^{\psi} e_{\psi} + K_d^{\psi} \dot{e_{\psi}} \right) \end{bmatrix} \tag{94}$$

We can now equate the right hand sides of equations (79) and (95) for an expression that relates the desired set points of the controller (in this case they are zero) with the angular velocities of the propellers necessary to achieve these states.

$$\begin{bmatrix} lK(\Omega_2^2 - \Omega_4^2) + (I_{yy} - I_{zz})\, qr \\ lK(\Omega_1^2 - \Omega_3^2) + (I_{zz} - I_{xx})\, pr \\ B\sum_{i=1}^{4}(-1)^{i+1}\Omega_i^2 + (I_{xx} - I_{yy})\, pq \end{bmatrix} = \begin{bmatrix} -I_{xx}\left(K_p^{\phi}\phi + K_d^{\phi}\dot{\phi}\right) \\ -I_{yy}\left(K_p^{\theta}\theta + K_d^{\theta}\dot{\theta}\right) \\ -I_{zz}\left(K_p^{\psi}e_{\psi} + K_d^{\psi}\dot{e_{\psi}}\right) \end{bmatrix} \tag{95}$$

This gives us three equations with four unknowns therefore we need an additional constraint. For the quadcopter to stay aloft we know that the component of thrust from the actuators in the $\hat{k}_i$ direction must be equal to the weight of the quadcopter given by mg. The thrust vector in $\mathcal{F}_B$ is given by:

$$T_B = \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \tag{96}$$

Where $T = K\sum_{i=1}^{4}\Omega_i^2$. As the the blades do not have variable pitch and we are neglecting the effects of blade flapping, the thrust vector in $\mathcal{F}_B$ only has a $\hat{k}_B$ component. If we define the thrust vector in $\mathcal{F}_I$ as $\mathcal{F}_I = \begin{bmatrix} T_i \\ T_j \\ T_k \end{bmatrix}$, we know that T_k is equal to mg and we are not interested in the other components. We must now use the rotation matrix defined in Section 5.3.4 to map the thrust vector into $\mathcal{F}_I$:

$$\begin{bmatrix} \cos\theta\cos\psi & \cos\psi\sin\phi\sin\theta - \cos\phi\sin\psi & \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi \\ -\sin\theta & \cos\theta\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = \begin{bmatrix} T_i \\ T_j \\ mg \end{bmatrix} \tag{97}$$

Therefore:

$$mg = \cos\phi\cos\theta T \tag{98}$$

Which gives:

$$T = \frac{mg}{\cos\phi\cos\theta} \tag{99}$$

And therefore:

$$K\sum_{i=1}^{4}\Omega_i^2 = \frac{mg}{\cos\phi\cos\theta} \tag{100}$$

So we now have four equations and four unknowns so we can solve the system of equations for Ω_1, Ω_2, Ω_3 and Ω_4. These equations are given in full below:

$$\begin{bmatrix} lK(\Omega_2^2 - \Omega_4^2) + (I_{yy} - I_{zz})\, qr \\ lK(\Omega_1^2 - \Omega_3^2) + (I_{zz} - I_{xx})\, pr \\ B\left(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2\right) + (I_{xx} - I_{yy})\, pq + I_R\dot{\Omega}_r \\ K\left(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right) \end{bmatrix} = \begin{bmatrix} -I_{xx}\left(K_p^{\phi}\phi + K_d^{\phi}\dot{\phi}\right) \\ -I_{yy}\left(K_p^{\theta}\theta + K_d^{\theta}\dot{\theta}\right) \\ -I_{zz}\left(K_p^{\psi}e_{\psi} + K_d^{\psi}\dot{e_{\psi}}\right) \\ \frac{mg}{\cos\phi\cos\theta} \end{bmatrix} \tag{101}$$

We can assume in steady flight, for the sake of this test controller at least, that the contributions from the gyroscopic effects of the quadcopter body and the propellers are negligible and therefore rewrite the equations as:

$$\begin{bmatrix} lK(\Omega_2^2 - \Omega_4^2) \\ lK(\Omega_1^2 - \Omega_3^2) \\ B\left(\Omega_1^2 - \Omega_2^2 + \Omega_3^2 - \Omega_4^2\right) \\ K\left(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2\right) \end{bmatrix} = \begin{bmatrix} -I_{xx}\left(K_p^{\phi}\phi + K_d^{\phi}\dot{\phi}\right) \\ -I_{yy}\left(K_p^{\theta}\theta + K_d^{\theta}\dot{\theta}\right) \\ -I_{zz}\left(K_p^{\psi}e_{\psi} + K_d^{\psi}\dot{e_{\psi}}\right) \\ \frac{mg}{\cos\phi\cos\theta} \end{bmatrix} \tag{102}$$

These equations can then be solved to give the necessary angular velocities of each propeller:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{mg}{4K\cos\phi\cos\theta} + \frac{-2B\left(K_p^{\theta}\theta + K_d^{\theta}\dot{\theta}\right)I_{yy} + lK\left(K_p^{\psi}e_{\psi} + K_d^{\psi}\dot{e_{\psi}}\right)I_{zz}}{4lBK} \\ \frac{mg}{4K\cos\phi\cos\theta} + \frac{-2B\left(K_p^{\phi}\phi + K_d^{\phi}\dot{\phi}\right)I_{xx} - lK\left(K_p^{\psi}e_{\psi} + K_d^{\psi}\dot{e_{\psi}}\right)I_{zz}}{4lBK} \\ \frac{mg}{4K\cos\phi\cos\theta} + \frac{2B\left(K_p^{\theta}\theta + K_d^{\theta}\dot{\theta}\right)I_{yy} + lK\left(K_p^{\psi}e_{\psi} + K_d^{\psi}\dot{e_{\psi}}\right)I_{zz}}{4lBK} \\ \frac{mg}{4K\cos\phi\cos\theta} + \frac{2B\left(K_p^{\phi}\phi + K_d^{\phi}\dot{\phi}\right)I_{xx} - lK\left(K_p^{\psi}e_{\psi} + K_d^{\psi}\dot{e_{\psi}}\right)I_{zz}}{4lBK} \end{bmatrix} \tag{103}$$

The MatLab code for this controller can be seen in Appendix B, Section 5. We can the plot the angular velocity and angular displacement of the quadcopter with respect to time when the angular velocities of the propellers are governed by the PD controller as shown in Figure 5. Some hypothetical values for physical constants such as mass and friction constants were given based upon existing quadcopters and a range of values for K_d, K_p and K_i were used until an appropriate system response was obtained. An initial disturbance is set in the form of an angular velocity so that the quadcopter has to restore to a stable orientation. On the graph showing angular displacement, the red line shows the roll, the green line shows pitch and the dashed blue line shows the yaw. On the graph showing angular velocity the same colours indicate the time derivatives of these same Euler angles.

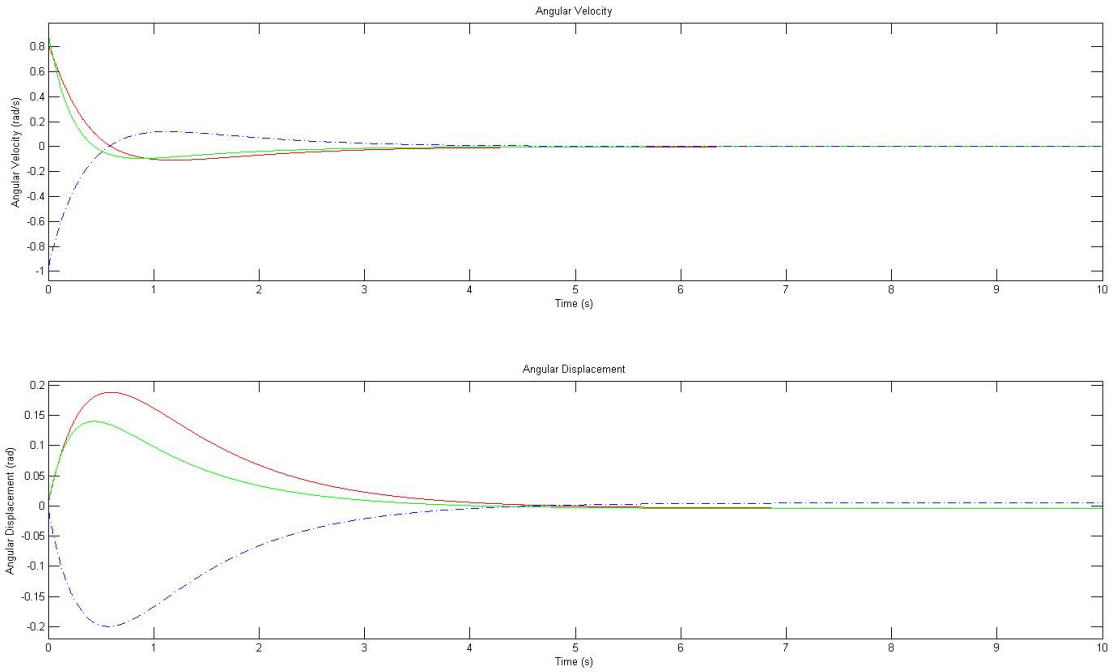


Figure 5: Stabilising the quadcopter using PD control

This result shows that the quadcopter can be stabilised using PD control. However, from the graph of angular displacement we can see that there is a small steady state error. In order to eliminate this error we should use a PID controller by augmenting our PD controller with an integral gain tuning parameter.

9 Rudimentary PID Controller

In order to create a PID controller it is simply a case of augmenting the output from the controller to include an integral gain tuning parameter. In this case the angular velocities of each propeller are given by:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{mg}{4K\cos\phi\cos\theta} + \frac{-2B\left(K_p^\theta\theta + K_d^\theta\dot{\theta} + K_I^\theta\int_0^T\theta dt\right)I_{yy} + lK\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi\int_0^T e_\psi dt\right)I_{zz}}{4lBK} \\ \frac{mg}{4K\cos\phi\cos\theta} + \frac{-2B\left(K_p^\phi\phi + K_d^\phi\dot{\phi} + K_I^\phi\int_0^T\phi dt\right)I_{xx} - lK\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi\int_0^T e_\psi dt\right)I_{zz}}{4lBK} \\ \frac{mg}{4K\cos\phi\cos\theta} + \frac{2B\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi\int_0^T e_\psi dt\right)I_{yy} + lK\left(K_p^\theta\theta + K_d^\theta\dot{\theta} + K_I^\theta\int_0^T\theta dt\right)I_{zz}}{4lBK} \\ \frac{mg}{4K\cos\phi\cos\theta} + \frac{2B\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi\int_0^T e_\psi dt\right)I_{xx} - lK\left(K_p^\phi\phi + K_d^\phi\dot{\phi} + K_I^\phi\int_0^T\phi dt\right)I_{zz}}{4lBK} \end{bmatrix} \quad (104)$$

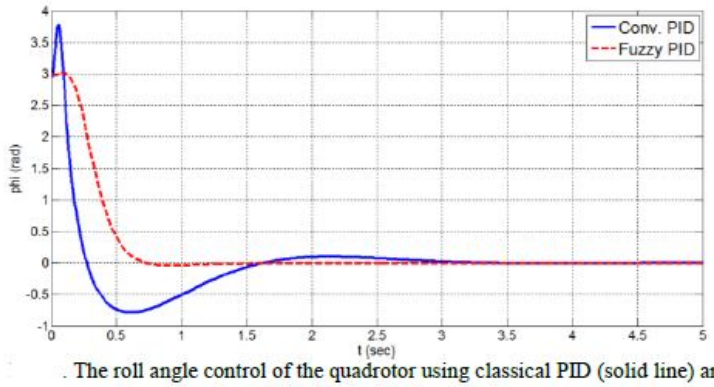
10 Further Literature Review

Although a large amount of literature review has gone into modelling quadcopter motion and designing the PID controller, further review into the possible control mechanisms that could be applied to a quadcopter, as well as various 3D visualization techniques, will be detailed in this section.

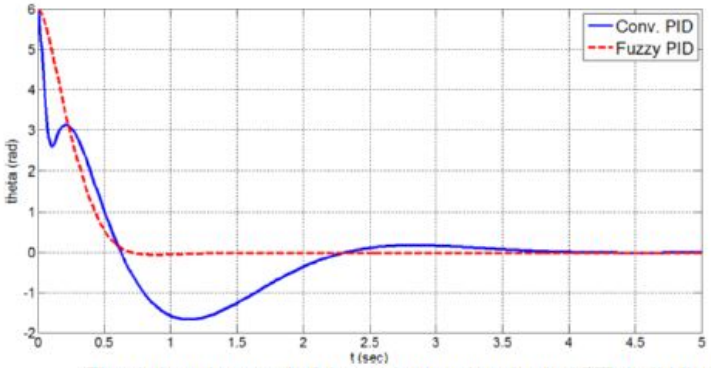
10.1 Other Possible Control Mechanisms

A number of control methods have been applied to the quadcopter problem including PID (Proportional Integral Derivative) control [7], LQR (Linear Quadratic Regulator) [8], H_∞ control [11] and fuzzy logic control [12]. Combinations of these control techniques can be used as seen in [13] where a 'fuzzy PID' controller was created with promising results. While it has been demonstrated that even a blunt instrument such as PD control can be used to stabilise the quadcopter, PID control has some limitations. The parameters K_d, K_p and K_i require tuning to find a stable configuration as PID control does not guarantee stability. Neither does PID control guarantee an optimal controller or a robust one. It lacks robustness in the sense that the above model of quadcopter motion (presented in Sections 5-7) does not take into account wind speed and although the controller will measure the error in position and orientation and react accordingly, PID controllers are linear and will not respond well to non-linear effects. It lacks optimality as there is no reason why a PID controller will offer the best way of controlling the quadcopter, the response time may be sluggish and the system process variables may overshoot their desired values.

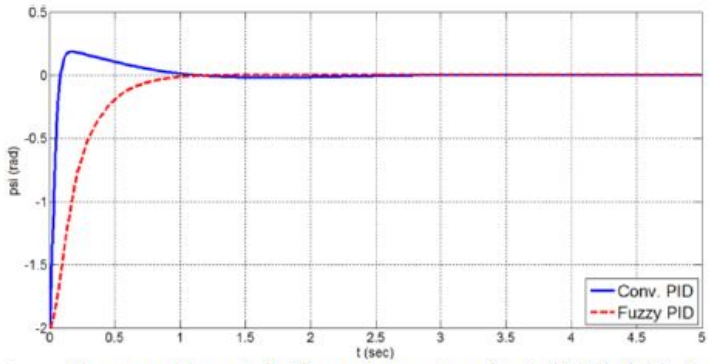
There are ways of improving the efficacy of a PID control such as tuning the gain parameters which can be done automatically as in [7] to good effect. An even more sophisticated method is to use a 'fuzzy PID' controller as in [13], the results of which can be seen in Figure 6.



. The roll angle control of the quadrotor using classical PID (solid line) and fuzzy PID (dashed line) controllers



The pitch angle control of the quadrotor using classical PID (solid line) and fuzzy PID (dashed line) controllers



The yaw angle control of the quadrotor using classical PID (solid line) and fuzzy PID (dashed line) controllers

Figure 6: 'Fuzzy' PID vs. PID from [13]

The 'Fuzzy' control uses fuzzy logic which is a form of logic in which the truth value may be any value between 1 and 0 (as opposed to binary logic in which truth values are either 1 or 0) in order to tune the PID parameters in various different operating conditions. Fuzzy control is non-linear and is therefore intrinsically better suited to dealing with the nonlinear dynamics of a quadcopter than a classical controller.

Another more modern approach to quadcopter control is the use of H_∞ control as practiced in [11]. H_∞ control is another nonlinear control technique that has been used to govern quadcopter motion. The objective of H_∞control is to optimise robust stabilisation and robust performance by taking into account the disturbances and the discrepancies between the actual plant and the model of the plant that we are

operating with.

10.2 3D Visualisation

A number of similar projects have used 3D visualisation in order to demonstrate the efficacy of their controllers. Figure 7 shows the simple visualiser designed by Gibianksy in [7] which does not allow the user to input desired values for the orientation or position of the quadcopter but assumes an initial disturbance and then simulates the quadcopter's restoration to the 'neutral' position (all Euler angles being equal to zero). The visualiser shows the thrust vectors of each propeller as arbitrarily scaled coloured cylinders to give an indication of the thrust generated by each propeller.

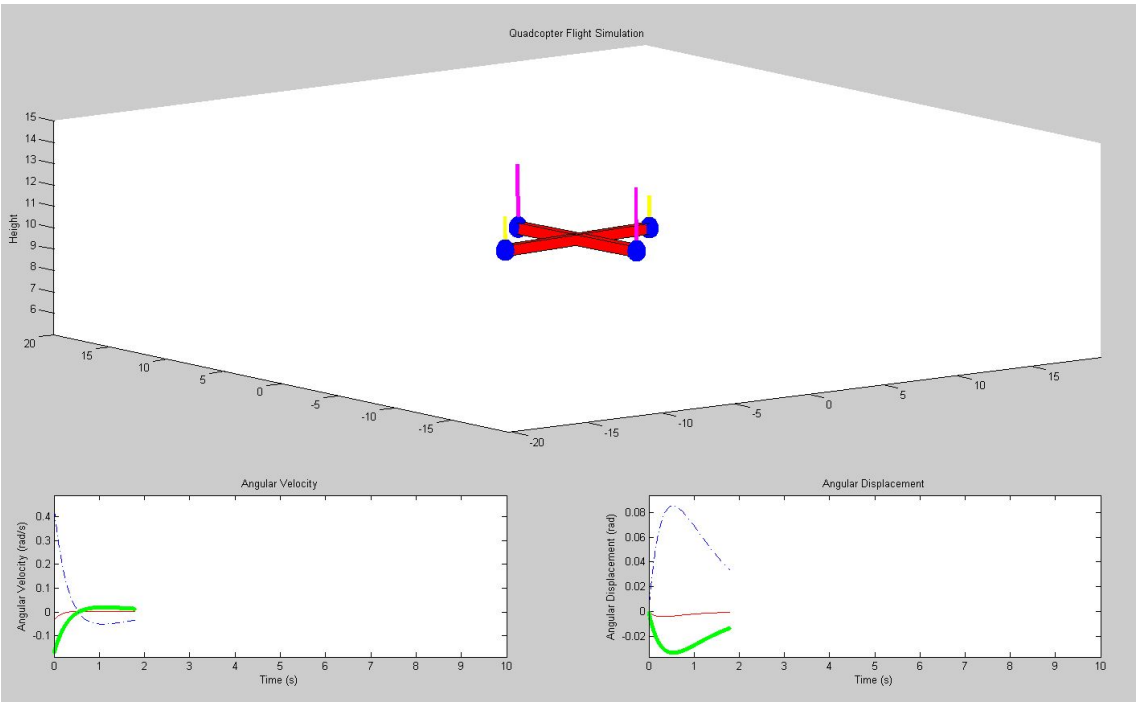


Figure 7: 3D visualisation by Gibianksy in [7]

A more detailed approach was taken by al-Omari et al. in [14]. In their visualiser the user could enter coordinates and a heading (yaw angle) for the quadcopter in real time as shown in Figure 8.

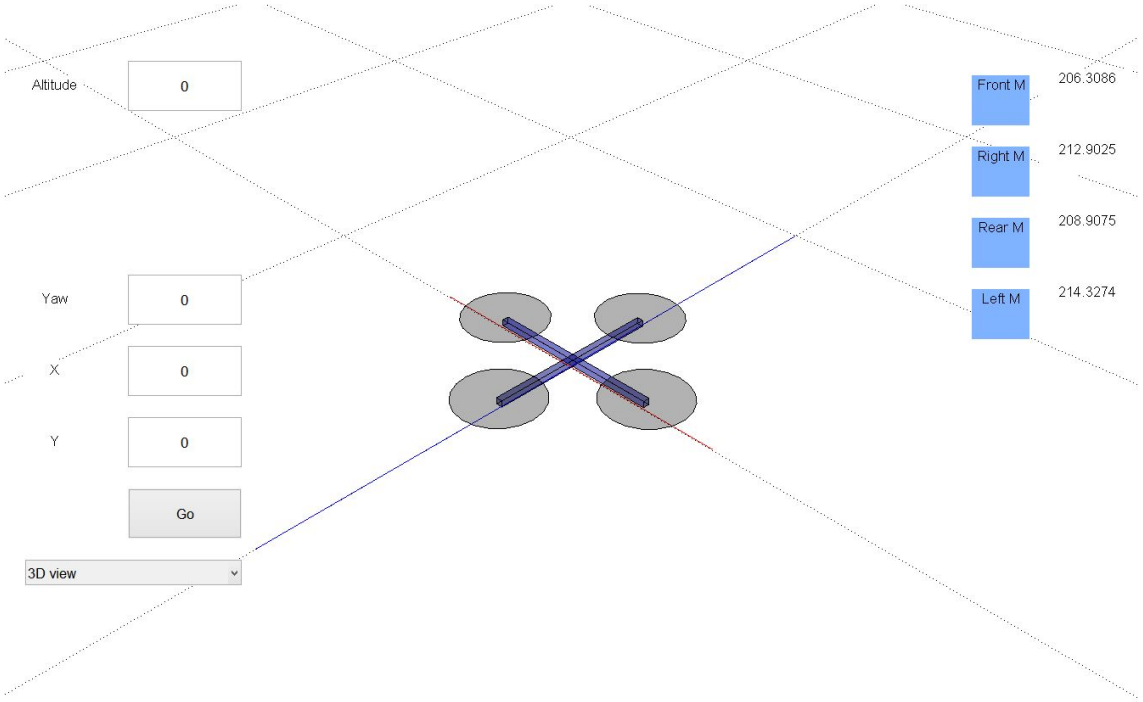


Figure 8: 3D visualisation by al-Omari et al. in [14]

No such visualisation has yet been created for this project but ideally one similar to that created by al-Omari et al. will be developed in order to demonstrate the efficacy of different controllers.

11 Developing the PID controllers

First the error terms for the spatial coordinates must be defined:

$$e_x = x_{desired} - x_{measured} \tag{105}$$

$$e_y = y_{desired} - y_{measured} \tag{106}$$

$$e_z = z_{desired} - z_{measured} \tag{107}$$

11.1 Controlling motion in the x and y directions

In the PID controllers shown above the functionality is extremely limited. These controllers are only capable of driving the orientation of the quadcopter back to a 'neutral' state. However, this is easily altered so that we can drive the orientation of the quadcopter to some desired orientation vector.

Once this is done we can set this desired roll and pitch angles as being proportional to the error (and the derivative and integral of the error) between the desired coordinates and the measured coordinates in the x and y directions. By this method, when the quadcopter is far away from its desired coordinate it will tilt towards said desired coordinate and thus tilt the thrust vector towards it. As it approaches

the destination the error is reduced and thus the quadcopter begins to pull back. This is shown by the following equations:

$$\phi_{desired} = K_p^y \times e_y + K_d^y \times \dot{e}_y + K_i^y \times \int_0^T e_y dt \tag{108}$$

$$\theta_{desired} = K_p^x \times e_x + K_d^x \times \dot{e}_x + K_i^x \times \int_0^T e_x dt \tag{109}$$

A limit must be imposed upon the desired angles otherwise the quadcopter would simply flip over if the desired coordinates were too far away. The limits can be expressed by:

$$-\frac{\pi}{4} \leq \phi \leq \frac{\pi}{4} \tag{110}$$

$$-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4} \tag{111}$$

The code to implement these equations can be viewed in Appendix 14.6.

11.2 Controlling motion in the z direction

Motion in the z direction is simpler to control as it can be done simply by setting the thrust as proportional to the errors in the z direction rather than the function used in the rudimentary controllers above which set thrust as the force necessary to maintain a constant altitude. In order to keep a constant altitude the thrust must be given by the following equation, derived in Section 8:

$$T = \frac{mg}{\cos\phi\cos\theta} \tag{112}$$

However, in order to have the altitude of the quadcopter as an input we can use a PID controller based upon the error between the desired and the measured altitude:

$$T = K_p^z \times e_z + K_d^z \times \dot{e}_z + K_i^z \times \int_0^T e_z dt \tag{113}$$

Some code must be included to prevent negative thrusts:

$$T \geq 0 \tag{114}$$

The code to implement this can be viewed in Appendix 14.7.

In the rudimentary PID controllers, adumbrated in Sections 8 and 9, the thrust is computed to always be exactly what is required to keep the quadcopter aloft. In this case the thrust is designed to approach that which is necessary to keep the quadcopter at the desired z-coordinate.

