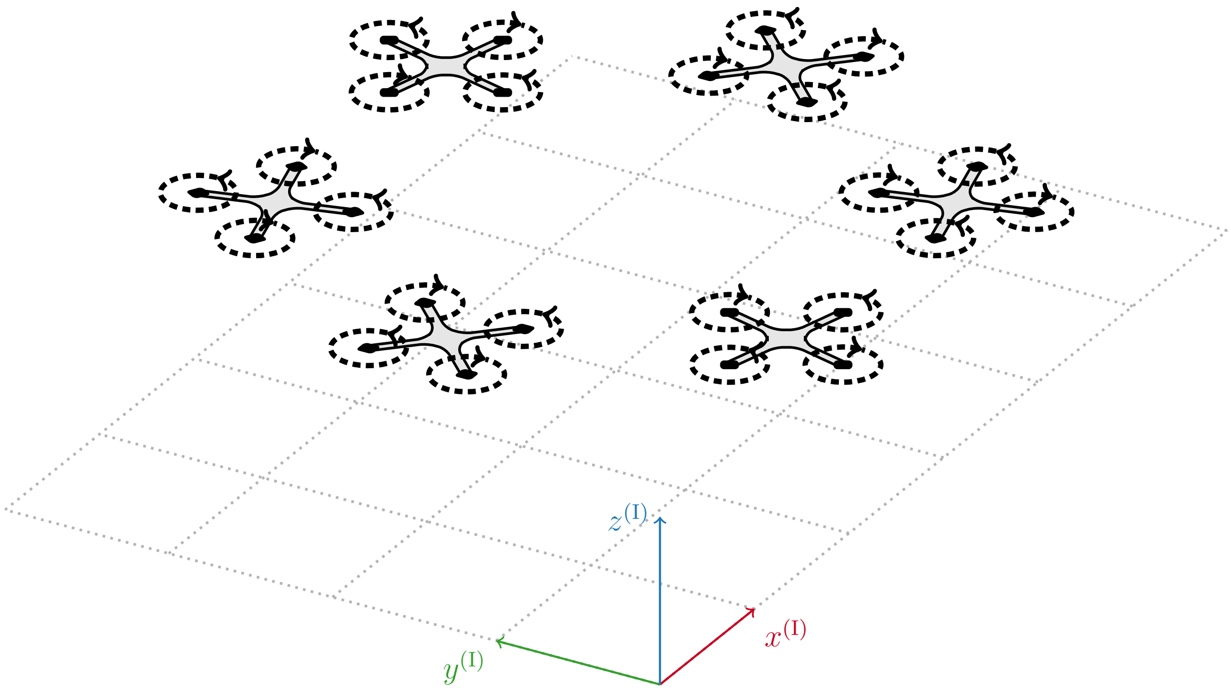
N-rotor vehicles: modelling, control, and estimation

Automatic Control Laboratory (IfA)
Eidgenössische Technische Hochschule (ETH) Zürich



Paul N. Beuchat

February 22, 2023

Abstract

Quad-rotors are becoming more and more commonplace as technological advancements increase capabilities and reduce costs. Quad-rotor vehicles are encountered both for domestic entertainment and industrial applications, some examples are: toys for people of all ages, professional cinematography, or, inspection of industrial scale structures and processes. One reason why quad-rotors have become so pervasive is their mechanical simplicity, which lends itself to the high operational reliability. Moreover, the control and estimation techniques required to stabilise a quad-rotor around hover require only the control theory taught to under-graduates, while acrobatic feats and fleet manoeuvres inspire many directions in current research.

The learning objective of this course are two-fold: (i) the students will gain knowledge about and understanding of the modelling and control theory for a quad-rotor application, and (ii) the students will make the connection to and apply theory taught in the under-graduate control system classes. By implementing the control and estimation algorithms on a real quad-rotor, the students will gain experience with how decisions in the modelling and design stage affect real-world performance.

Contents

Nomenclature

Indices

$(\cdot)'$ or $(\cdot)^T$	Vector or matrix transpose
I_n	Identity matrix of dimension n
$\mathbf{1}_{n\times m}$	Matrix of ones of dimension $n \times m$
s_θ, c_θ	short hand for the trigonometric functions $\sin(\theta)$ and $\cos(\theta)$