The angular velocities for the propellers are now given by:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{T}{4} + \frac{-2B\left(K_p^\theta \theta + K_d^\theta \dot{\theta} + K_I^\theta \int_0^T \theta dt\right) I_{yy} + lK\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi \int_0^T e_\psi dt\right) I_{zz}}{4lBK} \\ \frac{T}{4} + \frac{-2B\left(K_p^\phi \phi + K_d^\phi \dot{\phi} + K_I^\phi \int_0^T \phi dt\right) I_{xx} - lK\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi \int_0^T e_\psi dt\right) I_{zz}}{4lBK} \\ \frac{T}{4} + \frac{2B\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi \int_0^T e_\psi dt\right) I_{yy} + lK\left(K_p^\theta \theta + K_d^\theta \dot{\theta} + K_I^\theta \int_0^T \theta dt\right) I_{zz}}{4lBK} \\ \frac{T}{4} + \frac{2B\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi \int_0^T e_\psi dt\right) I_{xx} - lK\left(K_p^\phi \phi + K_d^\phi \dot{\phi} + K_I^\phi \int_0^T \phi dt\right) I_{zz}}{4lBK} \end{bmatrix} \tag{115}$$

Or, given in full:

$$\begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = \begin{bmatrix} \frac{K_p^z \times e_z + K_d^z \times \dot{e}_z + K_i^z \times \int_0^T e_z dt}{4} + \frac{-2B\left(K_p^\theta \theta + K_d^\theta \dot{\theta} + K_I^\theta \int_0^T \theta dt\right) I_{yy} + lK\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi \int_0^T e_\psi dt\right) I_{zz}}{4lBK} \\ \frac{K_p^z \times e_z + K_d^z \times \dot{e}_z + K_i^z \times \int_0^T e_z dt}{4} + \frac{-2B\left(K_p^\phi \phi + K_d^\phi \dot{\phi} + K_I^\phi \int_0^T \phi dt\right) I_{xx} - lK\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi \int_0^T e_\psi dt\right) I_{zz}}{4lBK} \\ \frac{K_p^z \times e_z + K_d^z \times \dot{e}_z + K_i^z \times \int_0^T e_z dt}{4} + \frac{2B\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi \int_0^T e_\psi dt\right) I_{yy} + lK\left(K_p^\theta \theta + K_d^\theta \dot{\theta} + K_I^\theta \int_0^T \theta dt\right) I_{zz}}{4lBK} \\ \frac{K_p^z \times e_z + K_d^z \times \dot{e}_z + K_i^z \times \int_0^T e_z dt}{4} + \frac{2B\left(K_p^\psi e_\psi + K_d^\psi \dot{e_\psi} + K_I^\psi \int_0^T e_\psi dt\right) I_{xx} - lK\left(K_p^\phi \phi + K_d^\phi \dot{\phi} + K_I^\phi \int_0^T \phi dt\right) I_{zz}}{4lBK} \end{bmatrix} \tag{116}$$

12 Simulating the PID controllers

12.1 Simulation

Once the tuning gain parameters have been calibrated, simulations can be run that will test the hypothetical quadcopter's response to given coordinates. When the simulation is run the code will generate five plots: a plot of the path through space that the quadcopter would take, a plot of quadcopter orientation against time, a plot of angular velocity against time, a plot of quadcopter position against time and a plot of the quadcopter's linear velocity against time. On the plot of the path taken by the quadcopter will also be printed the rise time of the quadcopter. This is defined by the time taken for the quadcopter to be within 5% of all desired coordinates.

In order to make the simulation environment more realistic, a randomised disturbance (in the form of an initial angular velocity of the quadcopter) is included in the simulation. However, this simulation is still very idealistic as it does not take into account wind and other external forcing conditions which make controlling quadcopters in reality a much more complex feat.

12.1.1 Example results

An example of the simulation, using the PD controller, is shown below in Figure 9. Default coordinates of $[100, 100, 100]$ are used in all simulations unless specified otherwise.

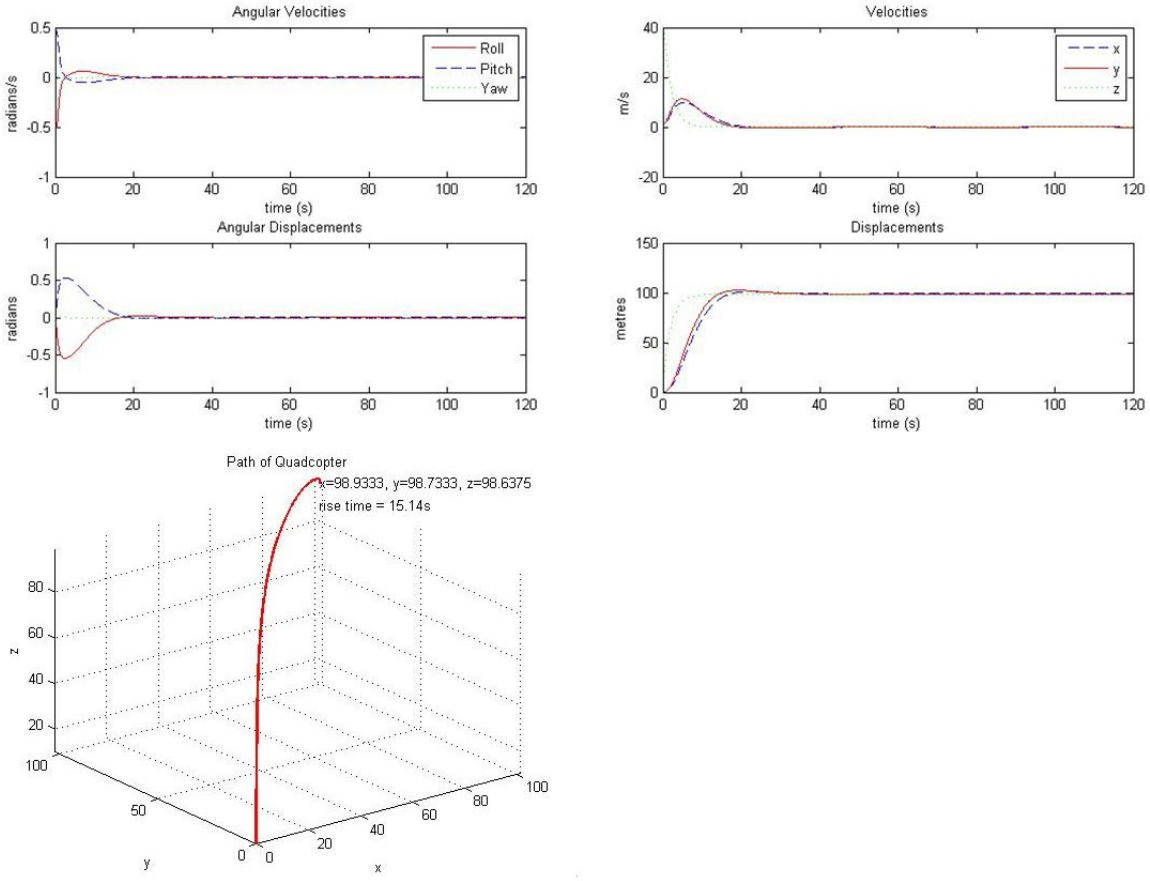


Figure 9: Simulating quadcopter motion when using a PD controller

The same simulation is run using a PID controller in Figure 10.

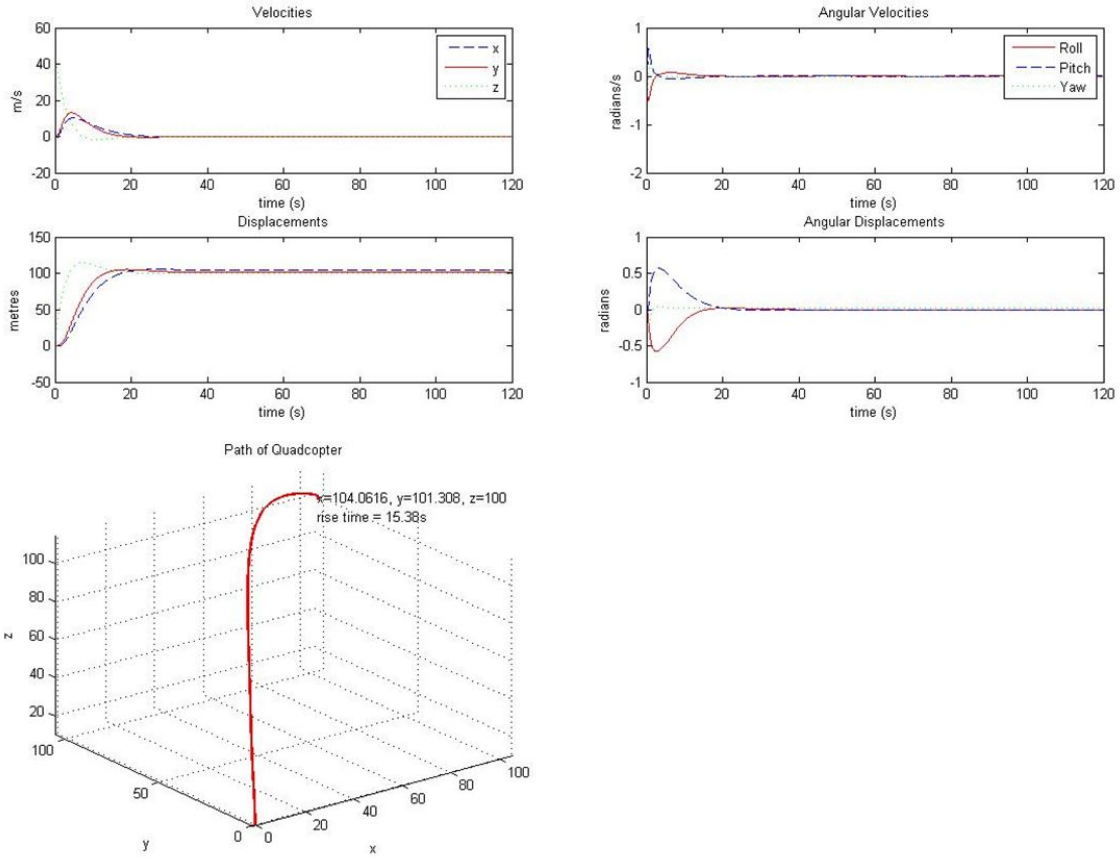


Figure 10: Simulating quadcopter motion when using a PID controller

12.1.2 3D, real-time visualisation

As well as the results shown in Figures 9 and 10 the performance of a controller can also be viewed in real-time. This is useful as the user can take information about the speed and acceleration of the quadcopter at various stages of its journey which is unavailable when simply looking at a complete plot of the quadcopter's path as in Figures 9 and 10. A still from one of these simulations can be seen in Figure 11. The code for this visualisation can be seen in Appendix 14.8.

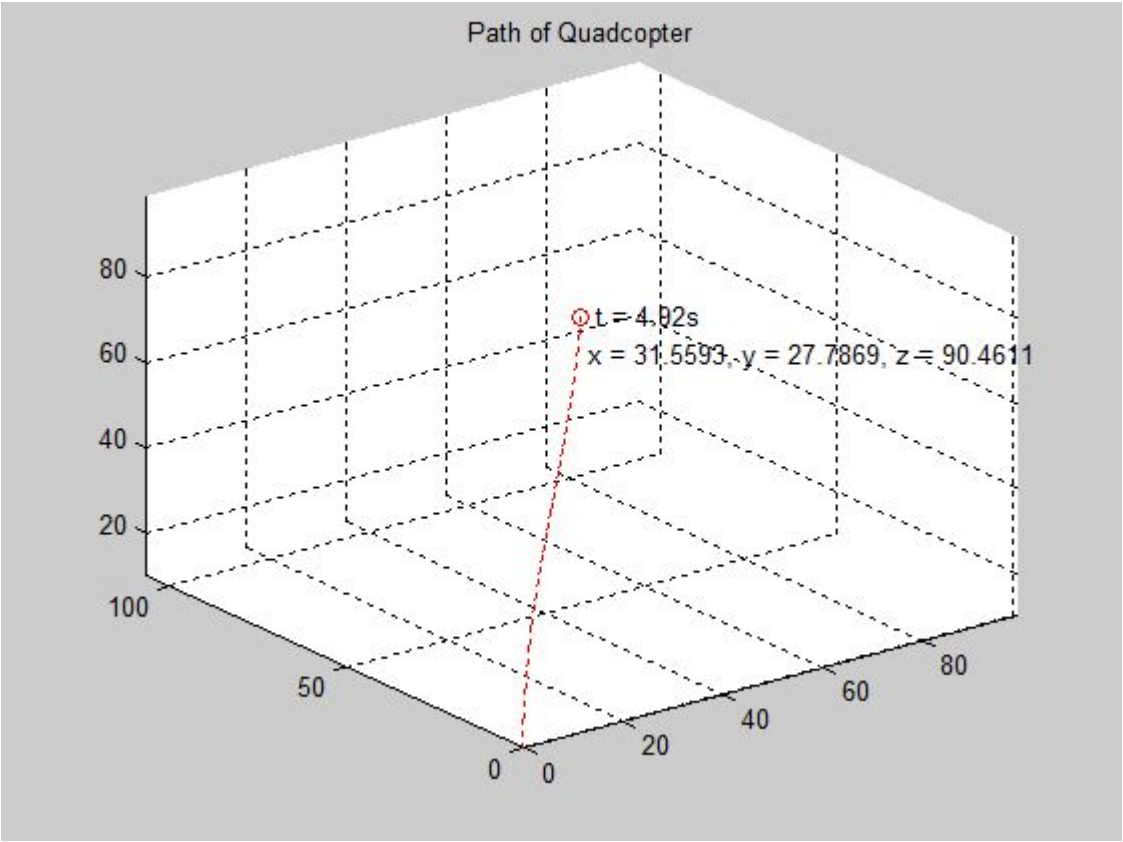


Figure 11: 3D, real-time visualisation of quadcopter motion

12.1.3 Gathering multiple results

It is difficult to compare the performance of controllers when there are random disturbance at work so it is useful to be able to see the mean results of multiple simulations in order to compare the important measures of merit for a controller: mean rise time and mean proximity to the desired coordinates. The graphs for the full range of results can be seen in Appendices F and G.

12.2 Selecting gain parameters

The gain parameters used are shown in Table 1:

Controller	PD	PID	Partial PID (see Section 13.2)
K_p^ϕ	5.0	5.0	5.0
K_d^ϕ	4.0	4.0	4.0
K_i^ϕ	0.0	0.0	0.0
K_p^θ	5.0	5.0	5.0
K_d^θ	4.0	4.0	4.0
K_i^θ	0.0	0.0	0.0
K_p^ψ	5.0	5.0	5.0
K_d^ψ	4.0	4.0	4.0
K_i^ψ	0.0	0.06	0.06
K_p^x	$0.3\times\frac{\pi}{180}$	$0.3\times\frac{\pi}{180}$	$0.3\times\frac{\pi}{180}$
K_d^x	$0.5\times\frac{\pi}{180}$	$0.5\times\frac{\pi}{180}$	$0.5\times\frac{\pi}{180}$
K_i^x	0.0	$1.5e-4\times\frac{\pi}{180}$	0.0
K_p^y	$0.3\times\frac{\pi}{180}$	$0.3\times\frac{\pi}{180}$	$0.3\times\frac{\pi}{180}$
K_d^y	$0.5\times\frac{\pi}{180}$	$0.5\times\frac{\pi}{180}$	$0.5\times\frac{\pi}{180}$
K_i^y	0.0	$1.5e-4\times\frac{\pi}{180}$	0.0
K_p^z	$1.2e6$	$1.2e6$	$1.2e6$
K_d^z	$-2.4e6$	$-2.4e6$	$-2.4e6$
K_i^z	0.0	$2.0e5$	$2.0e5$

Table 1: Gain parameters

These gain parameters were obtained by trial and error, using the values employed by Gibiansky in [7] as a starting point. This is certainly not an optimal solution but the values obtained have, as shown in Section 13, given a controller that functions well. In order to use trial and error to manually optimise the tuning parameters it was important to have the correct data. It was desirable to reduce error and rise time but also to reduce overshoot and attempt to have the quadcopter move in as logical a fashion as possible. To this end graphs were obtained to show how the quadcopter 'wanted' to behave in response to error. An example of this is shown in Figure 12 and then, after tuning the gain parameters, in Figure 13. These figures show that by properly tuning the controllers overshoot can essentially be eliminated and a smooth path can be obtained.

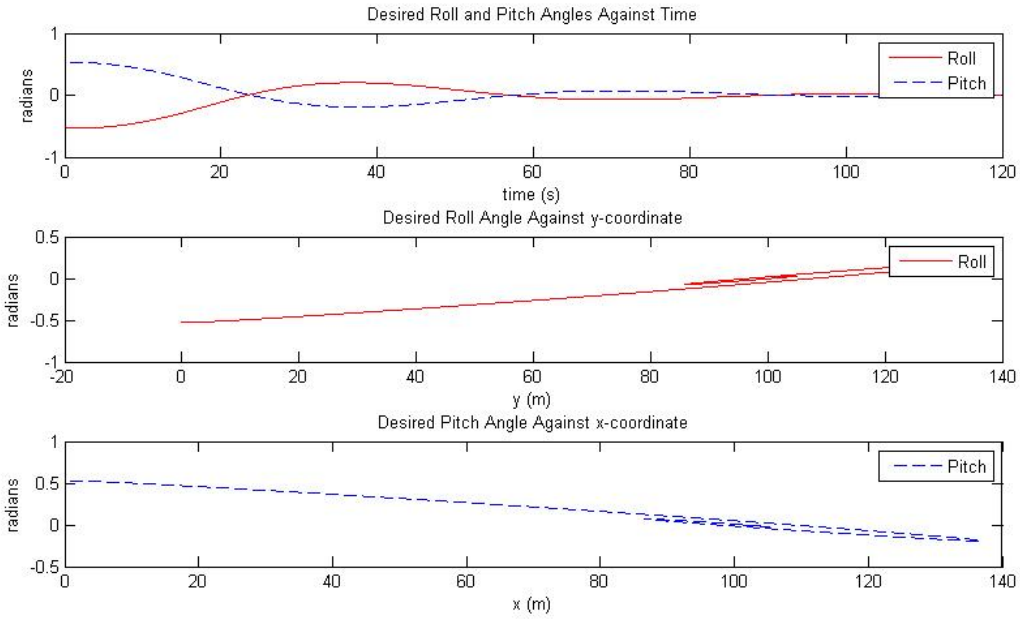


Figure 12: Improperly tuned gain parameters lead to large overshoot and oscillatory motion

When improperly tuned it can be seen from Figure 12 that the desired pitch or roll angle 'spirals' around the desired x or y coordinate respectively, moving through zero many times. This indicates oscillatory motion as the quadcopter struggles to settle on the desired coordinates.

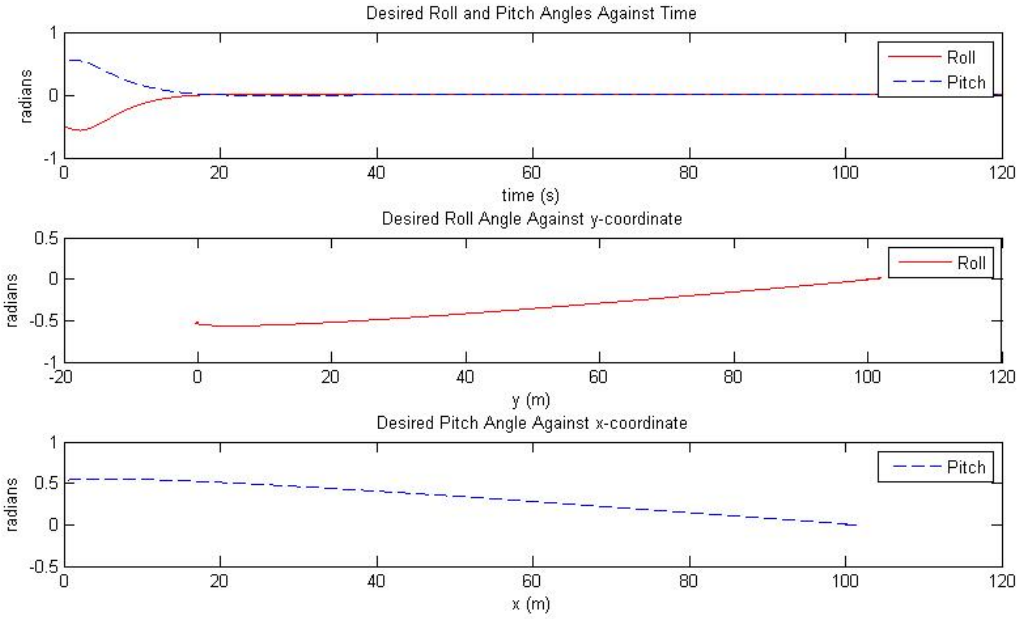


Figure 13: Properly tuned gain parameters will lead to no overshoot or oscillatory motion

When properly tuned it can be seen from Figure 13 that the desired pitch or roll angle does not spiral around the desired x or y coordinate but instead approaches it in a more linear fashion and equals zero

exactly as it reaches it. As discussed in Section 10, there are many superior ways of optimising the gain parameters. Automatic tuning as seen in [7] could be used or else a fuzzy PID controller as in [13] could be used.

12.3 Uncertainty and errors

There are many potential sources of error in the simulation which are discussed in this section alongside the approaches made to limit or eliminate these errors. As there does not appear to be any established conventions on how to approach the reduction of error in the simulation of controllers, the approach made has been based upon that used to reduce error in computational fluid dynamics. Four main sources of error have been identified: discretisation error, iteration error, model error and code error. In doing CFD one must also look out for boundary-condition errors which are not relevant here.

Although the efforts to eliminate error have been detailed below, there is still great uncertainty about the results as all helicopters are highly complex and non-linear systems and therefore controlling them outside of an idealised simulation environment is fraught with complexity.

12.3.1 Discretisation error

Discretisation error is the error due to having a grid or a time step that is not fine enough. In this instance the grid spacing is not an issue but it is possible that the time step could have an effect. It is desirable to obtain 'time independence', i.e. have the time step such that doubling it will not vary the results by more than 0.5%. The important results, as mentioned above, are the rise time and the proximity to the desired coordinates.

It is important to have a time-step that is as small as necessary but no smaller. This is because as the time-step diminishes, the number of computations that are necessary to complete the simulation will balloon and the simulation will become much more burdensome. While this may not matter when running the simulation one time, the simulation will be run several thousand times in order to gather mean data on the performance of the controllers.

The code shall therefore be run for a range of time steps until an appropriate value is obtained and the results presented in Table 2. For the purposes of reducing discretisation error the randomised disturbances are unnecessary and would cloud the results and have therefore been disabled. In Table 2, the spatial coordinates, x, y and z denote the final position of the quadcopter and all entries prefixed by the Greek letter, Δ, denote a percentage change in the pertinent value (given as an absolute value). The simulations are all run for 120s which is far longer than necessary to reach a steady state. The PD controller has been used for these results. The rise time for each time step is designated by R.T.

Time Step (s)	R.T (s)	ΔR.T (%)	x (m)	Δx (%)	y (m)	Δy (%)	z (m)	Δz (%)
0.1	Not reached	N/A	88.54	N/A	111.46	N/A	98.64	N/A
0.05	Not reached	N/A	94.27	6.47	105.73	5.14	98.64	0
0.025	15.4500	N/A	97.14	3.04	102.86	-2.71	98.64	0
0.0125	14.9000	3.56	98.57	1.47	101.43	1.39	98.64	0
$6.25e-3$	14.6563	1.64	99.28	0.73	100.72	0.71	98.64	0
$3.125e-3$	14.5438	0.77	99.64	0.36	100.36	0.36	98.64	0
$1.5625e-3$	14.4859	0.40	99.82	0.18	100.18	0.18	98.64	0
$7.8125e-4$	14.4586	0.19	99.91	0.09	100.09	0.09	98.64	0

Table 2: Effect of time-step on the accuracy of results

From the results in Table 2 we can see that a value between 3.125×10^{-3} and 1.5625×10^{-3} will be appropriate and therefore for the purposes of this paper a time step of 2.0×10^{-3}s will be used.