Symbols

I	the inertial frame
B	the body frame
$(\cdot)^{(\mathrm{I})}$, $(\cdot)^{(\mathrm{B})}$	indicating $(\cdot)$ is expressed in coordinates of the I or B frame
x, y, z	the Cartesian coordinates directions, following "right-hand" convention
$\vec{(\cdot)}$	indicating $(\cdot)$ is a vector
$\vec{p}$	position of body frame origin relative to inertial frame origin
p_x, p_y, p_z	the $x^{(\mathrm{I})}$, $y^{(\mathrm{I})}$, and $z^{(\mathrm{I})}$ component of $\vec{p}$ respectively
$\vec{\omega}$	angular velocity of the body frame, as measured in the body frame, (referred to as the *body rates*)
$\omega_x, \omega_y, \omega_z$	the $x^{(\mathrm{B})}$, $y^{(\mathrm{B})}$, and $z^{(\mathrm{B})}$ component of $\vec{\omega}$ respectively
N	The number of propeller on the vehicle
f_i	The thrust force produced by propeller i
$\vec{f}_{\text{thrust}}$	The total thrust force vector acting on the vehicle
$\vec{f}_{\text{gravity}}$	The gravity force vector acting on the vehicle
$\vec{f}_{\text{body drag}}$	The aerodynamic drag force on the body of the vehicle
m	mass of the vehicle
J	moment of inertia of the vehicle about the body frame axes, this is may also be referred to as angular mass or rotational inertia (in German it is named Trägheitsmoment, Massenträgheitsmoment, or Inertialmoment)

Abbreviations

PID	Proportional, Integral, Derivative
LQR	Linear Quadratic Regulator
GUI	Graphical User Interface
CoG	Center of Gravity

Preface

My motivation for creating this script first arose in early 2015 when I embarked on the endeavour to set up an autonomous flying arena in the Automatic Control Laboratory (IfA) at ETH Zürich. At the time I naively believed that it would take just one semester of work with a few students to assist... three years later and we achieved my original goal, and the journey has taught me many things. It seemed to me in 2015 that autonomous flight for quad-rotors was a solved problem, countless research groups around the globe had autonomous flying arenas, most using a camera-based motion capture system. Conveniently we had access to such a motion capture system at IfA, but it seemed that the software and documentation required to combine that with an off-the-shelf quad-rotor was not readily available or open source. The details of how we arrived at the current status are tedious, but suffice to say the journey required a few restarts and advice from many different people.

As part of setting up our autonomous flying arena I wanted to make it useful to others beyond our research group. My conclusion was that the hardware and software development should be accompanied by a concise body of material that contains the necessary control theory for any bachelor level student to understand and build an autonomous flying arena from scratch. Thus was born the Distributed Flying and Localisation Laboratory (or D-FaLL for short) as an umbrella for the whole endeavour, and the beginnings for this script.

Paul N. Beuchat.

Chapter 1

Introduction

There are many reasons to study, utilise, and perform research using quad-rotors. To list a handful of applications where the utilisation of quad-rotors enables practices and techniques not possible before: photography, package delivery, supervision of events, constructing 3D maps of large structures, agriculture, search and rescue missions, art performances. For research, quad-rotors can be used as a platform to demonstrate new methods from a range of fields, or the quad-rotor itself can be used as a dynamical system to motivate novel research. And finally, let's face it, quad-rotors are awe-inspiring to watch and fun to play with :-)

The main reasons that a quad-rotor has been chosen as the focus of this course are:

i) achieving stable hover of a flying vehicle has the inherent challenge of fast dynamics
ii) the modelling, control, and estimation techniques required to achieve stable hover are accessible to and instructive for the typical under-graduates control theory curriculum.
iii) availability of small, cheap quad-rotors with open source software and hardware design.

However, to understand why the symmetry of a quad-rotor vehicle simplifies the synthesis of control and estimation algorithms, we consider in the modelling phase of the course a generalisation of a quad-rotor vehicle that we refer to throughout the script as an *N-rotor vehicle*.

An N-rotor vehicle, like a quad-rotor, has its N propellers attached to a rigid body, all mounted in the same plane, and aligned so that all the thrust vectors point normal to the plane in which the propellers are mounted. The key generalisation we allow by considering N-rotor vehicles in the modelling phase is that the arrangement of the rotors on the vehicle is a design choice. By analysing the equations of motion of this general N-rotor vehicle we aim to provide insight into why the quad-rotor arrangement allows stable hover to be achieved with basic control and estimation techniques.

The remainder of this chapter covers the learning objective and curriculum of the course. Summarising briefly, in the first half of the course we introduce the physical model for an N-rotor vehicle and use this to derive control and estimation techniques that are taught in the 5th semester in the Control System 1 class. The students then create their own control algorithms for a quad-rotor and test these in simulation. In the second half of the course the students implement the control and estimation algorithms they designed on our fleet of nano-quad-rotors. Once stable hover is achieved, the students will have the freedom to perform tasks with the quad-rotors, or control multiple quad-rotors in collaboration.

1.1 Learning Objectives

A student who successfully completes this course is expected to have achieved the following learning objectives:

i) Be able to derive the continuous-time equations of motion for a general N-rotor vehicle,
ii) Be able to simulate a general N-rotor vehicle for the purpose of testing and tuning controller and estimator designs,
iii) Be able to explain how flight performance is affected by changes in the parameters of the N-rotor vehicle, for example mass, centre of gravity location, propeller aerodynamics,

iv) Be able to explain why a quad-rotor design allows the control architecture to be de-coupled into a collection of separate simple controllers,

v) Experienced the challenges of tuning a PID and/or LQR controller for achieving stable hover of a quad-rotor, both in simulation and on the real-world system,

vi) Be able to write C++ code for implementing a PID and/or LQR controller.

1.2 Curriculum

The course will be conducted over six sessions, each session expected to take 3-4 hours. The first three sessions will be dedicated to the theory and simulation of N-rotor vehicle modelling and controller design, and the last three sessions are dedicated to experimental implementation.

Session 1: Modelling and equations of motion

- Introduction, motivation, and learning objectives for the course,
- Rotation matrices, motivated as being needed for the equations of motion,
- Non-linear, continuous-time, equations of motion for an N-rotor vehicle.

Session 2: Simulation of the equations of motion

- Pre-work: introduction to Simulink tutorial,
- The students are given a template Simulink model and should implement the equations of motion for an N-rotor vehicle. A stabilising controller will be provided for testing.
- Review of linearisation theory.

Session 3: Simulation and tuning of controller

- Pre-work: derive the linearisation of the equations of motion of an N-rotor vehicle,
- Review PID/LQR theory and discuss motivation for using a decoupled and cascaded structure for controlling an N-rotor vehicle,
- Students implement their PID/LQR controller in the same Simulink model developed for the equations of motion,
- Tune the PID/LQR gains using the rules of thumb introduced,
- By adjusting the PID/LQR gains and N-rotor vehicle parameters, achieve an understanding of how the flight performance is affected.

Session 4: Practical

- Pre-work: introduction material explaining the experimental system.
- Familiarise with the experimental system,
- Briefly discuss common pitfalls while implementing, and tools available to assist with debugging,
- Students begin with writing and implementing their own controller.

Session 5: Practical

- Students continue with implementing their own PID/LQR controller,
- Once working, the students should gain hands-on experience with adjusting the PID/LQR gains in real time, sensitivity of the flight performance to changing the parameters of the quad-rotor (mass, damaged propeller blades, controller frequency).

Session 6: Practical

- If the previous 2 practical sessions are insufficient, the students can continue with implementing and testing their own controller,
- If some (or all) student groups are finished, they can experiment with ideas of their own. For example flying a specified trajectory, carrying a load, or multiple quad-rotors flying in formation by sending the required information across the network.

Chapter 2

Modelling

The goal of this chapter is to derive the full non-linear equations of motion for the N-rotor vehicle. As discussed in the introduction (Chapter 1), this course will focus on a symmetric quad-rotor for real-world implementation. Despite this, the motivation for deriving the model for an N-rotor vehicle is not "just for the sake of generality", but instead to gain insight into what aspects of the symmetric quad-rotor's design simplifies the synthesis of control algorithms. The frames of reference are defined in §2.1 and then the forces acting on the N-rotor vehicle are described in §2.2 using the "natural" frame of reference for each force. In order to derive the equations of motion all of the forces need to be expressed in the same frame of reference, thus §2.3 derives the transformations between the frames. Putting all these pieces together, §2.4 derives the full non-linear equations of motion for the N-rotor vehicles. The equations of motion derived in §2.1–2.4 will always be used for simulating the N-rotor vehicle because they provide the highest fidelity representation of the physical system. Chapter 3 describes techniques for simulating the equations of motion.

The high fidelity equations of motion cannot readily be used for controller design because of their complexity. Thus, for the purpose of synthesising a controller, a range of approximations are derived that can be used with various controller synthesis methods. In §2.5 we introduce the concepts of "actuators" and use this in Chapter 3 to propose a nested control structure for N-rotor vehicle. Discussion points that related to the material throughout this chapter are collected in §2.6.

A quad-rotor is a commonly encountered flying vehicle, both for domestic entertainment and industrial applications.For the purpose of this script, we use the term *N-rotor vehicle* to refer to a flying vehicle that:

i) is a rigid body device with $N \geq 1$ propellers mounted in the same plane,
ii) has all propellers mounted such that their thrust vectors are normal to the plane in which the propellers are mounted.