12.3.2 Iteration error

Iteration error is error due to a steady state not being reached, i.e. the simulation is not allowed to run for long enough to obtain good results. For these simulations there are two possible sources of iteration error: firstly, there could be an error in the final 'location' of the quadcopter if the simulation were not allowed to reach a steady state and the program might indicate that no steady state would be reached when in fact it would be if the simulation were allowed to continue. This kind of error is obviously very easy to root out by simply giving the program much more time than it needs to reach a steady state. From Table 2e we can see that the rise times will always be in the neighbourhood of 15s and although we can expect deviations from this (especially once random disturbances are reintroduced), it is simply unrealistic to expect rise times of greater than two minutes. This is also visible from Figures 9 and 10 (in which disturbances are active); there is no activity visible after the 30s mark.

There is another kind of iteration error that could effect a reading of the results. As stated above, it is the mean results of multiple simulations, rather than how well the controllers perform in single simulations, that are of interest. It is important, therefore, that the simulations are run enough times for the mean results to be representative. Given that the disturbances, though random, will always fall in the same range, we should expect that as the number of simulations increases, the mean results should start to converge on a value. By doubling the number of simulations until the results converge by an acceptable amount, in the same way as the time-step, an appropriate number of simulations to get accurate results can be obtained. The results of this method can be seen in Table 3 (the PID controller has been used), where the mean rise time is designated by M.R.T:

# Simulations	M.R.T (s)	ΔM.R.T (%)	x (m)	Δx (%)	y (m)	Δy (%)	z (m)	Δz (%)
2	14.93		104.31		97.93		100	0
4	15.20	1.81	101.48	2.71	100.76	2.89	100	0
8	15.88	4.47	100.19	1.27	98.70	2.04	100	0
16	15.80	0.50	100.33	0.14	100.98	2.31	100	0
32	15.86	0.38	100.69	0.36	98.51	2.45	100	0
64	16.06	1.26	99.64	1.04	98.64	0.13	100	0
128	15.66	2.49	100.58	0.94	99.34	0.71	100	0
256	16.14	3.07	100.38	0.20	99.60	0.26	100	0
512	16.09	0.31	100.14	0.24	99.69	0.09	100	0
1024	16.16	0.44	100.30	0.16	99.57	0.12	100	0
2048	16.19	0.19	100.34	0.04	99.60	0.03	100	0
4096	16.14	0.31	100.34	0.00	99.64	0.04	100	0

Table 3: Effect of number of simulations on the mean results

The randomised disturbances heavily effect the results such that at low numbers of simulations the results do not seem to converge and the variation in mean rise time fluctuates up to around 5%. However, at higher numbers of simulations the variation drops down to less than 0.5% and the mean rise times start to approach a value around 16.14s. Similarly the final x coordinate values converge at around 100.34 and the final y coordinate values converge at around 99.64. The final z coordinate values are always exactly 100 as the integral gain parameter completely eliminates the steady state error in the z direction. A second set of results was obtained to corroborate this one, available in Appendix E, and when 4096 simulations were computed all of the values were within 0.1%. A plot of the mean rise times against the number of simulations can be seen in Figure 14.

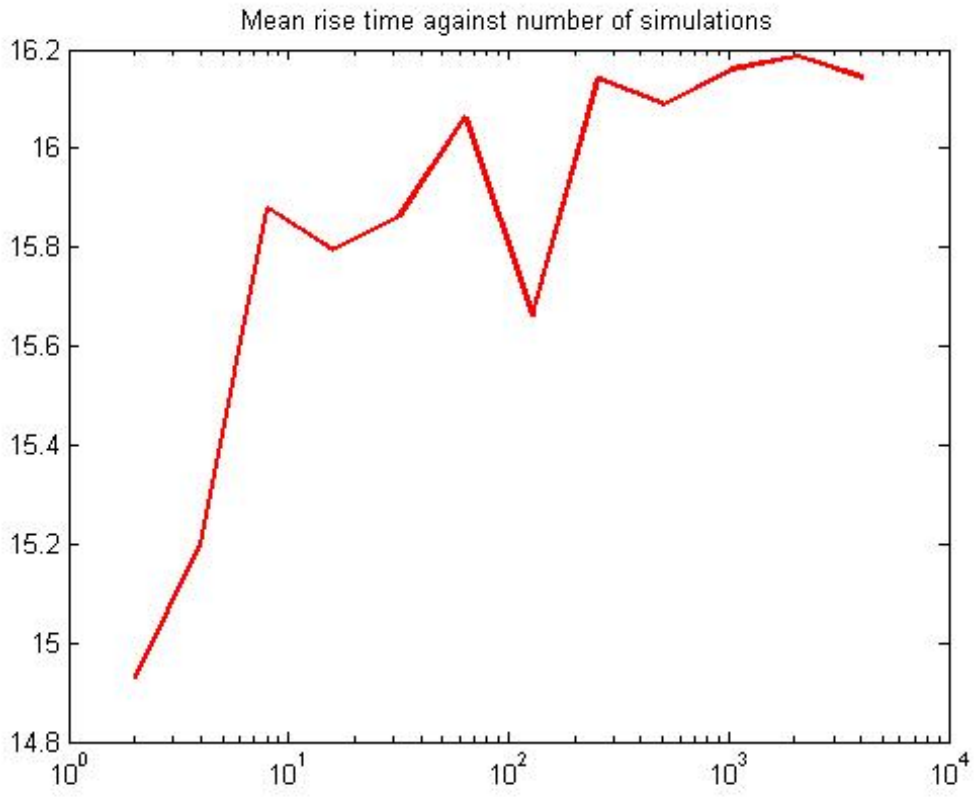


Figure 14: Mean rise times against no. of simulations

However, 4096 simulations is extremely computationally burdensome so a compromise must be made between reducing the iteration error and the time taken to obtain results. 1024 simulations seems to be an appropriate number as doubling or halving the number of simulations will vary the results by less than 0.5% (according to either set of results).

12.3.3 Modelling error

Modelling error is error due to a fault with the model itself or due to the incorrect equations being used. This is potentially a large source of error in these simulations as several simplifications have been made such as ignoring hub forces and certain aerodynamic effects. Aerodynamic friction has been modelled as simply being proportional to quadcopter velocity, the model is also based upon Momentum Theory which is less accurate than Blade Element Theory and lastly, the model does not take into account wind gusts that may knock the quadcopter off course.

There are a number of ways that modelling error can be reduced. Firstly, every effort to ensure the validity of the derivation of quadcopter dynamics has been made. Secondly, the simplifications that been made to the model are all supported by the literature review; the work done in [5],[7] and [8] all make similar simplifications.

In terms of future work, the two most important ways in which the simulation could be made more realistic would be to introduce a randomised wind gust into the code and to alter the model to use Blade Element Theory rather than Momentum Theory.

12.3.4 Code error

Code error is error due to the numerical method being implemented incorrectly. As analytic solutions would be extremely difficult to obtain, the main method by which this kind of error can be eradicated is simply checking the code against the mathematical model and ensuring that there are no discrepancies.

13 Results from the PID controllers

In this section the results themselves are discussed, having established the method for obtaining results, as well as their validity, in Section 12. The full range of results can be viewed in Appendices F and G.

13.1 Effect of integral gain on eliminating steady-state error

The most noticeable change when using the PID controller is that the steady-state error is the z direction is completely eradicated. When using a simple Proportional-Derivative controller is used, aiming for a z-coordinate of 100, the quadcopter will always end up at 98.6375. Although this is a small error it may be more pronounced when attempting more precise movements with the quadcopter. This will be discussed in Section 13.3.

Introducing an integral gain parameter has excellent results in the z direction and steady state error is completely eradicated. Even when looking at low numbers of simulations (or indeed, single simulations), there is zero error which indicates that the error is zero every time and does not just have a mean value of zero. This can be seen from Table 3 or from the plots in Appendix G.

Despite the efficacy of integral term in eradicating steady state error in the z direction, in the x and y directions the performance is not as impressive. The most accurate results are yielded when looking at the mean results of the most number of simulations so the results of the PD and PID controllers when simulated 4096 times shall be compared in Table 4. As previously stated the desired coordinates are [100,100,100].

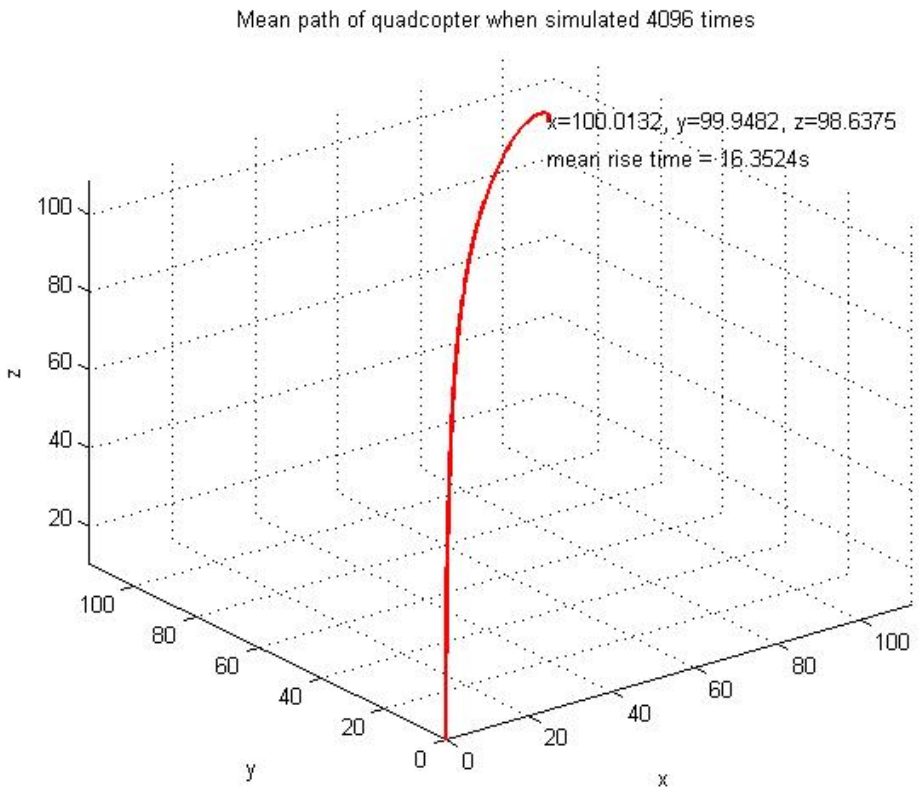


Figure 15: Mean results of 4096 simulations for the PD controller

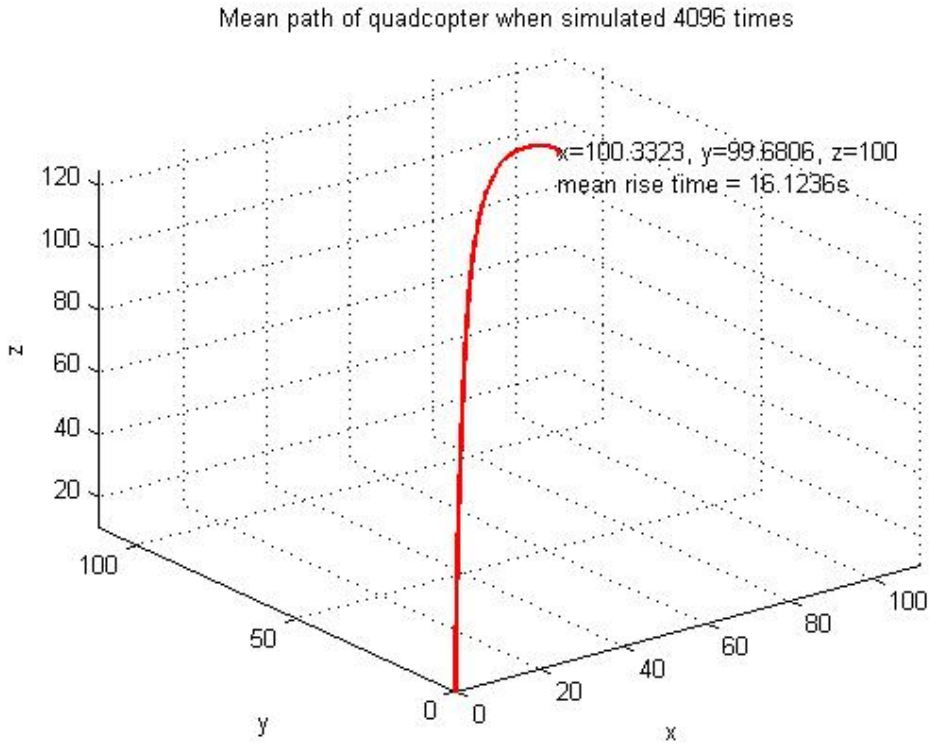


Figure 16: Mean results of 4096 simulations for the PID controller

Controller	Error in x (%)	Error in y (%)	Error in z (%)	Mean Rise Time (s)
PD	0.0132	0.0518	1.3625	16.3524
PID	0.3323	0.3194	0.0	16.1236

Table 4: Effect of integral gain on eliminating steady-state error

We can therefore see a slight reduction in rise time by using the PID controller rather than the PD. The more striking result however, is that the steady state errors in the x and y directions actually increases when using an integral term. The largest error was in the z direction for the PD controller and this was successfully eliminated with an integral term but the error in x and y directions, which was negligible for the PD controller, has risen to around 0.3% when using the PID controller. This result raises the possibility of only using an integral term for the z direction. The PID controller can easily be modified just by setting the values of K_i^x and K_i^y equal to zero.This gives the results shown in Table 5.

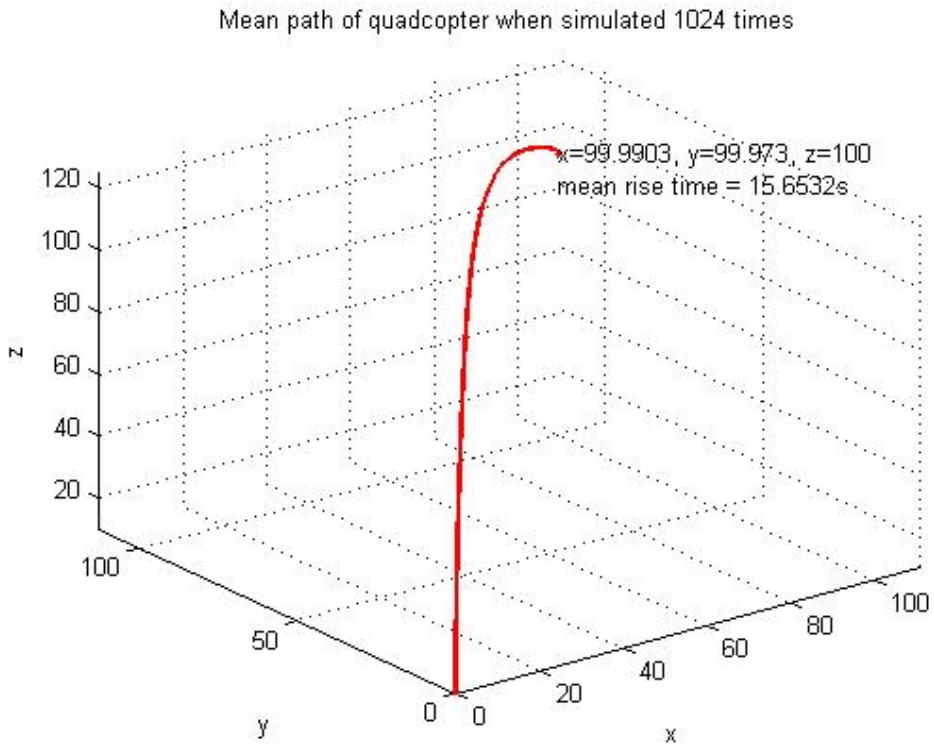


Figure 17: Mean results of 1024 simulations for the partial PID controller

Controller	Error in x (%)	Error in y (%)	Error in z (%)	Mean Rise Time (s)
Partial PID	0.0097	0.027	0.0	15.6532

Table 5: Effect of partial PID on eliminating steady-state error

The results in Table 5 show an improvement in all areas; the errors are all less than 0.03% and the mean rise time has been reduced.

13.2 Effect of random disturbances

The effect of the random disturbances can be seen in Figure 18.

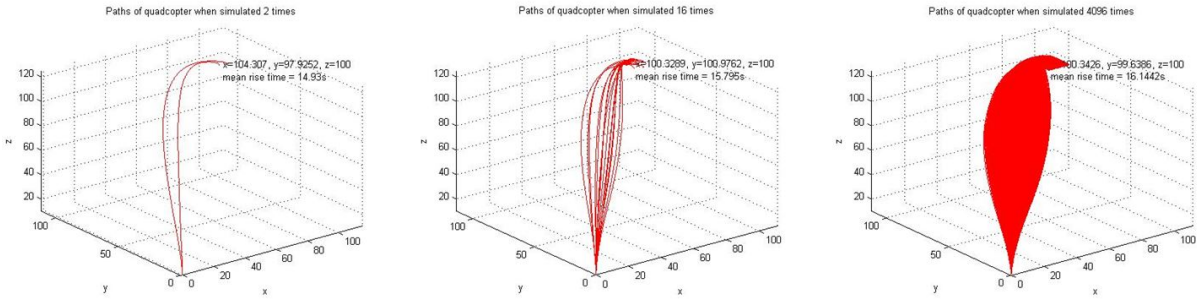


Figure 18: Effect of random disturbances

The disturbances throw the path of the quadcopter away from a linear route and when the full range of possible paths are examined (as in Figure 18) it can be seen that it forms a cobra's head like shape.

This shape can be explained by the way that the PID controllers have been designed. The desired x and y coordinates are reached by setting the desired roll and pitch angles to be proportional to the error in x and y. This means that when the disturbance alters the yaw angle of the quadcopter it will still roll and pitch and therefore give motion in the incorrect direction. The quadcopter will still reach the desired coordinates as there is a controller that corrects the yaw angle and besides, unless disturbance is very large then the motion of the quadcopter is likely to have components in the desired direction. This means that the quadcopter will drift off course, correct it's heading and then head straight for the desired coordinates.

13.3 Efficacy and accuracy of the controller

So far all of the simulations have been testing the performance of the controllers with respect to getting the quadcopter to the same coordinate ([100,100,100]). To fully evaluate the performance of a controller a range of coordinates must be used as it may perform well over long distances where small steady state errors are negligible but behave worse over short distances where the user may be more sensitive to smaller errors. The mean results of 1024 simulations (to do this many simulations takes around 90 minutes and, as there are diminishing returns from doing more as proved in Section 12.3.2, 1024 is sufficient for this demonstration) for the PD controller are in Table 6, given to 2 decimal places.

Des. Coordinates	Final Coordinates	x-error (%)	y-error (%)	z-error (%)	Rise Time (s)
[0.1,0.1,0.1]	[0.09,0.10,-1.26]	8.95	3.22	1362.5	Not reached
[0.5,0.5,0.5]	[0.51,0.47,-0.86]	1.08	5.78	272.5	Not reached
[1,1,1]	[0.97,1.02,-0.36]	3.14	1.6	136.25	Not reached
[5,5,5]	[5.02,4.98,3.64]	0.34	0.37	27.25	3.64
[10,10,10]	[10.01,9.96,8.64]	0.14	0.39	11.37	Not reached
[25,25,25]	[25,25,23.64]	0.02	0.01	5.45	Not reached
[50,50,50]	[50.01,50.02,48.64]	0.02	0.04	2.73	15.87
[100,100,100]	[100.02,99.98,98.64]	0.019	0.02	1.36	14.84
[200,200,200]	[199.97,199.95,198.64]	0.02	0.02	0.68	24.25
[500,500,500]	[499.87,499.84,498.64]	0.03	0.03	0.27	30.04
[1000,1000,1000]	[1005.06,998.28,998.64]	0.51	0.17	0.14	51.05

Table 6: Efficacy and accuracy of the PD controller

The most obvious conclusion to draw from the results in Table 6 is that the error in the z direction is unacceptably large over small distances. For all three simulations where the desired z coordinates were less than 5, the simulation yielded a negative result for the final z coordinate. This indicates that the quadcopter would have hit the floor. It is also interesting to note that the error in the z direction never changes, it is always equal to 1.3625. The controller performs much better in the x and y directions although the error is considerably higher over very small or very large distances. At any distance above 5m and below 500m it can be seen that the quadcopter will reach its destination to within 0.5%. However, there is quite often an error of between 1 and 5cm which is much more noticeable over short distances.

It is also worth noting that the simulation shows that as the quadcopter tries to reach distances of 1000m in each direction the error begins to increase again. This shows that the controller is very effective over short to medium distances but when given very short or very long distances to track, it performs less well.

The mean results of 1024 simulations for the partial PID controller are in Table 7:

Des. Coordinates	Final Coordinates	x-error (%)	y-error (%)	z-error (%)	Rise Time (s)
[0.1,0.1,0.1]	[0.11,0.13,0.1]	9.44	29.7	0.0	Not reached
[0.5,0.5,0.5]	[0.49,0.49,0.5]	2.04	1.4	0.0	Not reached
[1,1,1]	[1.01,0.95,1.0]	0.8	4.61	0.0	43.44
[5,5,5]	[4.99,4.97,5]	0.2	0.55	0.0	16.38
[10,10,10]	[9.99,9.98,10.0]	0.06	0.25	0.0	16.11
[25,25,25]	[24.99,25.02,25.0]	0.04	0.07	0.0	16.40
[50,50,50]	[50.01,50,50]	0.02	0.04	0.0	16.29
[100,100,100]	[99.96,99.99,100]	0.04	0.01	0.0	15.63
[200,200,200]	[199.98,200,200]	0.01	0.0	0.0	19.55
[500,500,500]	[499.94,499.75,500]	0.01	0.05	0.0	31.86
[1000,1000,1000]	[975.33,1004.09,1000.28]	2.47	0.41	0.03	55.52

Table 7: Efficacy and accuracy of the partial PID controller

The results in Table 7 corroborate much of what was seen in simulations for the PD controller (as they are identical except in terms of yawing action and motion in the z direction). Again it is seen that the performance of the controller slips over the very short and very long ranges. However, the use of an integral term dramatically improves the performance in the z direction. There is now no error over any range less than 1000m so that the controller could be used to navigate the quadcopter over very short distances without it hitting the floor, as would be an issue with the PD controller.

The poorer performance over larger distances is probably due to the tuning parameters being more suited for shorter distances. This is indicated by Figure 19 in which it can be seen that the shape of the path taken by the quadcopter is much less straight forward than that taken when the quadcopter is travelling over shorter distances (as can be seen in Figure 17).

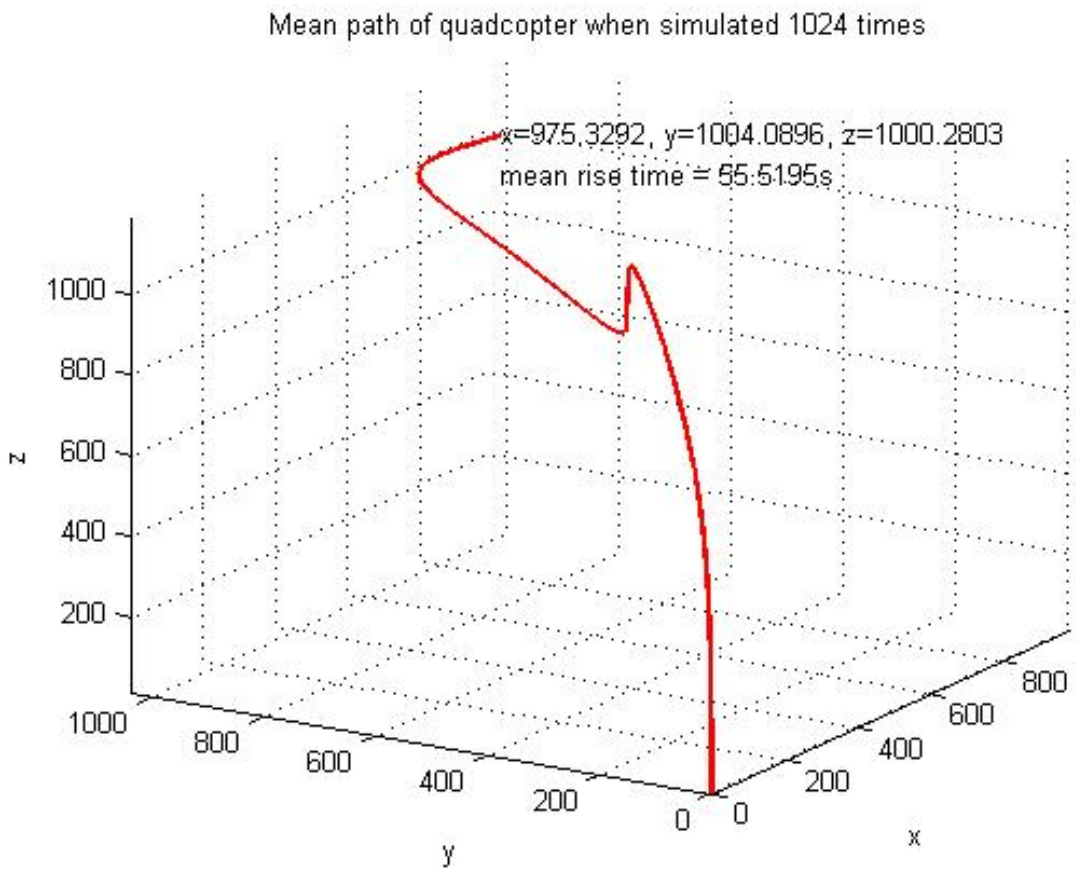


Figure 19: Poor controller performance over long distances

13.4 Limitations of PID controllers

13.4.1 Linearity

As mentioned throughout the project, all helicopters are high order, non-linear systems whereas PID controllers are linear. This means that there is significant 'plant uncertainty', i.e. one cannot be exactly certain as to how the quadcopter will behave, and therefore it is not guaranteed that the controller will perform as well in reality as it has in simulations.

13.4.2 Noise

Another major problem is that, would the controllers designed in this project to be used in reality, there could be a lot of noise in the signal from the sensors. Therefore the signal may read very large changes of error in the system, the derivative of which would amplify the signal and cause the quadcopter to behave erratically. Although it is possible to remove the derivative term and have a PI controller, it is possible to use electronic filters to process the signals and remove unwanted high-frequency constituents and enhance desirable ones. This will result in a 'smoother' signal without spikes in the error and therefore the derivative term can still be used.

13.4.3 Windup

Wind up, and in particular integral windup, is a phenomenon whereby the output from the integrator exceeds the capacity of the actuator. In practice this would mean that the propellers could not spin any faster but the error will continue to be integrated and therefore the integrator will continue to grow and can cause the system to become unstable. This problem can be addressed in many ways, most obviously by setting a limit on the value of the integral term.

14 Future Work

There is a multitude of ways, many of them already discussed, in which the work in this project could be expanded upon.

14.1 Improving the model

The model of quadcopter dynamics should be expanded and refined. It should be made to use Blade Element Theory rather than Momentum Theory, it should incorporate external forcing conditions such as wind gusts and it should take hub forces into account.

14.2 Improving the PID controllers

As discussed elsewhere in the project the PID controller could be improved in a number of ways. A hybrid fuzzy-PID controller could be used to improve the response of the controller, filters could be used to eradicate signal noise (in the event of a practical experiment taking place), and the gain parameters could be automatically tuned which optimise the controller performance and improve performance, particularly over very short and very long distances.

14.3 Develop other types of controllers

As discussed in the literature review, there are many types of controllers that could be used to operate a quadcopter and indeed would probably be better suited to the task. Non-linear controllers such as H_∞ controllers in particular would be an avenue worth investigating as they would handle the non-linear effects much better than a PID controller. There are other linear controllers such as LQR controllers that could also be tested against the PID controllers performance.

By putting the equations of motion into state space form and linearising them, MATLAB can be used to obtain a transfer function that could in turn be used in Simulink to automatically generate a very wide variety of controllers. This would be an excellent method of comparing the respective merits of multiple controllers.

14.4 Improve 3D visualisation

The 3D visualisation utilised in this project was very limited and just took the form of a marker denoting the position of the quadcopter at any given time. This was a useful aid as it allows a user to judge the speed, acceleration and deceleration as well as situations in which it arrives in the vicinity of the destination very quickly but does not satisfy the rise time criteria. However, a more sophisticated 3D visualisation, such as the one shown in Figure 8, would allow the user to see more information about the orientation and the angular velocity and acceleration of the quadcopter.

It would also be able to give the quadcopter real time commands so that its performance in changing direction could be evaluated.

14.5 Experiment

Ideally any future work should involve implementing the controller to operate an actual quadcopter. This would give much better insight into the disparity between simulation of the controllers and their actual performance in a real life situation.

15 Conclusion

In this project, the aims were defined as:

- Derive a mathematical model for helicopter dynamics.
- Design a low complexity controller that will keep the helicopter stable and allow it to follow a defined trajectory.
- Test the controller using a simulation in MatLab.
- If possible, to design a 3D visualization of the helicopters motion in MatLab.

All of these aims have been successfully completed with varying degrees of success. In Section 4, quadcopters were selected as the rotor configuration that this project would focus upon. Section 5 contained a derivation of quadcopter kinematics, giving a means by which vectors could be mapped from the inertial to the body-fixed frames of reference and thus allow us to model quadcopter motion. Section 6 contained a catalogue of the forces and moments that act upon the quadcopter and then the findings of Sections 5 and 6 were combined to give the quadcopter's equations of motion in Section 7. This completes the first aim of the project, to derive a mathematical model for helicopter dynamics.

Having completed the quadcopter model, Sections 8 and 9 contained rudimentary PD and PID controllers that were only built to stabilise the quadcopter but not actually control it. A brief literature review was given in Section 10 that discussed various means of controlling and simulating quadcopters. Section 11 then focused on developing the rudimentary controllers of Sections 8 and 9 so that they could be used to give the quadcopter coordinates to follow. This completes the second aim of the project, a low-complexity controller that both stabilises the quadcopter and allows a user to send it to given coordinates. These controllers were then simulated in Section 12 alongside an extended discussion of the validity of the results of those simulations. This section completes the final two aims of the project, to simulate the quadcopter's motion and to give an appropriate 3D visualisation.

The results of the simulations were then discussed in Section 13 and the limitations of the technique used (Proportional-Integral-Derivative control) were discussed. This led onto Section 14 where the possibilities for expanding and improving upon the project were discussed.

16 Appendix A: Position Dynamics Additional Steps

Some steps detailing the derivation of the two rotation matrices $R^T(\theta)$ and $R^T(\psi)$ were eliminated from the derivation presented in Section 5.3 for the sake of brevity. These are presented here:

16.1 Mapping from F_3 to F_2

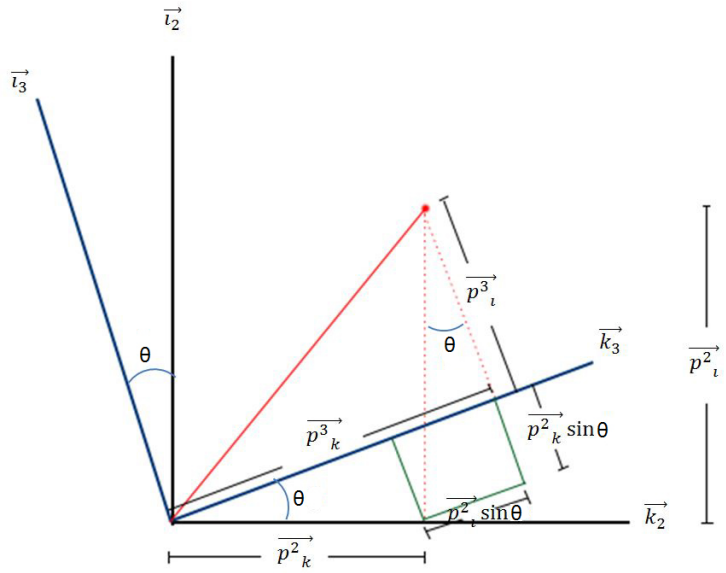


Figure 20: Mapping from F_3 to F_2

From Figure 20 it can be made out that:

$$p_i^3 = p_i^2 \sin\theta - p_k^2 \sin\theta \tag{117}$$

$$\vec{p_j^3} = \vec{p_j^2} \tag{118}$$

$$\vec{p_k^3} = \vec{p_k^2} \cos\theta + \vec{p_i^2} \sin\theta \tag{119}$$

Therefore:

$$\begin{bmatrix} \vec{p_i^3} \\ \vec{p_j^3} \\ p_k^3 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \vec{p_i^2} \\ \vec{p_j^2} \\ \vec{p_k^2} \end{bmatrix} \tag{120}$$

We now have our rotation matrix that will map vectors from F_B to F_3:

$$\begin{bmatrix} \hat{i_3} \\ \hat{j_3} \\ \hat{k_3} \end{bmatrix} = R^T(\theta) \begin{bmatrix} \hat{i_2} \\ \hat{j_2} \\ \hat{k_2} \end{bmatrix} \tag{121}$$

Where:

$$R^T(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{122}$$

16.1.1 Mapping from F_2 to F_1

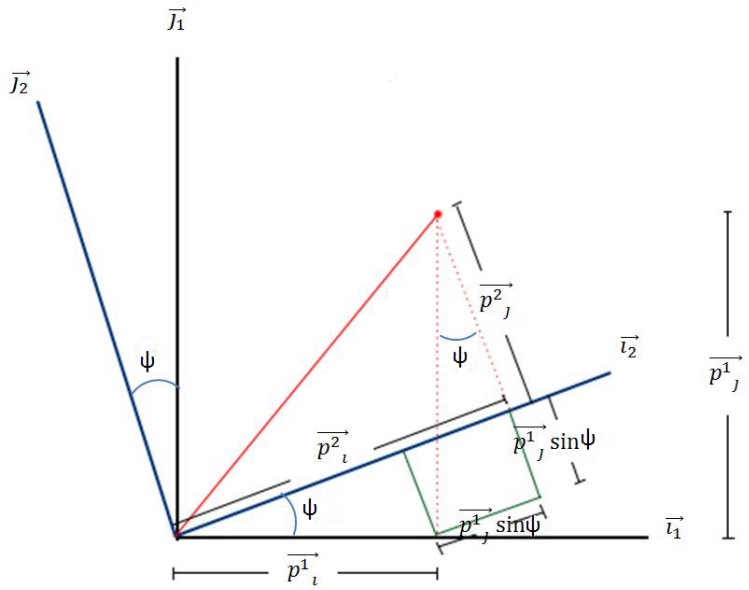


Figure 21: Mapping from F_2 to F_1

From Figure 21 it can be made out that:

$$p_i^2 = p_i^1 \cos\psi + p_j^1 \sin\psi \tag{123}$$

$$\vec{p_j^2} = \vec{p_j^1} \cos\psi - \vec{p_i^1} \sin\psi \tag{124}$$

$$\vec{p_k^2} = \vec{p_k^1} \tag{125}$$

Therefore:

$$\begin{bmatrix} \vec{p_i^2} \\ \vec{p_j^2} \\ \vec{p_k^2} \end{bmatrix} = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \vec{p_i^1} \\ \vec{p_j^1} \\ \vec{p_k^1} \end{bmatrix} \tag{126}$$

We now have our rotation matrix that will map vectors from $\mathcal{F}_B$ to $\mathcal{F}_3$:

$$\begin{bmatrix} \hat{i_2} \\ \hat{j_2} \\ \hat{j_2} \end{bmatrix} = R^T(\psi) \begin{bmatrix} \hat{i_1} \\ \hat{j_1} \\ \hat{k_1} \end{bmatrix} \tag{127}$$

Where:

$$R^T(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{128}$$

17 Appendix B: MatLab Code

17.1 Code to map vectors from $\mathcal{F}_B$ to $\mathcal{F}_I$:

```
function R = rotation(angles)

    phi=angles(1);
    theta=angles(2);
    psi=angles(3);

    R=zeros(3);

    R(1,:) = [
    cos(theta)*cos(psi)
    cos(psi)*sin(phi)*sin(theta)-cos(phi)*sin(psi)
    sin(phi)*sin(psi)+cos(phi)*cos(psi)*sin(theta)
    ];

    R(2,:) = [
    cos(theta)*sin(psi)
    cos(phi)*cos(psi) + sin(phi)*sin(theta)*sin(psi)
    cos(phi)*sin(theta)*sin(psi)-cos(psi)*sin(phi)
    ];

    R(3,:) = [
    -sin(theta)
    cos(theta)*sin(phi)
    cos(phi)*cos(theta)
    ];

%This rotation matrix maps between the body-fixed frame and the inertial %frame.

end
```

17.2 Code to convert from angular velocities to time derivatives of the Euler angles:

```
function thetadot = omega2thetadot(omega,angles)

    phi=angles(1);
    theta=angles(2);
    psi=angles(3);

    A=zeros(3);

    A(1,:)=[
    1
    0
    -sin(theta)
    ];

    A(2,:) =[
    0
    cos(phi)
    cos(theta)*sin(phi)
    ];

    A(3,:) =[
    0
    -sin(phi)
    cos(phi)*cos(theta)
    ];

    W=inv(A);
    thetadot=W*omega;

%This function converts from angular velocity to the time derivatives
%of roll, pitch and yaw.

end
```

17.3 Code to compute the quadcopter's translational acceleration

```
function a= acceleration(orientation,m,K,inputs,velocities,Cx,Cy,Cz,g);

    gravity = [0 0 -g].';
    thrust_FB = K*sum(inputs); %thrust in the body-fixed frame
    R = rotation(orientation); %rotation matrix to map vectors from F_B to F_I
    thrust_FI = R*thrust_FB; %thrust in the inertial frame
    drag = [-Cx*velocities(1);-Cy*velocities(2);-Cz*velocities(3)];

    a = gravity + 1/m*thrust_FI + 1/m*drag;

%This function computes the acceleration of the quadcopter based upon
%its orientation, mass, thrust proportionality constant, inputs (the
%squares of each propeller's angular velocity, the linear velocity, the
%drag constants and the acceleration due to gravity.

end
```

17.4 Code to compute the quadcopter's rotational acceleration

```
function a = angular_acceleration(I,l,K,inputs,angular_velocities);

    i1=inputs(1);
    i2=inputs(2);
    i3=inputs(3);
    i4=inputs(4);

    p=angular_velocities(1);
    q=angular_velocities(2);
    r=angular_velocities(3);

    Ixx=I(1,1);
    Iyy=I(2,2);
    Izz=I(3,3);

    actuator_action=[
        l*K*(i2-i4);
        l*K*(i1-i3);
        0;
        ];

    dragtorque=[
        0;
        0;
        B*(i1-i2+i3-i4);
        ];

    gyrotorque=[
        (Iyy-Izz)*q*r;
        (Izz-Ixx)*p*r;
        (Ixx-Iyy)*p*q;
        ];

a = inv(I)*(actuator_action + gyrotorque + dragtorque);

%This function computes the angular acceleration of the quadcopter
%based upon the inertia matrix, the length of the quadcopter arms, the
%thrust proportionality constant, the inputs (squares of the angular
%velocities of the propellers) and the angular velocity of the
%quadcopter.

end
```

17.5 Code for a PD controller

```
function [inputs,previous_error2] = PD(m,g,K,B,I,l,start_time,end_time...
    ,dt,orientation,desired_orientation,previous_error)
%eulerdot represents the time derivatives of the euler angles.

    inputs=zeros(4,1);
    previous_error2=zeros(3,1); %At the end of the code the function will set
                                %the error as the new previous error. In
                                %themain simulator code previous_error2
                                %will be set as the previous_error argument
```

```
                                %for this function.

    Ixx=I(1,1);
    Iyy=I(2,2);
    Izz=I(3,3);

    phi=orientation(1);
    theta=orientation(2);
    psi=orientation(3);

    Kp_phi=4;  %the tuning parameters
    Kd_phi=3;
    Kp_theta=4;
    Kd_theta=3;
    Kp_psi=4;
    Kd_psi=3;

    phi_des=desired_orientation(1);
    theta_des=desired_orientation(2);
    psi_des=desired_orientation(3);

    error_phi=phi_des-phi;
    error_theta=theta_des-theta;
    error_psi=psi_des-psi;

    previous_error_phi=previous_error(1);
    previous_error_theta=previous_error(2);
    previous_error_psi=previous_error(3);

    errordot_phi=(error_phi-previous_error_phi)/dt;
    errordot_theta=(error_theta-previous_error_theta)/dt;
    errordot_psi=(error_psi-previous_error_psi)/dt;

    thrust=m*g/(K*cos(phi)*cos(theta));

    inputs(1)=thrust/4 ...
        + 1/(4*l*B*K)*(-2*B*(Kp_theta*error_theta+Kd_theta*errordot_theta)*Iyy ...
        + l*K*(Kp_psi*error_psi+Kd_psi*errordot_psi)*Izz);

    inputs(2)=thrust/4 ...
        + 1/(4*l*B*K)*(-2*B*(Kp_phi*error_phi+Kd_phi*errordot_phi)*Ixx ...
        - l*K*(Kp_psi*error_psi+Kd_psi*errordot_psi)*Izz);

    inputs(3)=thrust/4 ...
        + 1/(4*l*B*K)*(2*B*(Kp_psi*error_psi+Kd_psi*errordot_psi)*Iyy...
        + l*K*(Kp_theta*error_theta+Kd_theta*errordot_theta)*Izz);

    inputs(4)=thrust/4 ...
        + 1/(4*l*B*K)*(2*B*(Kp_psi*error_psi+Kd_psi*errordot_psi)*Ixx...
        - l*K*(Kp_phi*error_phi+Kd_phi*errordot_phi)*Izz);

    previous_error2(1)=error_phi;
    previous_error2(2)=error_theta;
    previous_error2(3)=error_psi;

%This function computes the propeller angular velocities necessary to obtain
%the desired orientation based upon PD control.
```

```

end
```

17.6 Controlling motion in the x and y directions with a PID controller

```
phi_des= Kp_y*error_y + Kd_y*errordot_y + Ki_y*int_y;
if phi_des>pi/4
    phi_des=pi/4;
end
if phi_des<-pi/4
    phi_des=-pi/4;
end

theta_des= Kp_x*error_x + Kd_x*errordot_x + Ki_x*int_x;
if theta_des>pi/4
    theta_des=pi/4;
end
if theta_des<-pi/4
    theta_des=-pi/4;
end
```

17.7 Controlling motion in the z direction with a PID controller

```
thrust=Kp_z*error_z + Kd_z*errordot_z + Ki_z*int_z;
if thrust<0
    thrust=0;
end
```

17.8 3D, real-time visualization

```
if risetime ~= 0
    figure;
    for i=1:2:gecount
            %If the quadcopter reaches the rise time criteria,
            %the simulation will cut off 10s after the rise time.
        plot3(xs(1:i),ys(1:i),zs(1:i),':r')
        hold on
        scatter3(xs(i),ys(i),zs(i),'r')
        hold off
        axis([xmin,xmax,ymin,ymax,zmin,zmax])
        posstr2=strcat({' x = '},num2str(xs(i))...,
                {', y = '},num2str(ys(i)),{', z = '},num2str(zs(i)));
        text(xs(i),ys(i),0.9*zs(i),posstr2)
        clock=strcat({'  t = '},num2str(times(i)), 's');
        text(xs(i),ys(i),zs(i),clock)
        title('Path of Quadcopter')
        grid on
        drawnow
    end

else
    for i=1:2:length(result.x)
```

```
            %If the rise time criteria are not met,
            %the whole simulation will be shown.
        plot3(xs(1:i),ys(1:i),zs(1:i),':r')
        hold on
        scatter3(xs(i),ys(i),zs(i),'r')
        hold off
        axis([xmin,xmax,ymin,ymax,zmin,zmax])
        posstr2=strcat({' x = '},num2str(xs(i)),{', y = '}...
                ,num2str(ys(i)),{', z = '},num2str(zs(i)));
        text(xs(i),ys(i),0.9*zs(i),posstr2)
        clock=strcat({'  t = '},num2str(times(i)), 's');
        text(xs(i),ys(i),zs(i),clock)
        title('Path of Quadcopter')
        grid on
        drawnow
    end
end
```

18 Appendix C: Orientation Dynamics Additional Steps

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + R^T(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + R^T(\phi)R^T(\theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{129}$$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{130}$$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ \sin\phi\sin\theta & \cos\phi & \cos\theta\sin\phi \\ \cos\phi\sin\theta & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \tag{131}$$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \dot{\theta}\cos\phi \\ -\dot{\theta}\sin\phi \end{bmatrix} + \begin{bmatrix} -\dot{\psi}\sin\theta \\ \dot{\psi}\cos\theta\sin\phi \\ \dot{\psi}\cos\phi\cos\theta \end{bmatrix} \tag{132}$$

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} - \dot{\psi}\sin\theta \\ \dot{\theta}\cos\phi + \dot{\psi}\cos\theta\sin\phi \\ -\dot{\theta}\sin\phi + \dot{\psi}\cos\phi\cos\theta \end{bmatrix} \tag{133}$$

$$\omega^B = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{134}$$

19 Appendix D: Momentum Theory

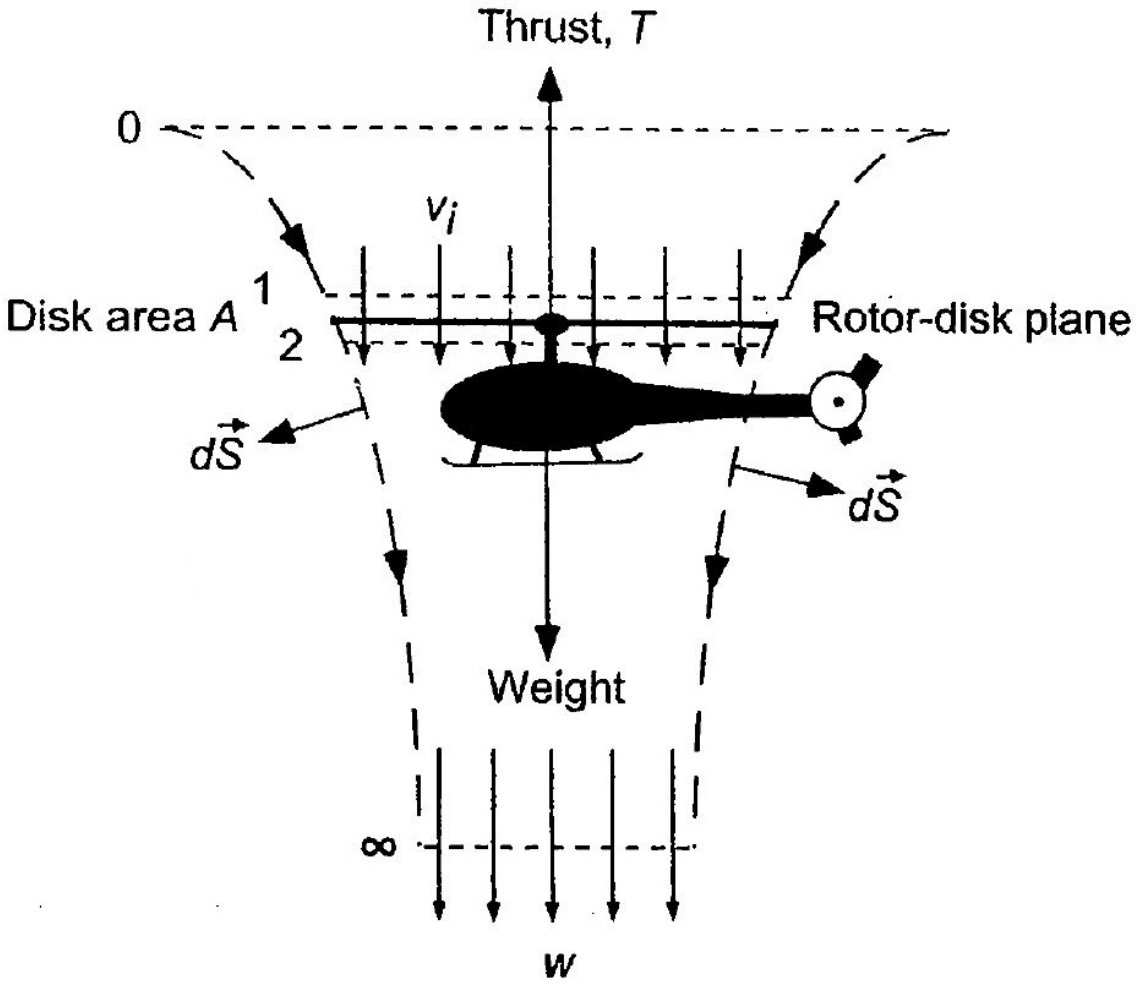


Figure 22: Momentum Theory. Image from [4, p.99]

The mass flow rate across the rotor is given by:

$$\dot{m} = \rho A v_i \tag{135}$$

Where $\dot{m}$ is the mass flow rate, ρ is the air density, A is the rotor disk area and v_i is the induced velocity. Then by Newtons 2^{nd} Law (Force = rate of change of momentum):

$$T = \dot{m} w \tag{136}$$

$$T = \rho A v_i w \tag{137}$$

Where T is the thrust and w is the downwash. Then by energy conservation:

$$T v_i = \frac{1}{2} \dot{m} w^2 \tag{138}$$

Therefore:

$$\rho A v_i w v_i = \frac{1}{2} \rho A v_i w^2 \tag{139}$$

$$v_i = \frac{1}{2}w \tag{140}$$

Substituting this expression for v_i into equation (125):

$$T = \rho A v_i \cdot 2v_i \tag{141}$$

$$T = 2\rho A v_i^2 \tag{142}$$

Therefore:

$$v_i = \sqrt{\frac{T}{2\rho A}} \tag{143}$$

20 Appendix E: Second set of PID results from varying number of simulations

The results in Table 8 corroborate those presented in Section 12.3.2 which suggest that 1024 simulations is sufficient to obtain accurate results and reduce iteration error.

# Simulations	M.R.T (s)	ΔM.R.T (%)	x (m)	Δx (%)	y (m)	Δy (%)	z (m)	Δz (%)
2	16.08		100.60		101.52		100	0
4	15.20	5.47	101.48	0.87	100.76	0.75	100	0
8	15.88	4.47	100.19	1.27	98.70	2.04	100	0
16	15.80	0.50	100.33	0.14	100.98	2.31	100	0
32	15.86	0.38	100.69	0.36	98.51	2.45	100	0
64	16.06	1.26	99.64	1.04	98.64	0.13	100	0
128	15.66	2.49	100.58	0.94	99.34	0.71	100	0
256	16.14	3.07	100.39	0.19	99.60	0.26	100	0
512	16.18	0.25	100.41	0.02	100.21	0.61	100	0
1024	16.12	0.37	100.30	0.11	99.66	0.55	100	0
2048	16.18	0.37	100.34	0.04	99.53	0.13	100	0
4096	16.12	0.37	100.33	0.01	99.68	0.15	100	0

Table 8: Second set of PID results from varying number of simulations

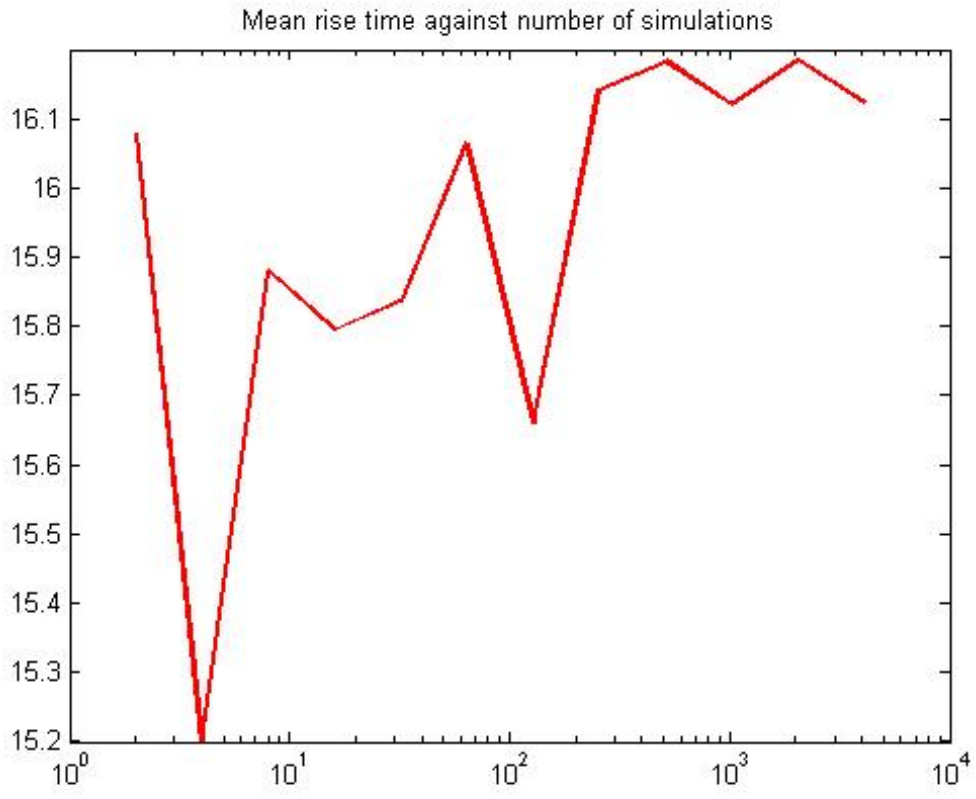


Figure 23: Mean rise times against no. of simulations

21 Appendix F: Graphs of results from the PD controller

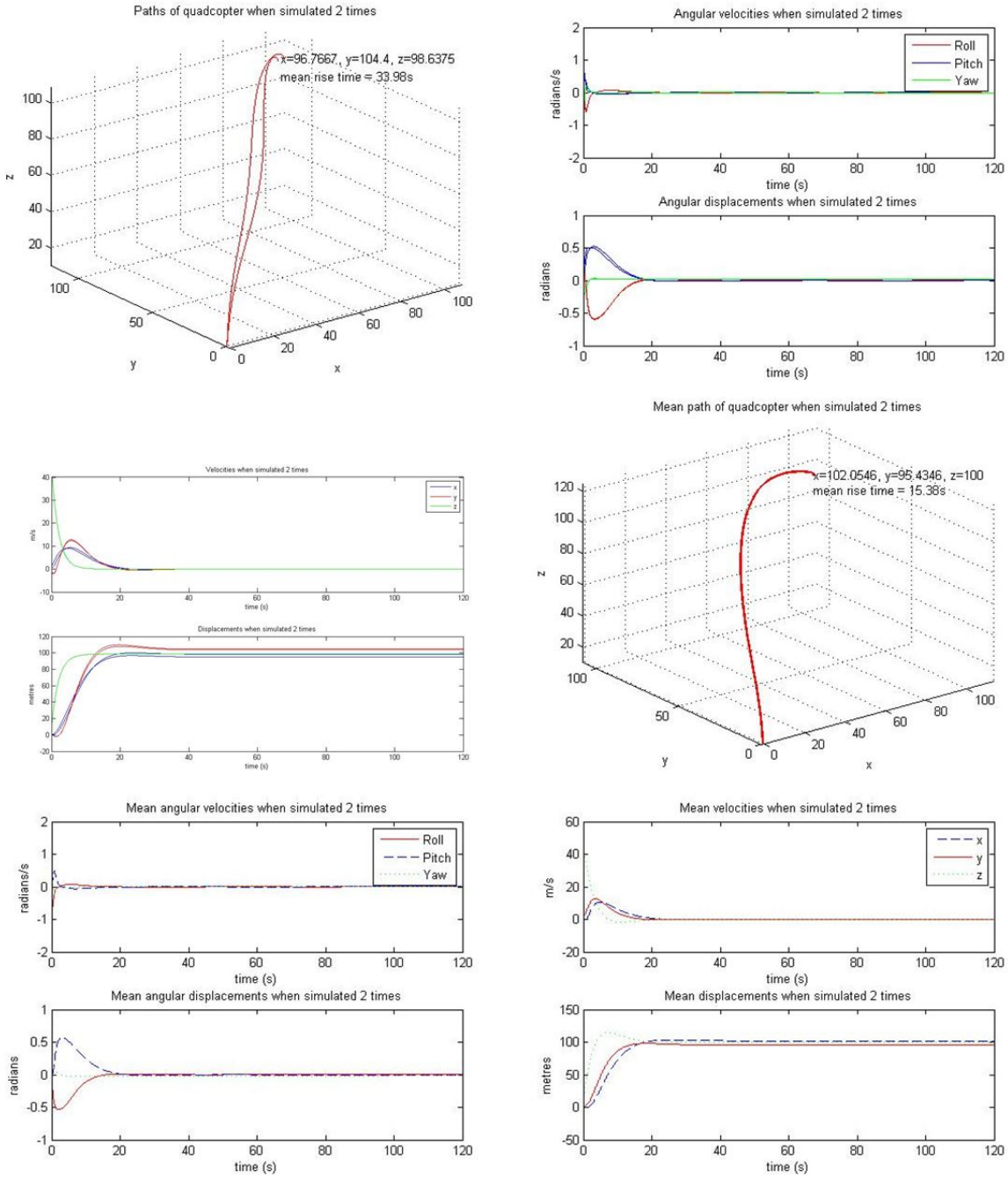


Figure 24: Results of 2 simulations

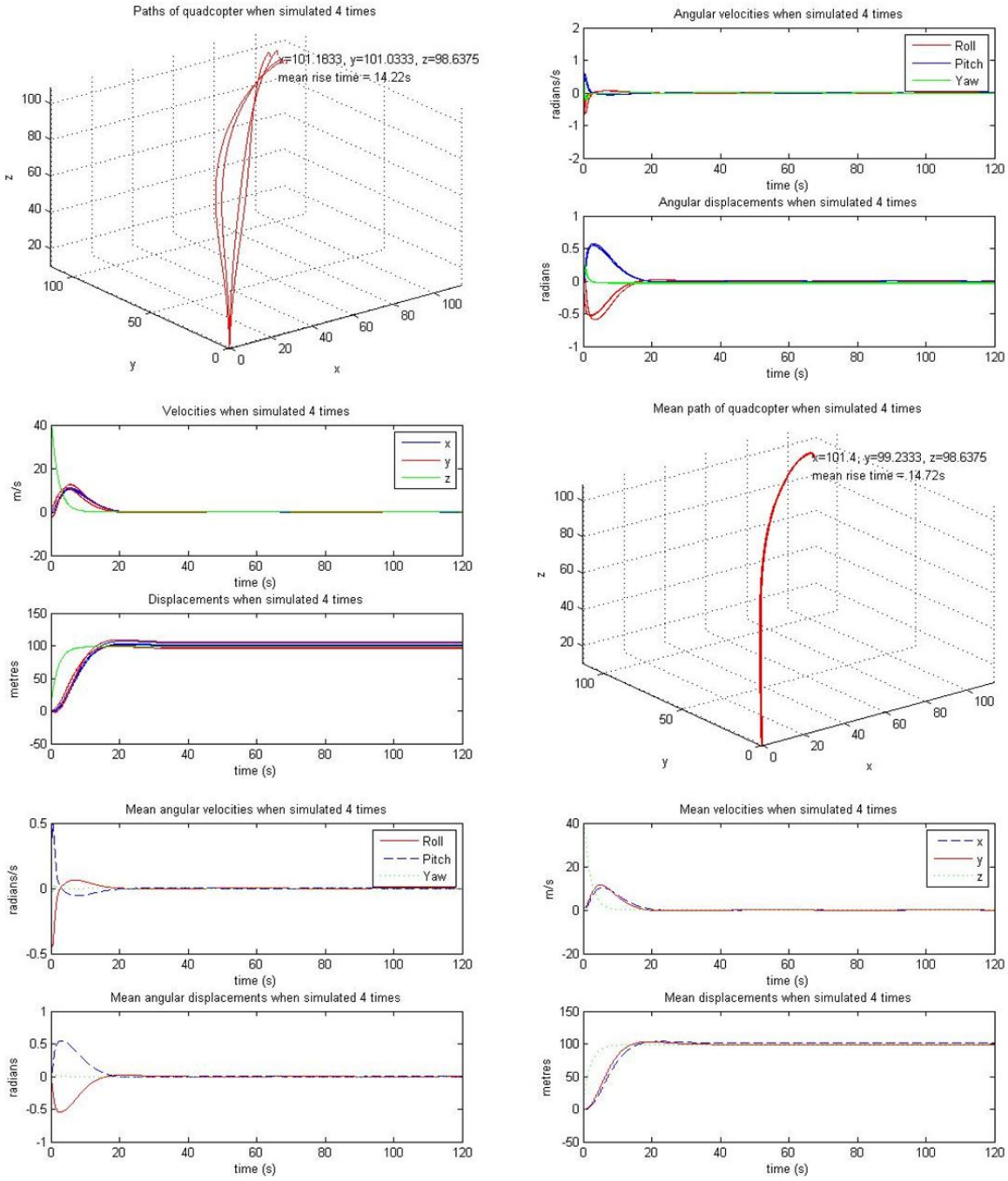


Figure 25: Results of 4 simulations

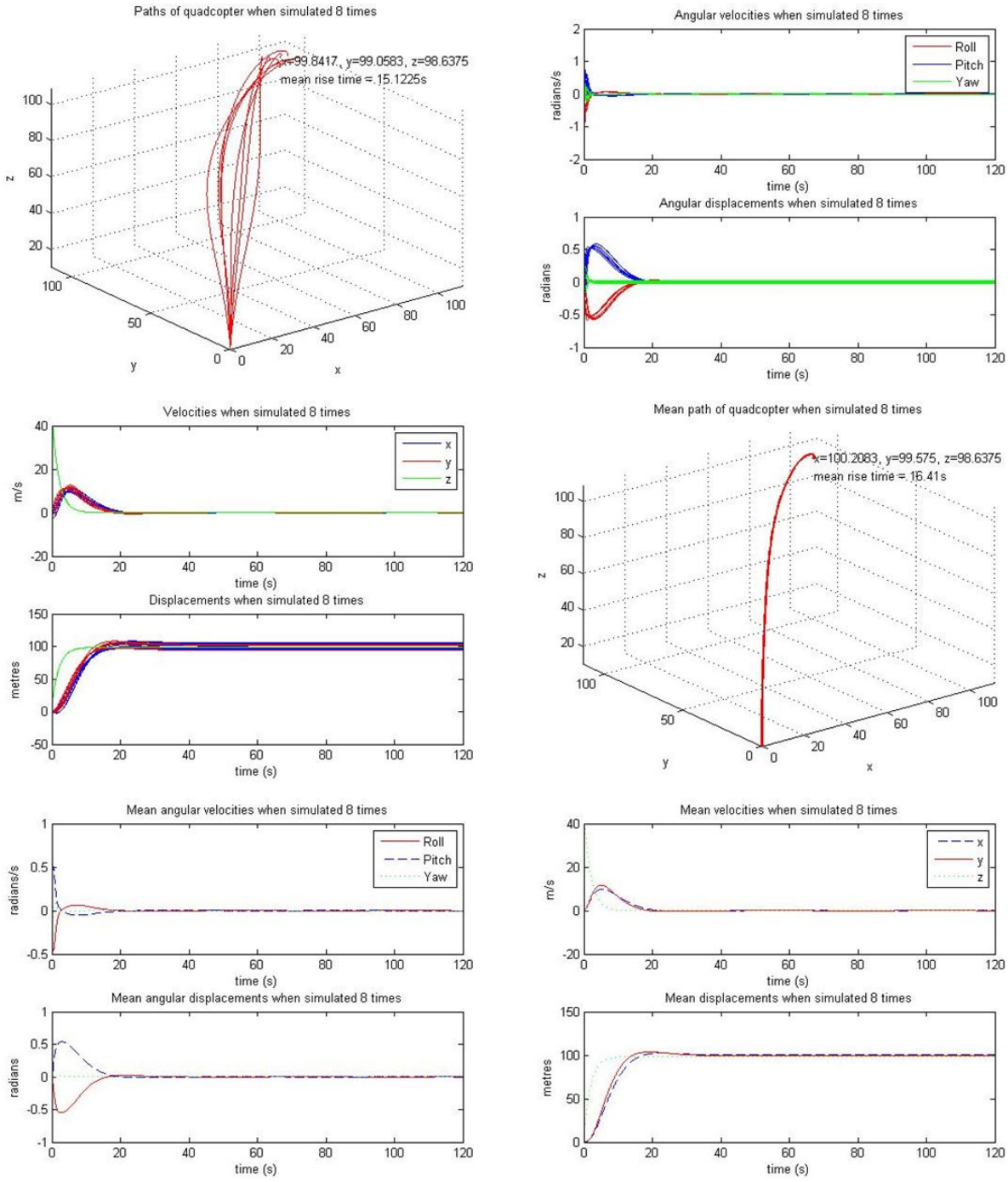


Figure 26: Results of 8 simulations

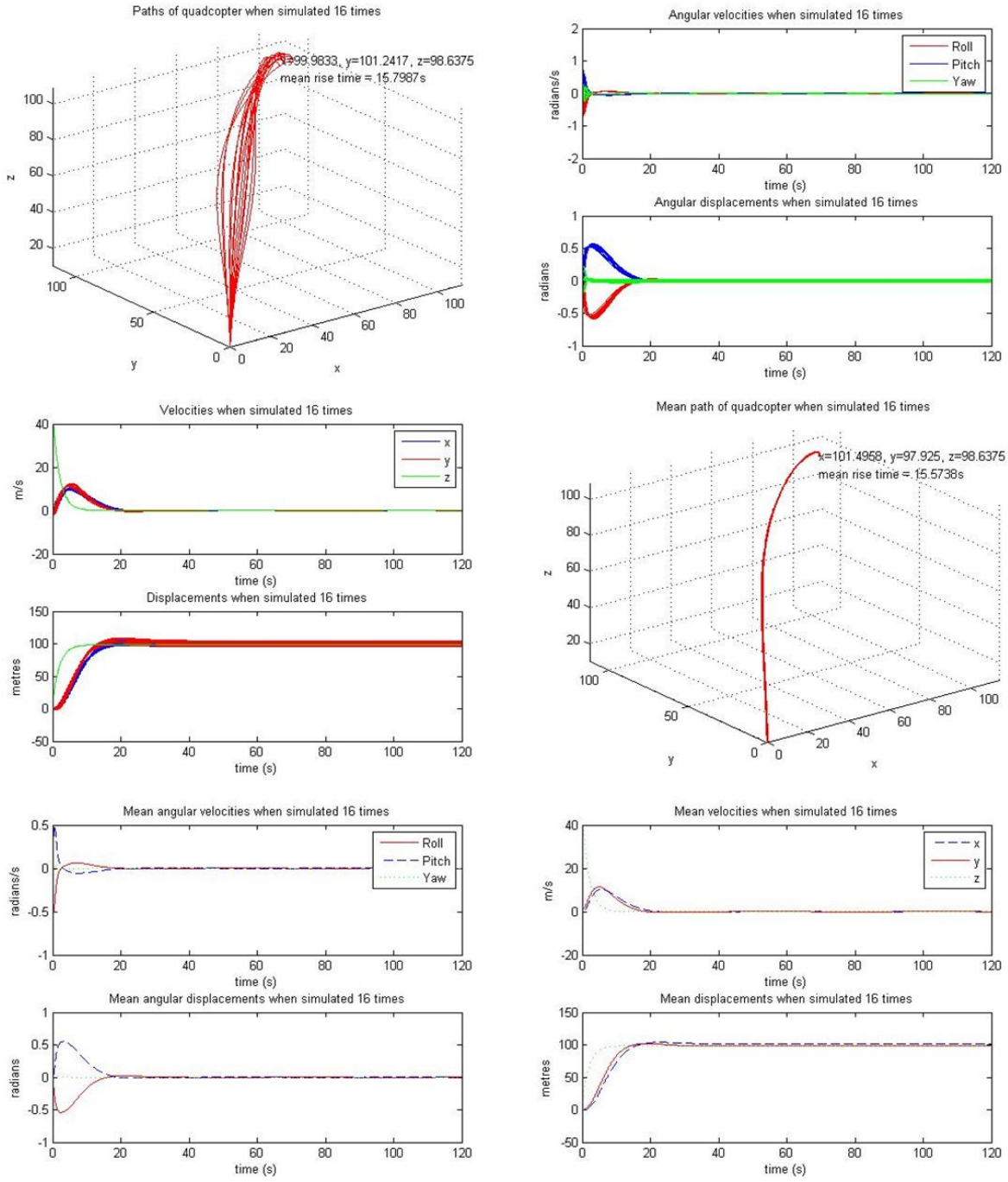


Figure 27: Results of 16 simulations

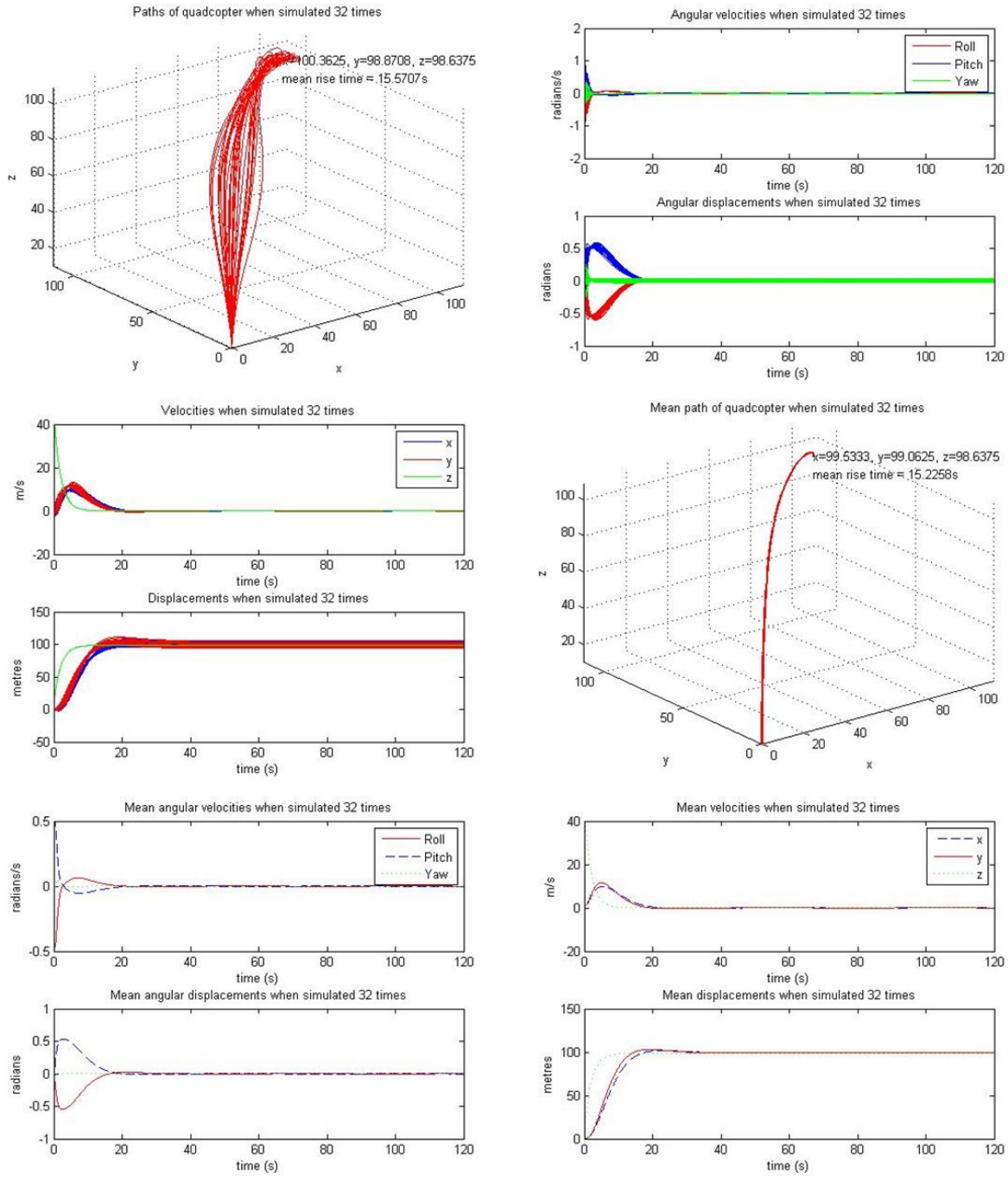


Figure 28: Results of 32 simulations

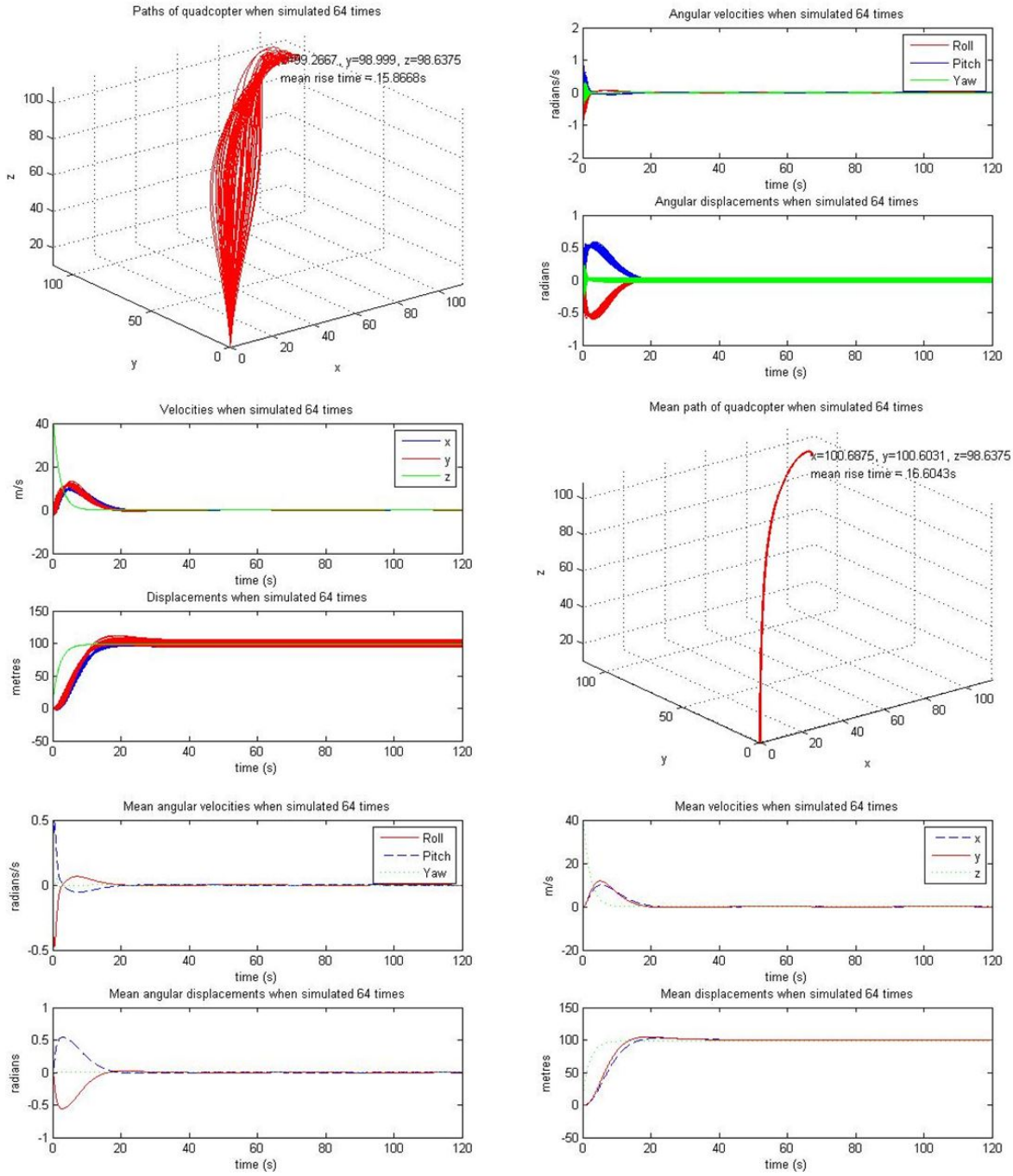


Figure 29: Results of 64 simulations

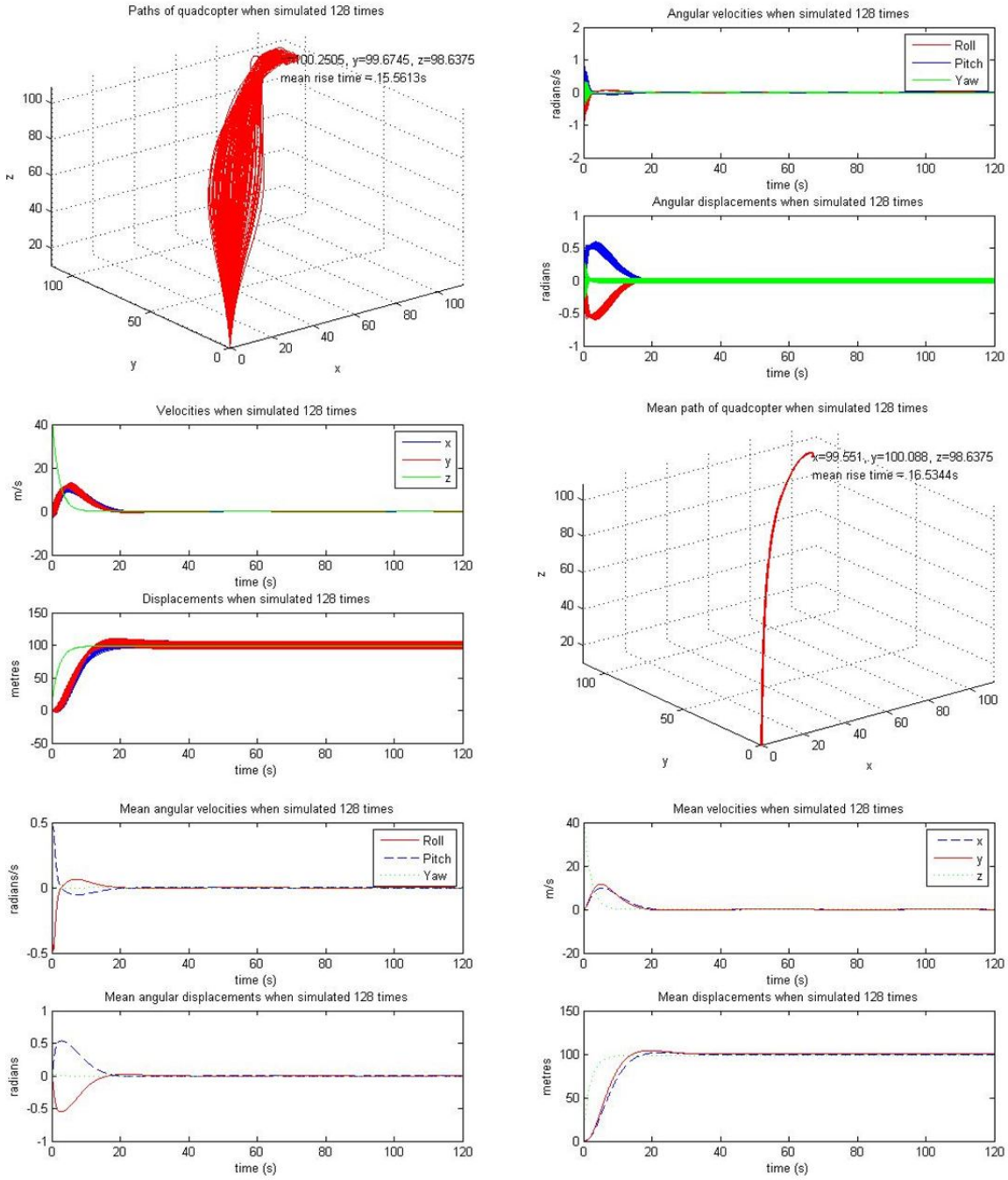


Figure 30: Results of 128 simulations

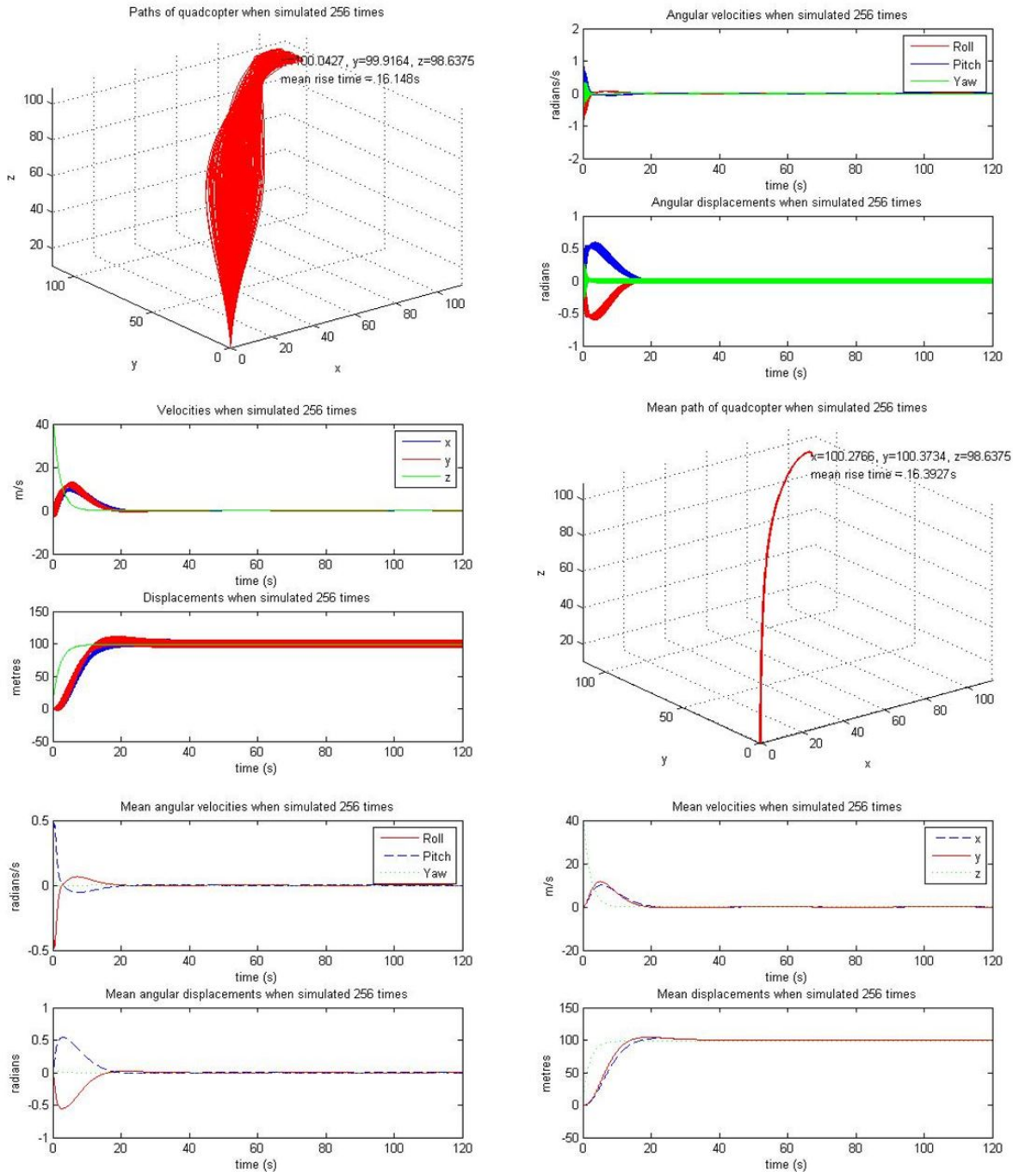


Figure 31: Results of 256 simulations

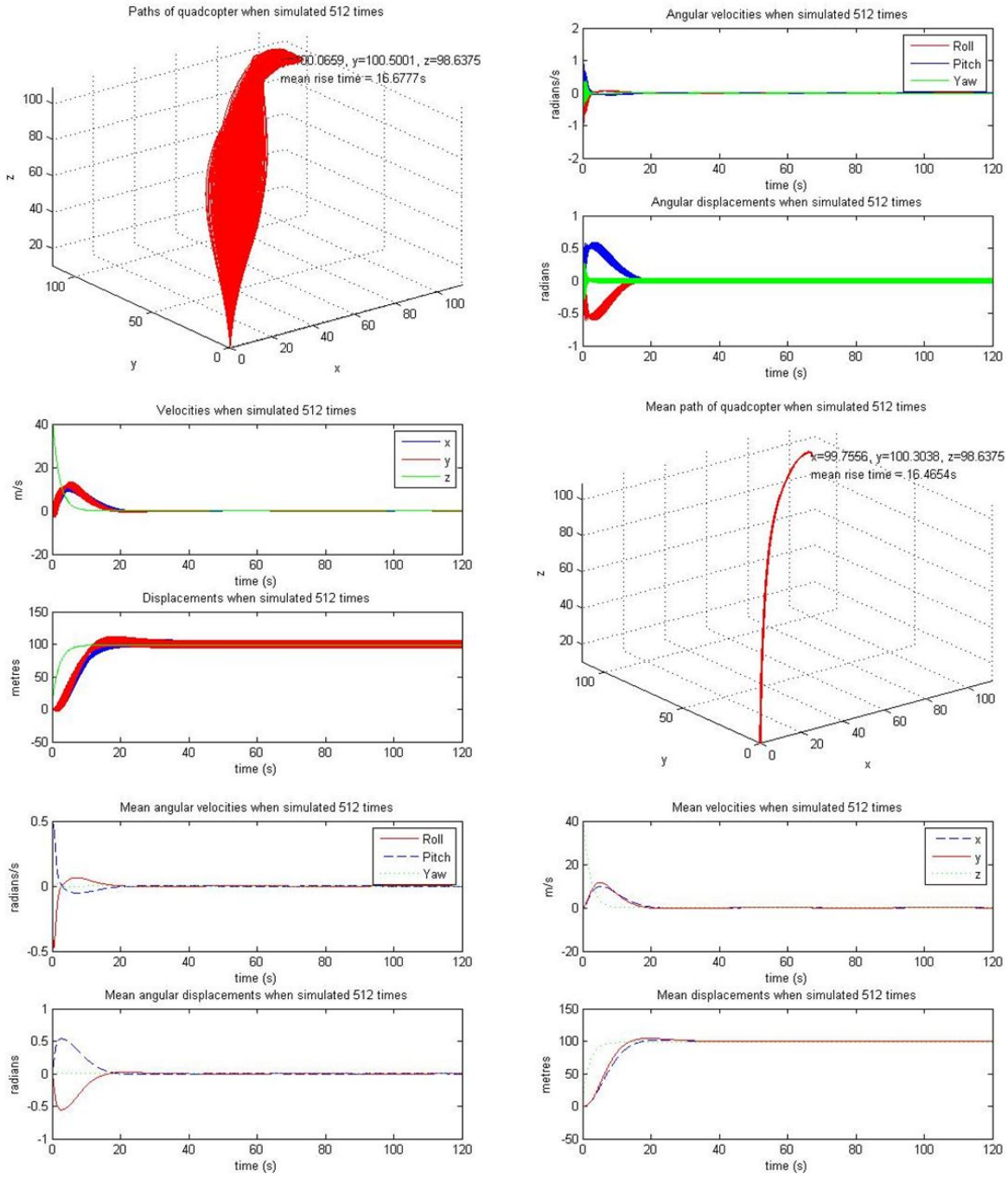


Figure 32: Results of 512 simulations

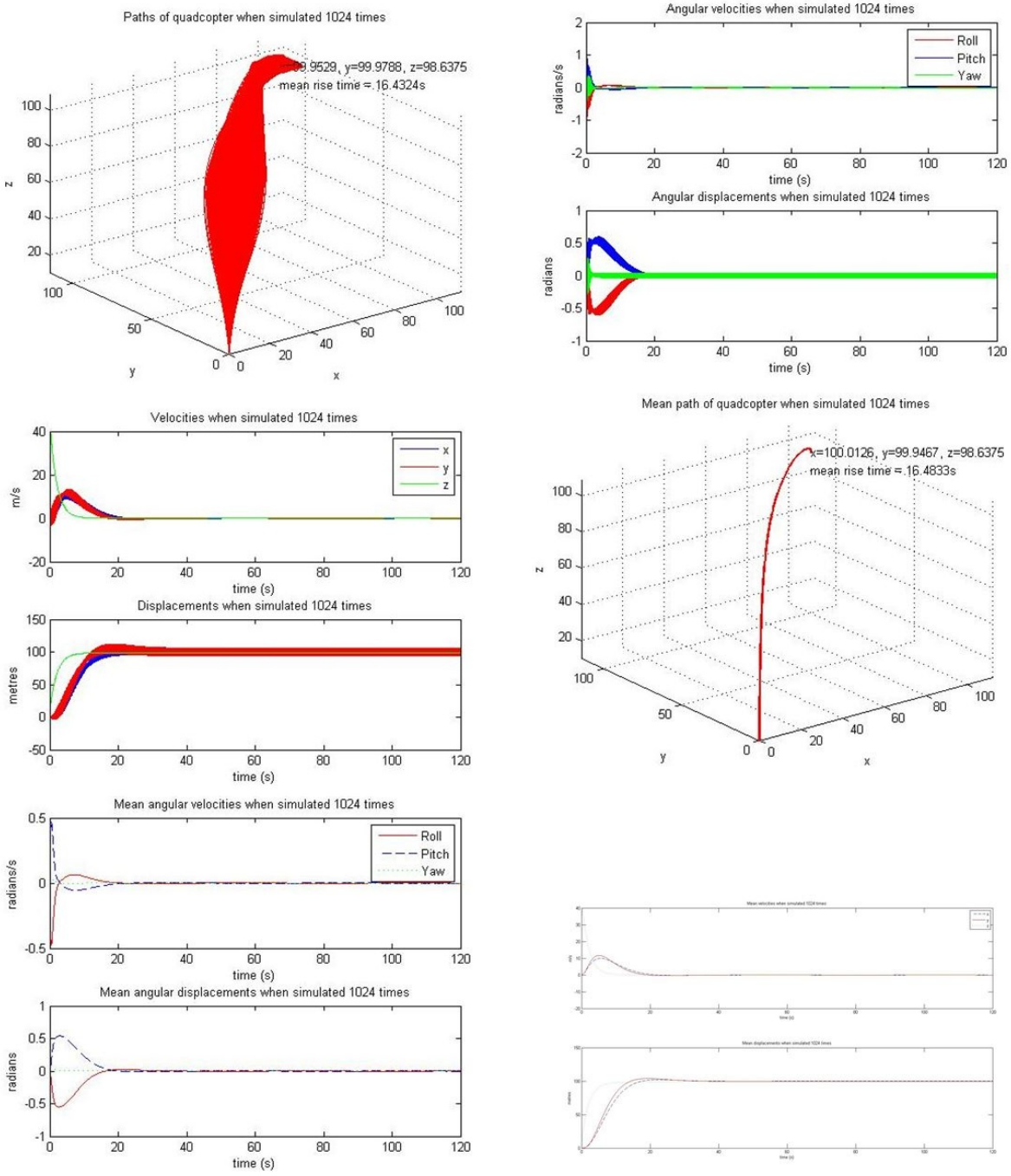


Figure 33: Results of 1024 simulations

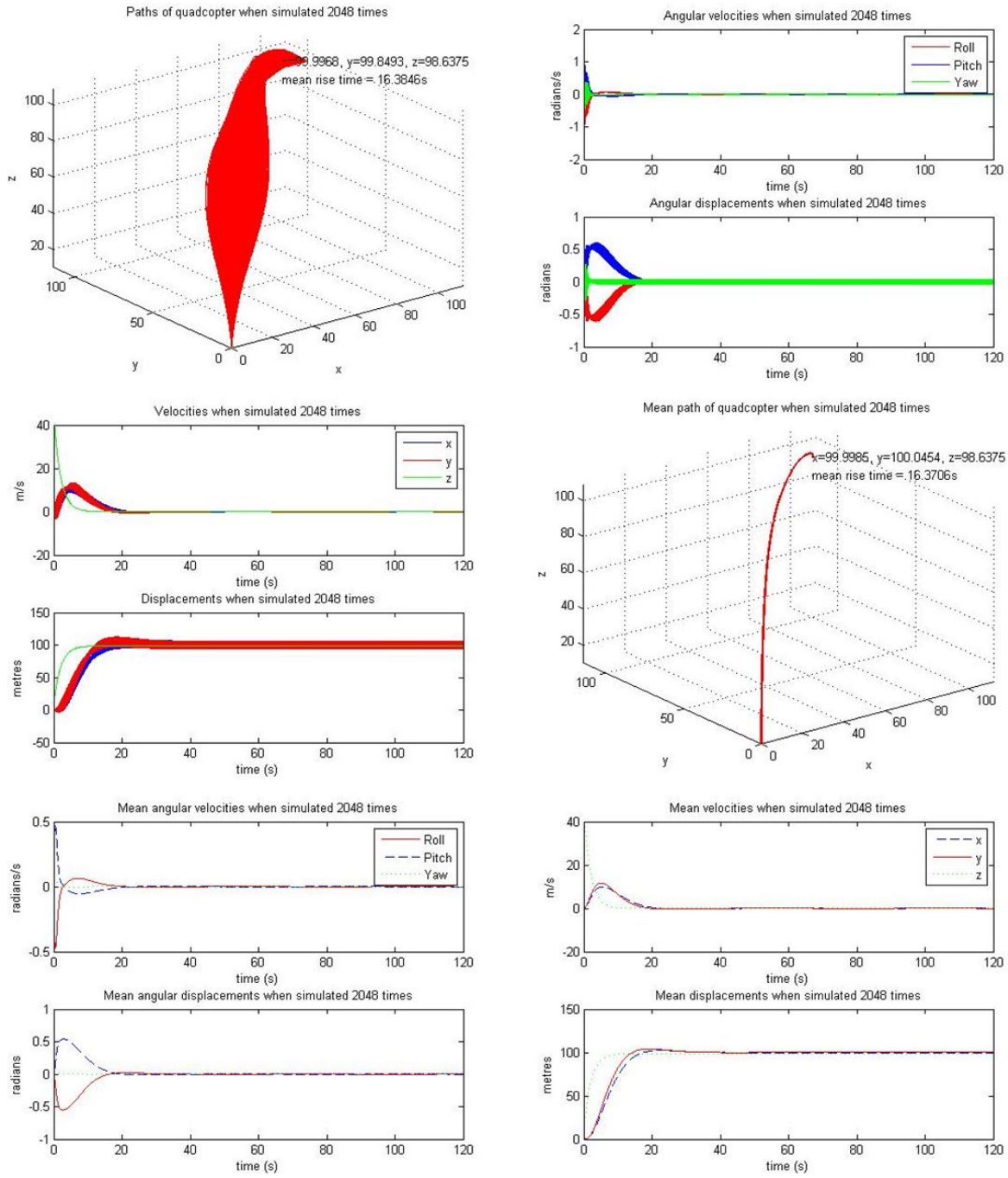


Figure 34: Results of 2048 simulations

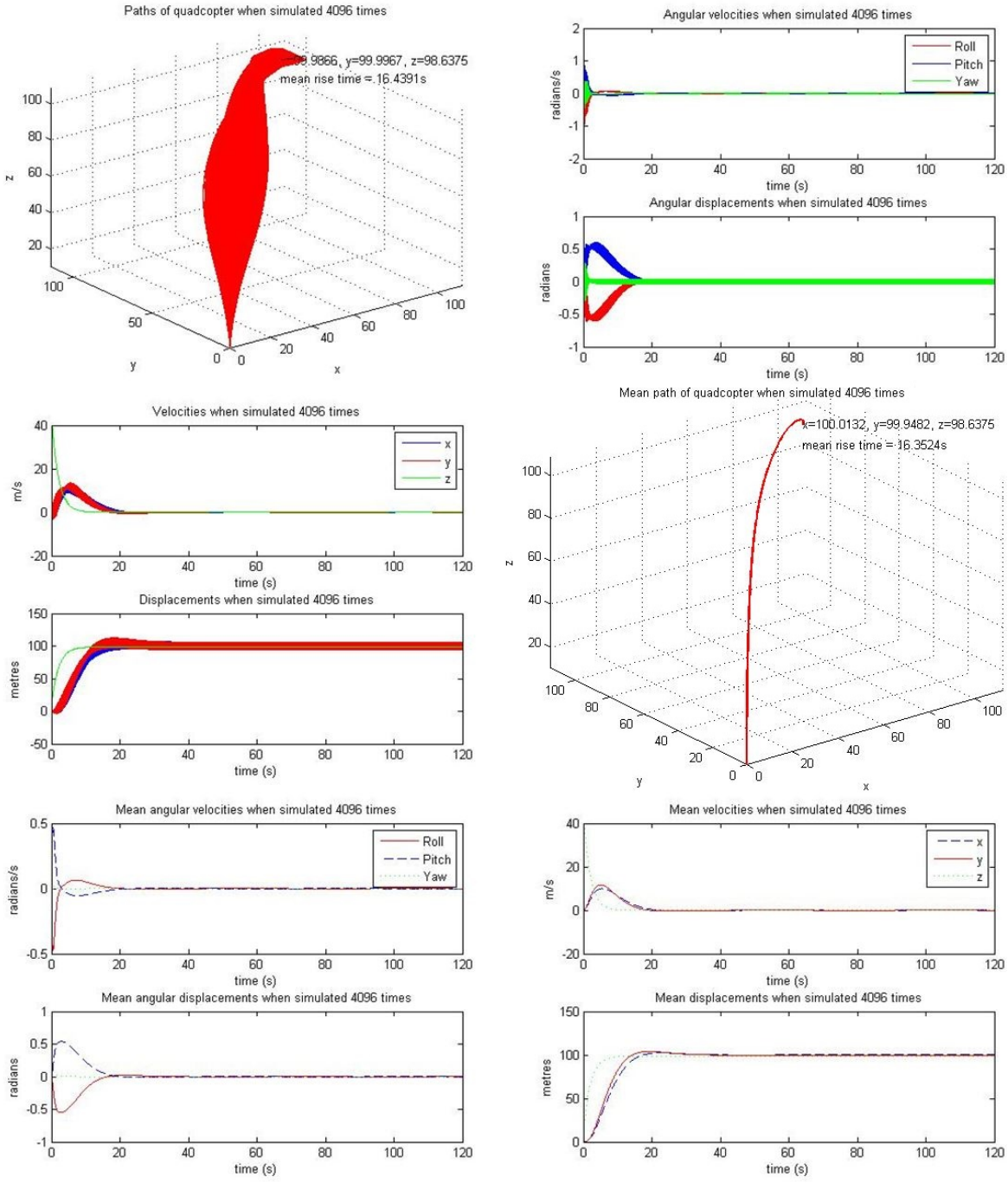


Figure 35: Results of 4096 simulations

22 Appendix G: Graphs of results from the PID controller

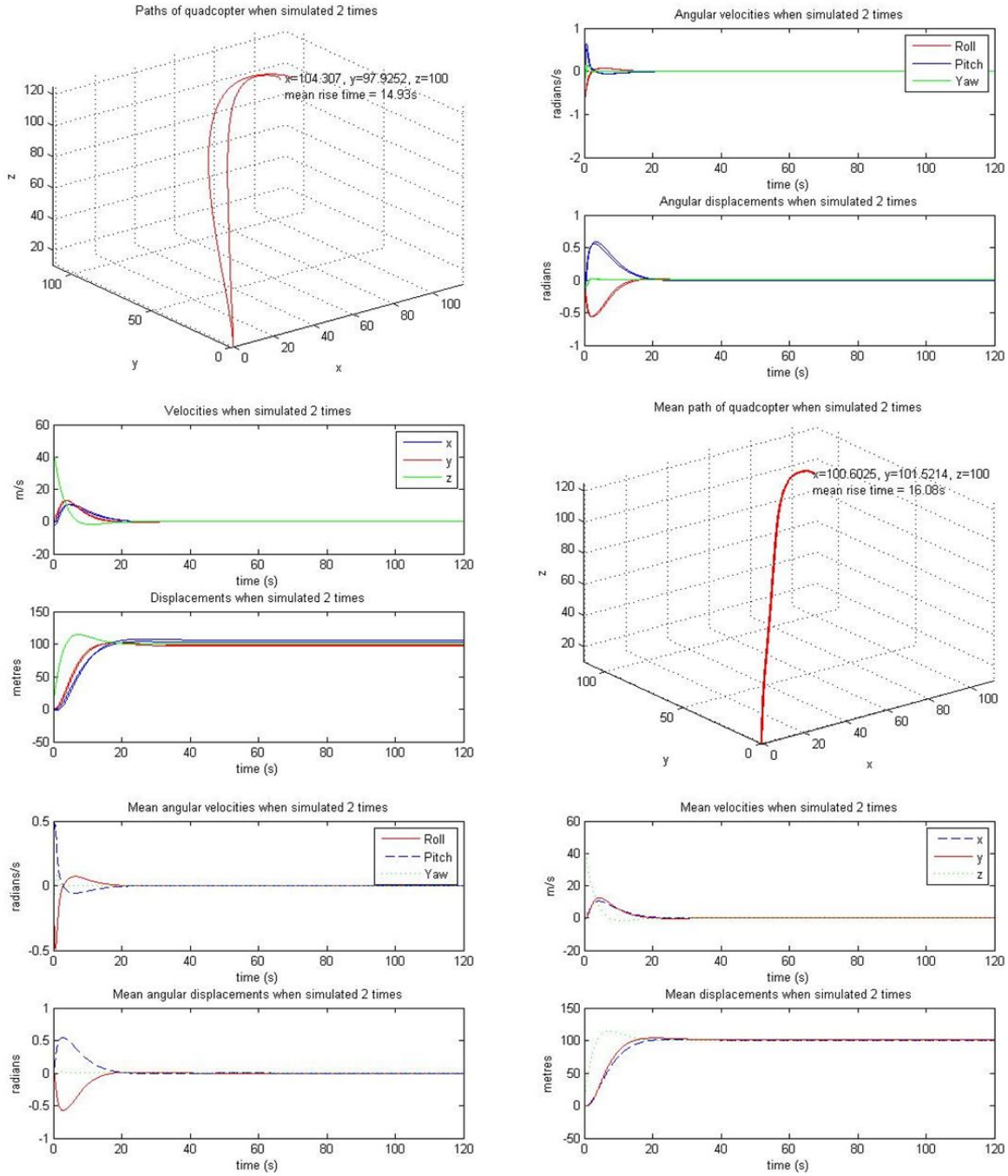


Figure 36: Results of 2 simulations

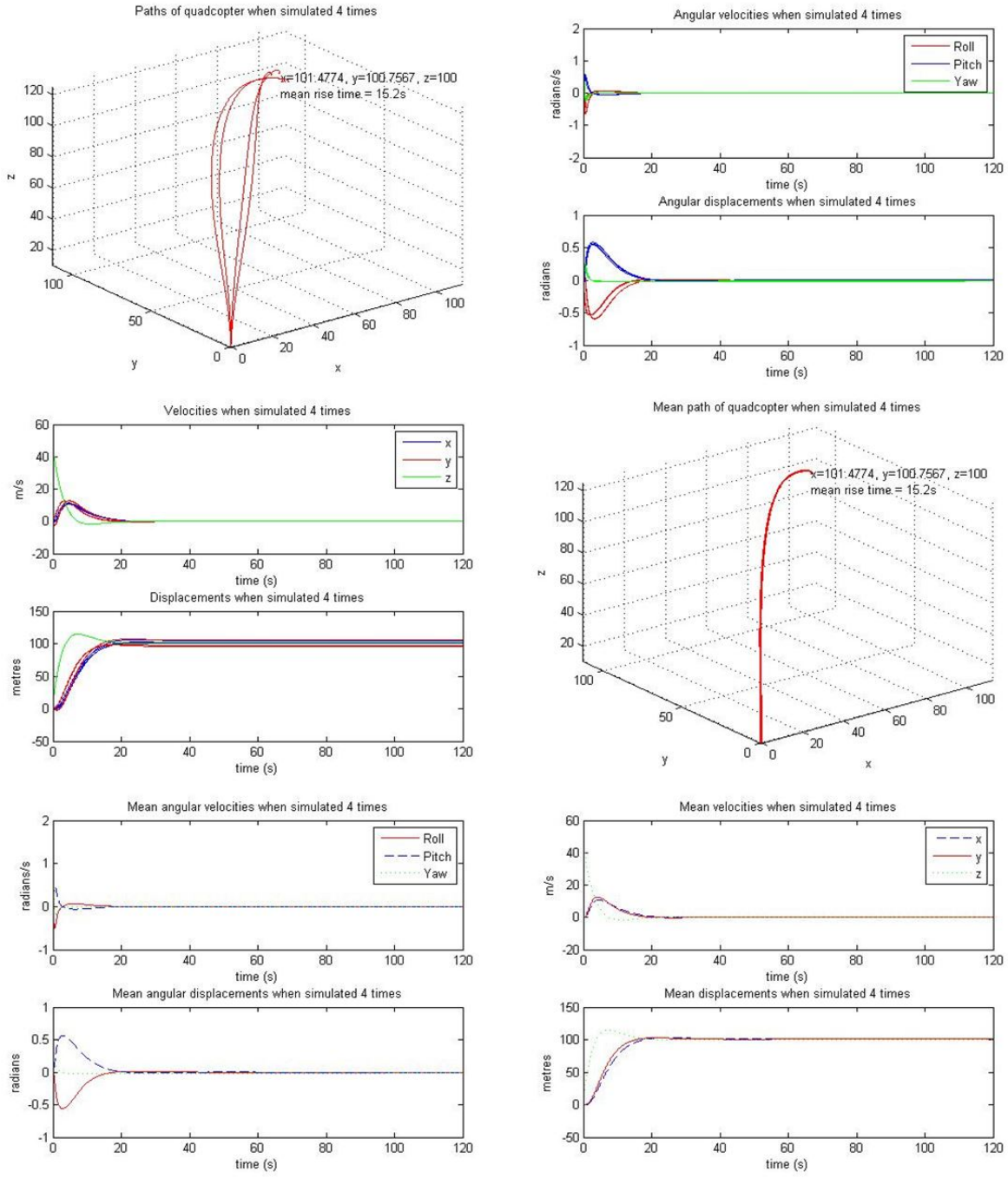


Figure 37: Results of 4 simulations

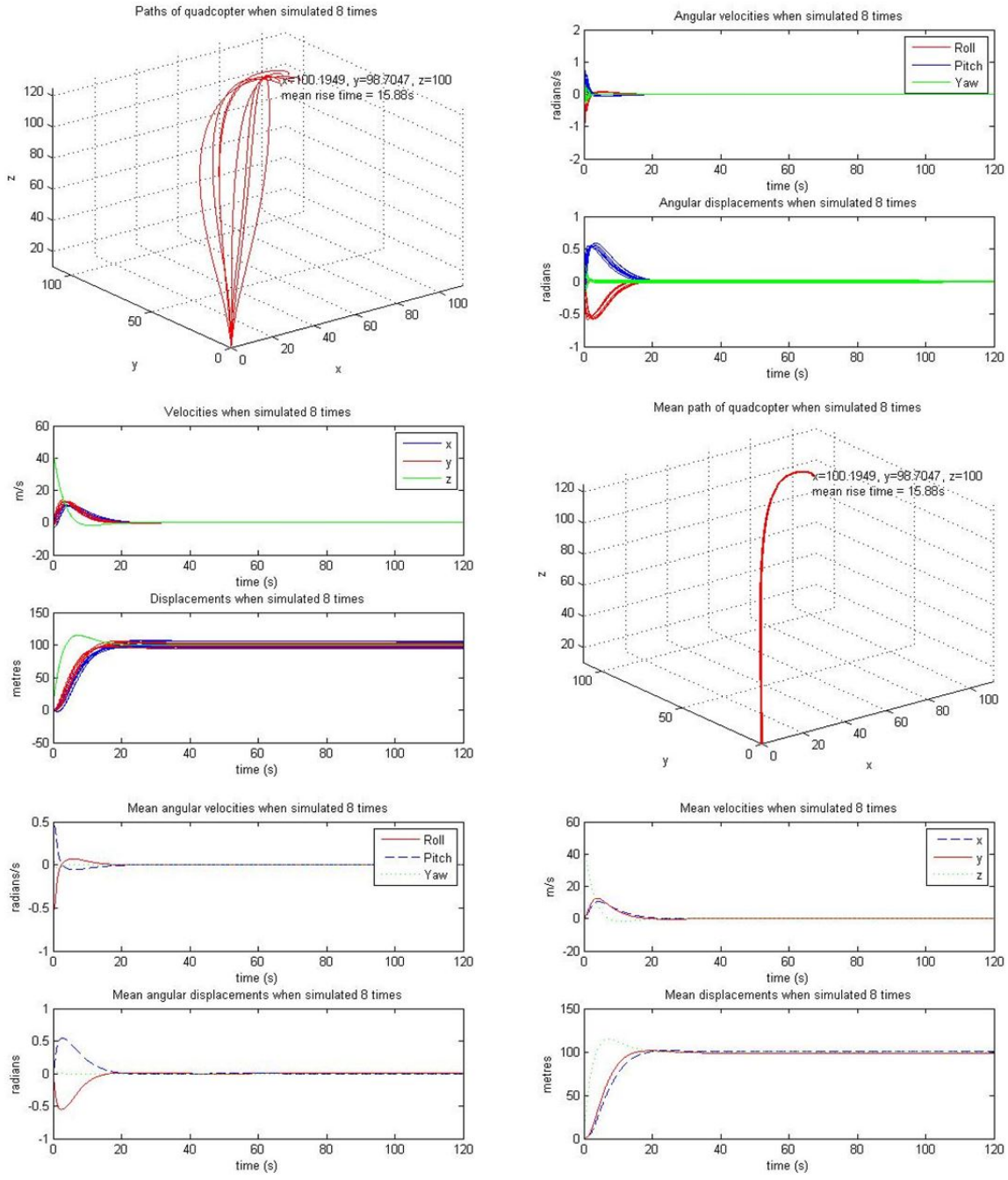


Figure 38: Results of 8 simulations

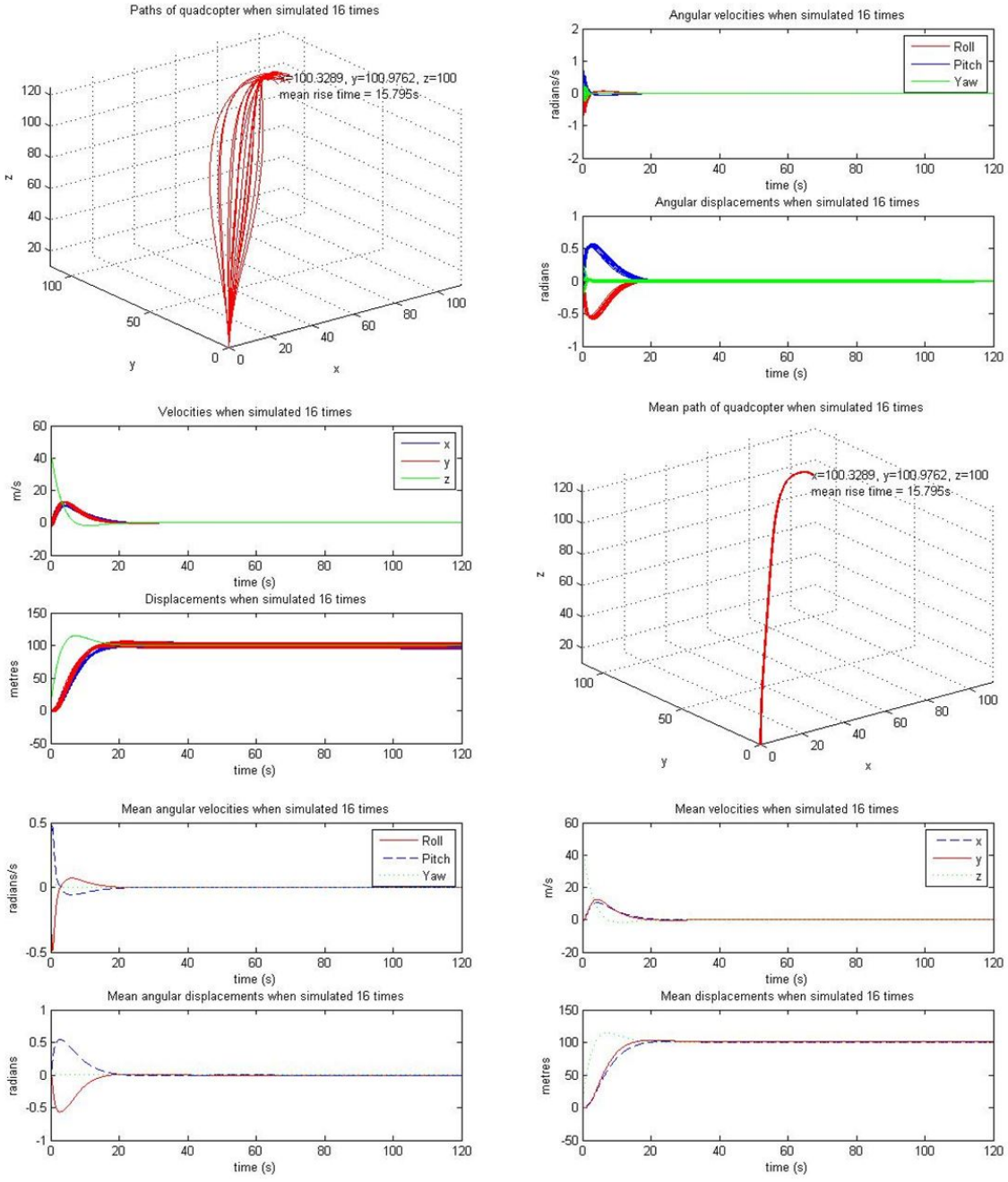


Figure 39: Results of 16 simulations

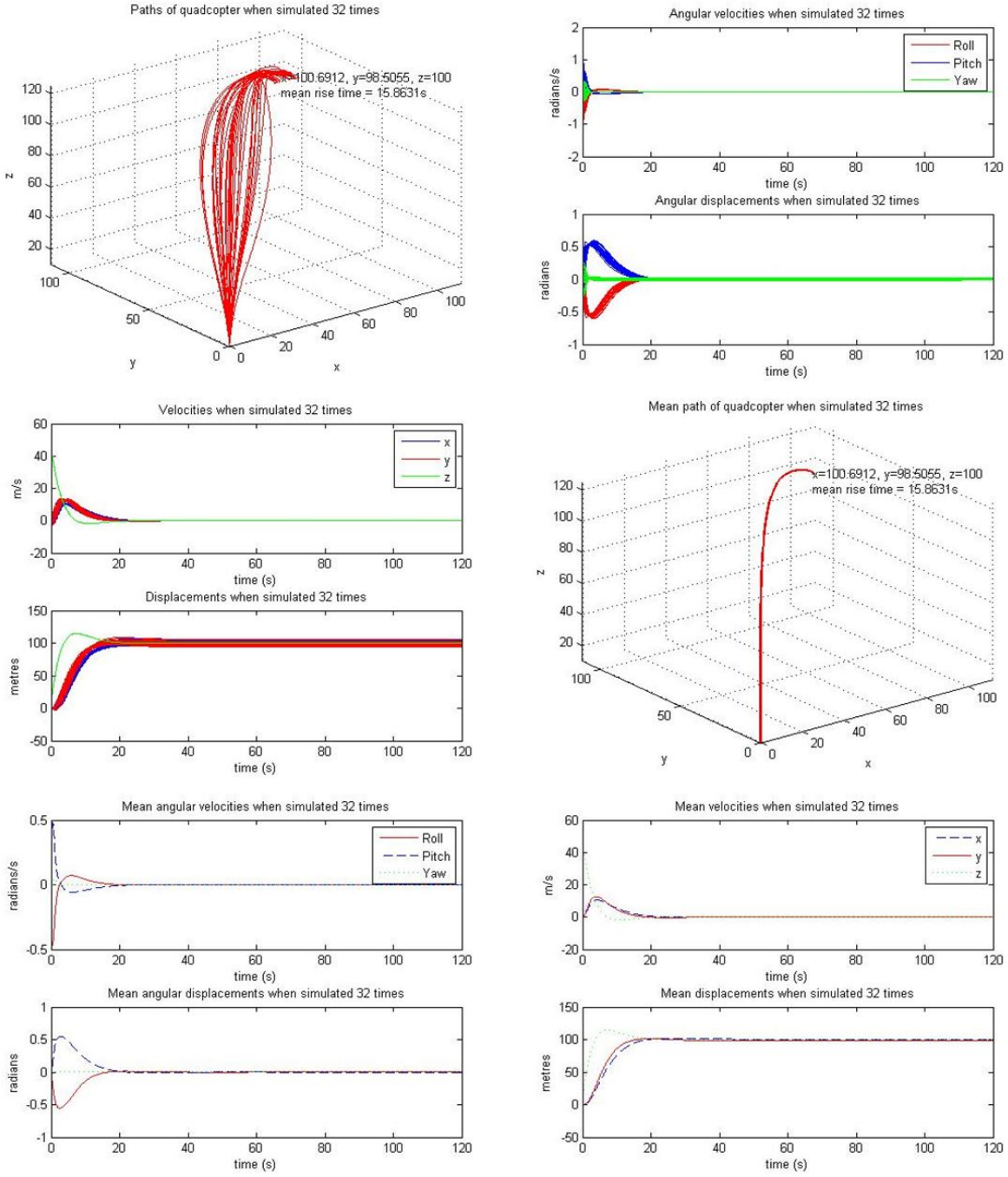


Figure 40: Results of 32 simulations

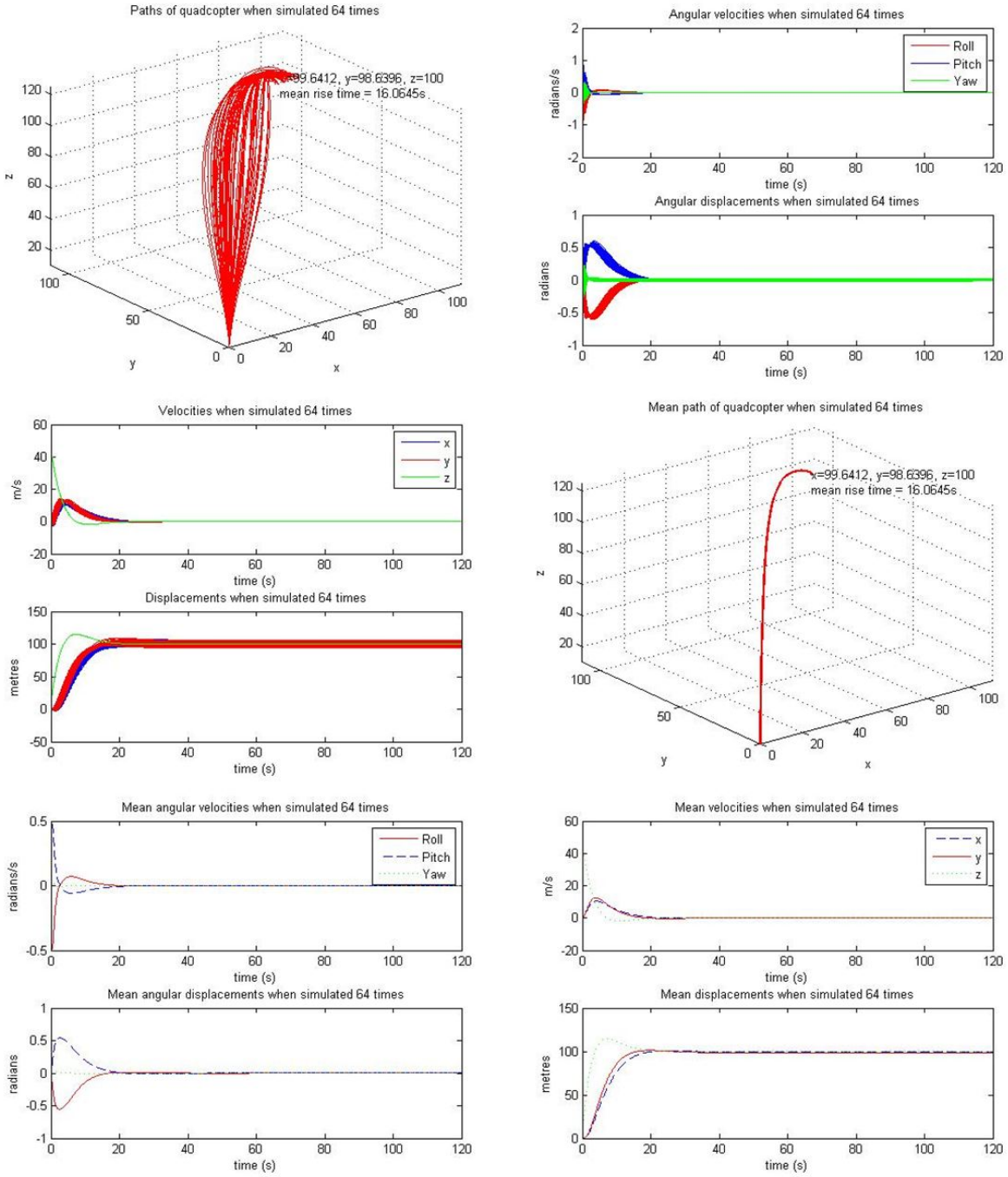


Figure 41: Results of 64 simulations

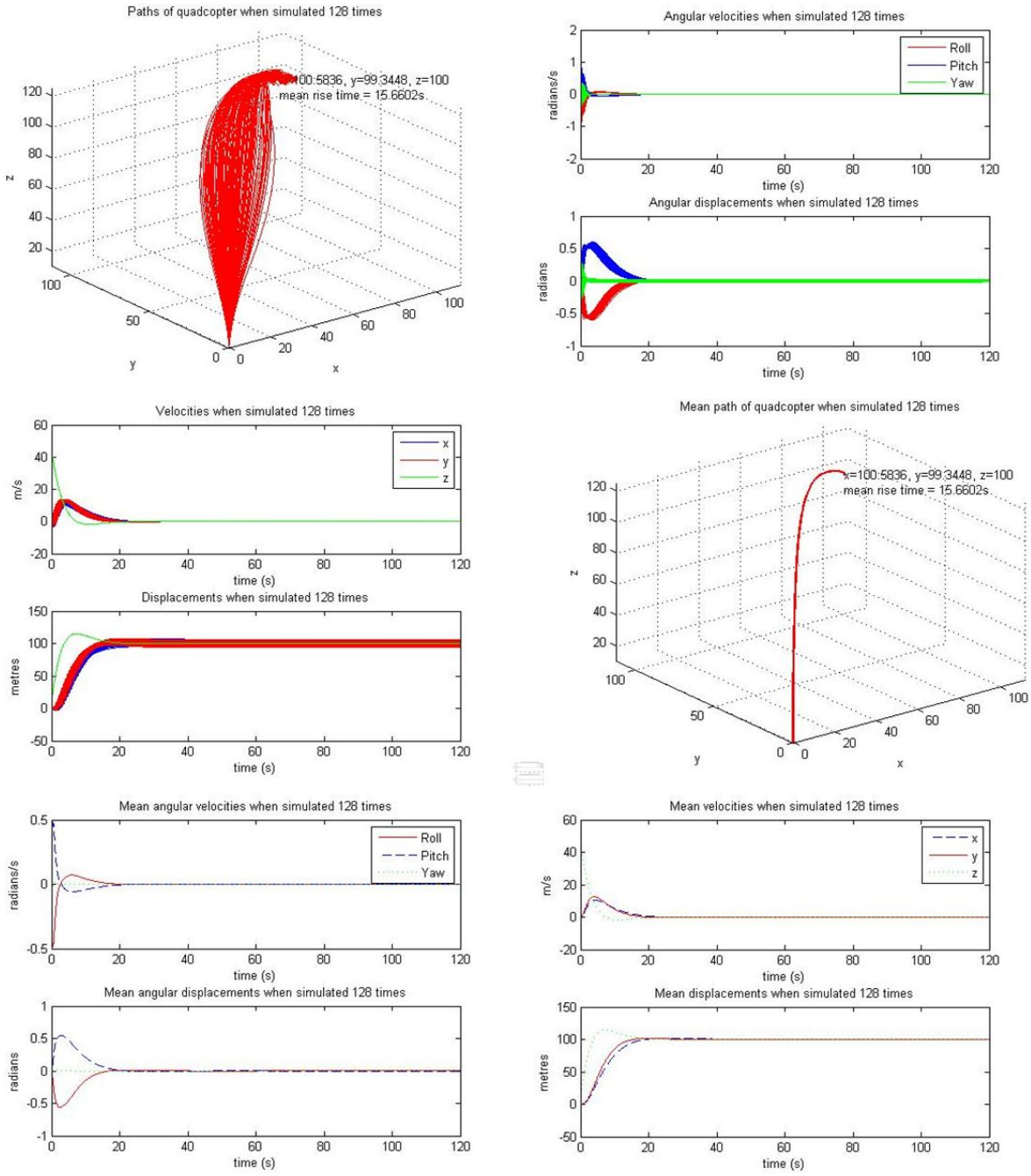


Figure 42: Results of 128 simulations

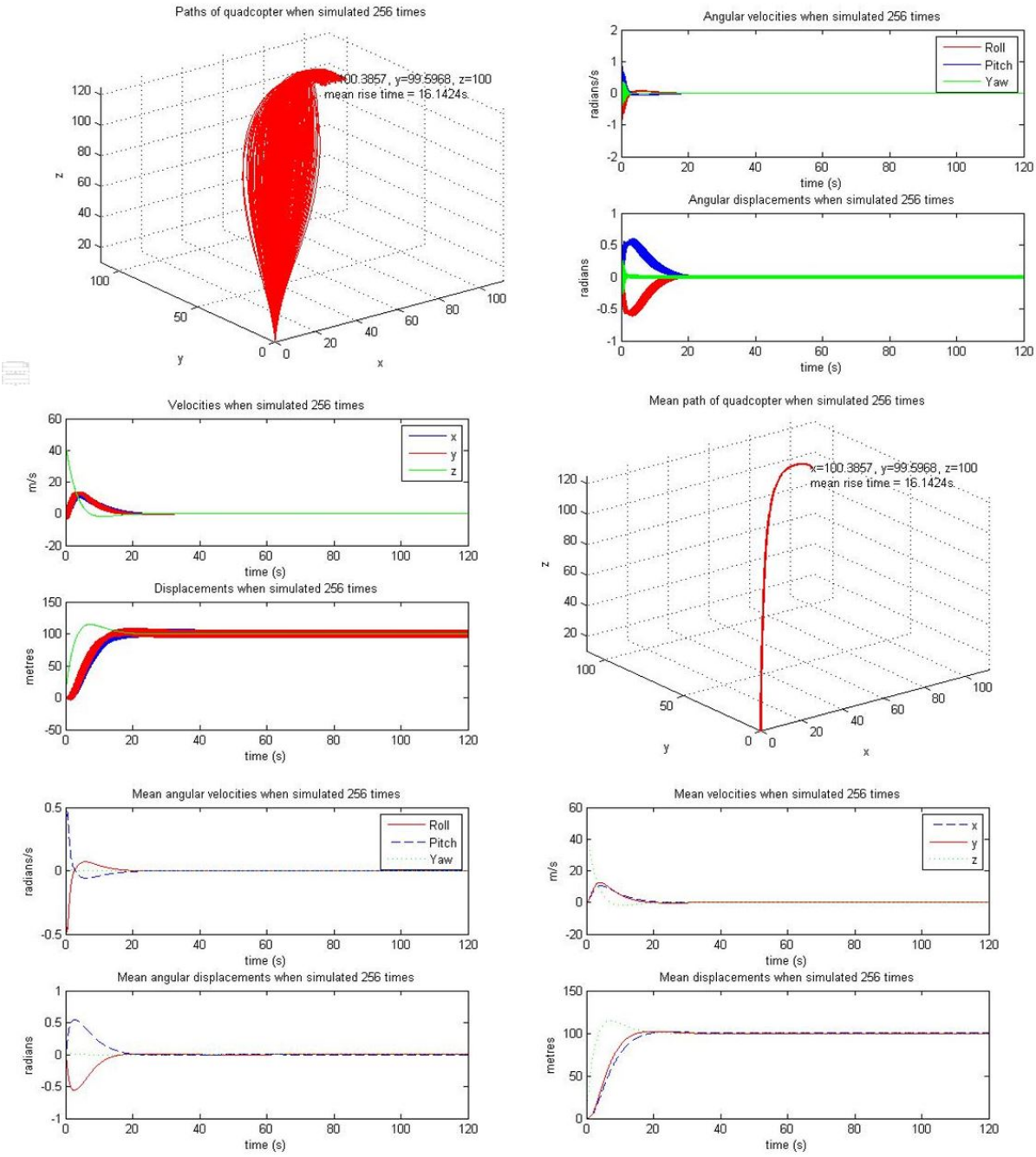


Figure 43: Results of 256 simulations

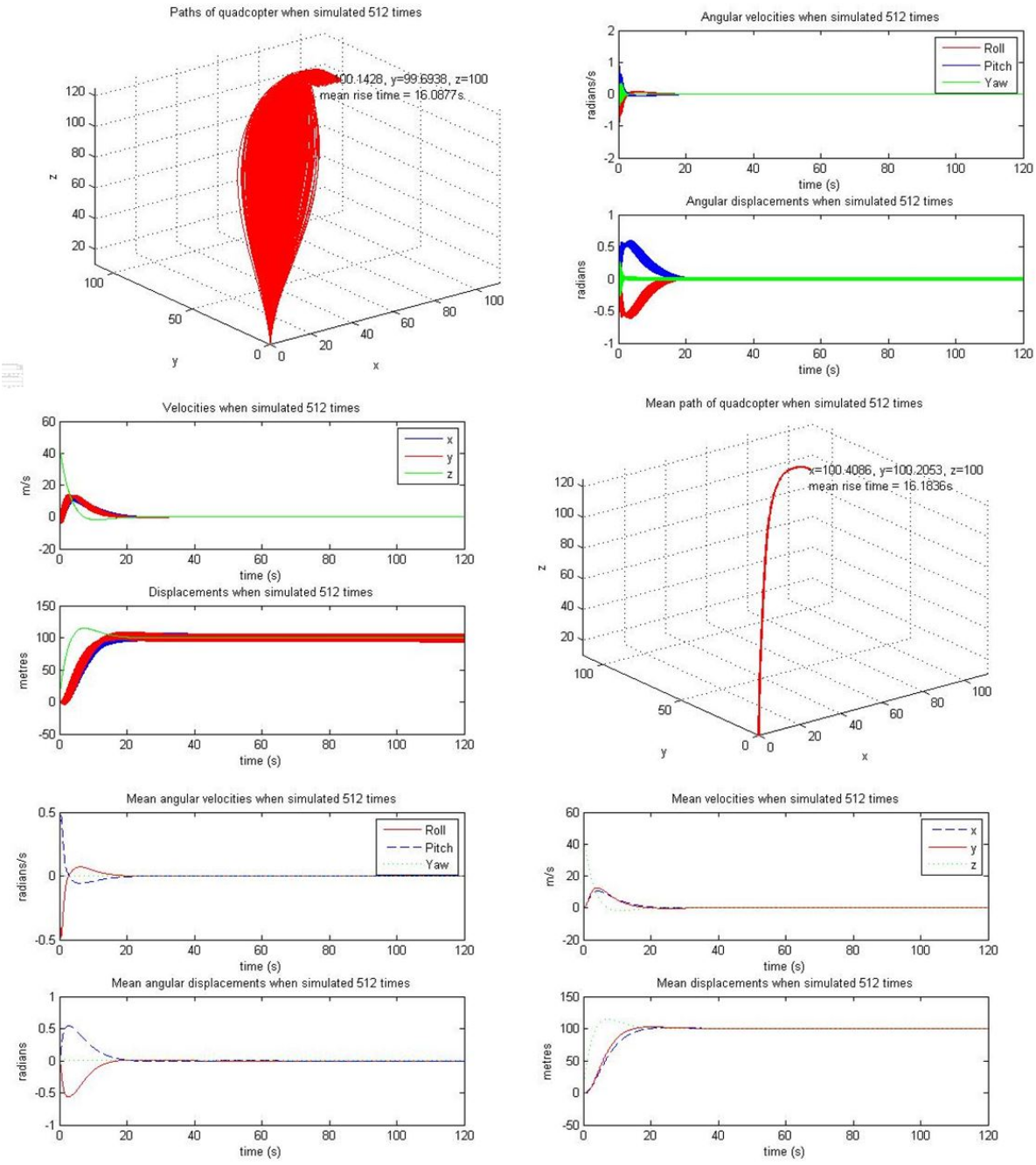


Figure 44: Results of 512 simulations

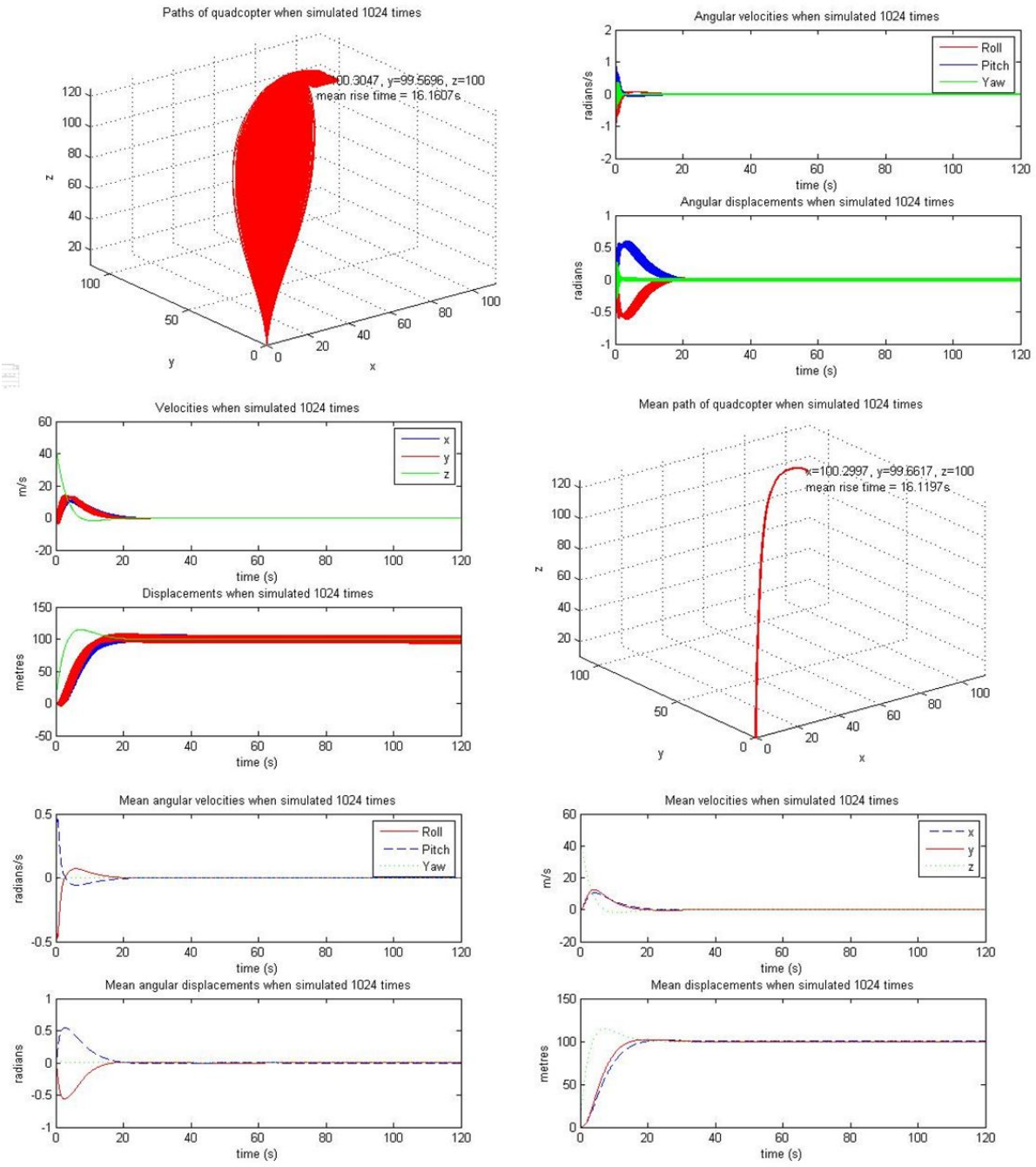


Figure 45: Results of 1024 simulations

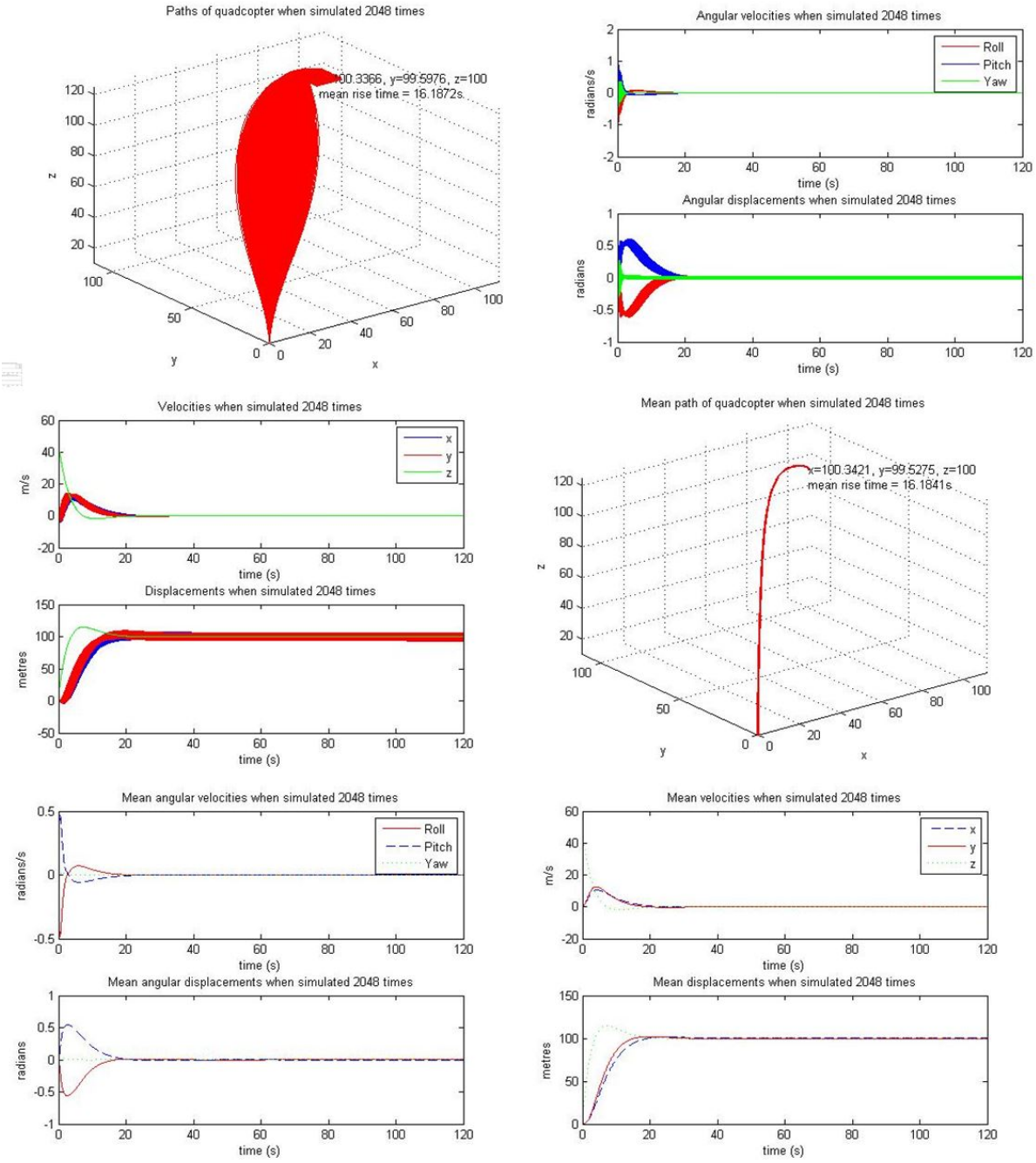


Figure 46: Results of 2048 simulations

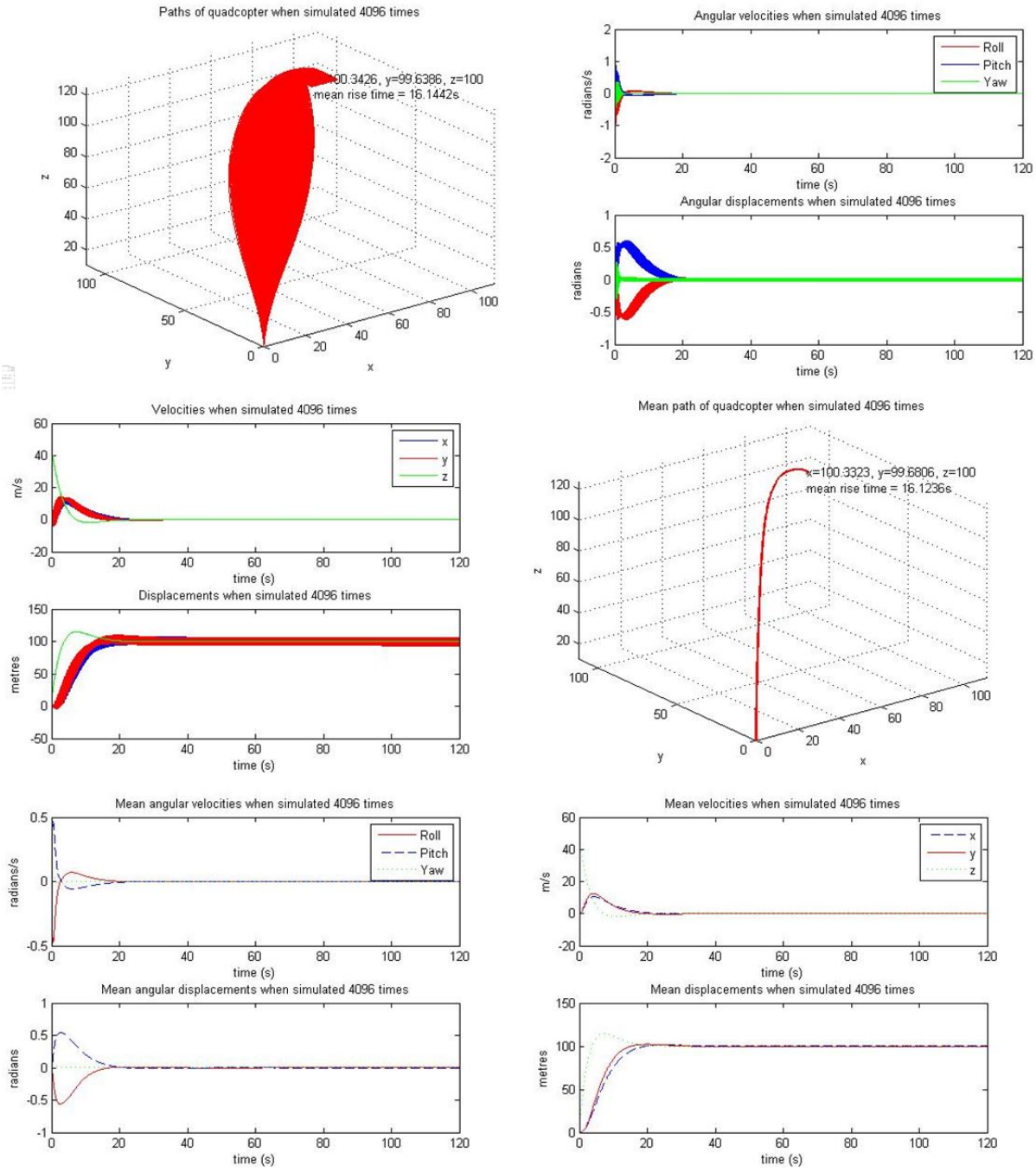


Figure 47: Results of 4096 simulations

23 References

[1] Tice, B. (1991) *Unmanned Aerial Vehicles: The Force Multiplier of the 1990s*, Airpower Journal, Spring Edition

[2] Cox, T., Nagy, C., Skoog, M., Somers, I. (2004) *Civil UAV Capability Assessment*, Dryden Flight Research Centre

[3] Raptis, A., Valvanis K. (2011) *Linear and Nonlinear Control of Small Scale Unmanned Helicopters*, 1st ed. Atlanta: Springer.

[4] Leishman, J. (2006) *Principles of Helicopter Aerodynamics*, 2nd ed. Cambridge: Cambridge University Press.

[5] Bouabdallah, S. (2007) *Design and Control of Quadrotors with Application to Autonomous Flying*, École Polytechnique Fédérale de Lausanne, Lausanne

[6] Etkin, B., Duff Reid, L. (1996) *Dynamics of Flight: Stability and Control*, 3rd ed. Toronto: John Wiley & Sons, Inc.

[7] Gibiansky, A. (2012). *Quadcopter Dynamics, Simulation, and Control.*
Available: http://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/. Last accessed 29th November 2013

[8] Bouabdallah, S., Noth, A., Siegwart R. (2004) *PID vs LQ Control Techniques Applied to an Indoor Micro Quadrotor*

[9] Bennett, S. (1986) *A History of Control Engineering*, 1st ed. London: Peter Peregrinus Ltd.

[10] Schwarzenbach, J., Gill, K. (1992) *System Modelling and Control*, 3rd ed. Leeds: Edward Arnold

[11] Raffo, G., Ortega, M., Rubio, F. (2009) *An Integral Predictive/Nonlinear H_∞ Control Structure for a Quadrotor Helicopter*, Universidad de Sevilla, Sevilla.

[12] Shaikh, M. (2011) *Quadrocopter Fuzzy Flight Controller*, Örebro University, Närke, Sweden.

[13] Abassi, E., Mahjoob, M., Yazdanpanah. (2013) *Controlling of Quadrotor UAV Using a Fuzzy System for Tuning the PID Gains in Hovering Mode*, University of Tehran, Tehran.

[14] al-Omari, M., Jaradat, M., Jarrah, M. (2013) *Integrated Simulation Platform for Indoor Quadrotor Applications*, University of Sharjah, Sharjah, UAE.