2.1 Frames of reference

For the purpose of the derivations in this chapter, a N-rotor vehicle is a rigid body moving in space with N actuators that can apply a thrust force to the rigid body at the location of the propeller. Thus the N-rotor vehicle has 6 degrees of freedom (DoF) that uniquely describe its position and orientation in space. In order to define the position and orientation we use the following conventions:

- An inertial reference frame, denoted I, that has the Cartesian-coordinate z-direction aligned with the gravity field and facing upwards.
- A body reference frame, denoted B, that is rigidly attached to the N-rotor vehicle with the origin placed at the vehicle's centre-of-gravity (CoG) and the Cartesian-coordinate z-direction aligned with the direction of positive thrust.
- Notationally, for all Cartesian coordinates we add a superscript indicating the frame in which the coordinates are given. For example $\vec{v}^{(\mathrm{I})}$ denotes a vector $\vec{v}$ with coordinates given in the inertial frame, and $\vec{v}^{(\mathrm{B})}$ denotes a vector in the body frame.
- The position of the body frame relative to the inertial frame is the vector $\vec{p}^{(\mathrm{I})}$ that points from the origin of I to the origin of frame B and is given in coordinates of the inertial frame.

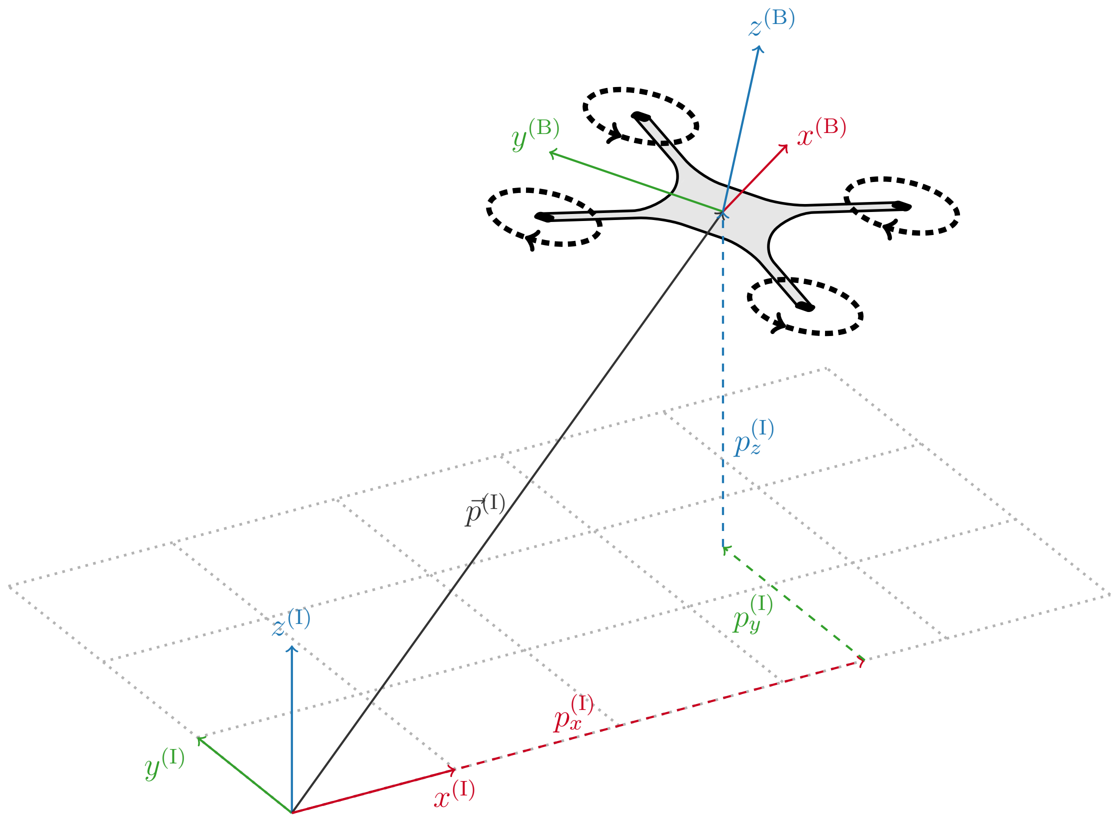


Figure 2.1: The inertial frame of reference I is attached to the ground plane with the $z^{(I)}$ coordinate axis counter-aligned with the gravity vector. The body fixed frame of reference B is rigidly attached to the N-rotor vehicle with the $z^{(B)}$ coordinate axis aligned with the direction of positive thrust from the propellers that we assume are all aligned. This figure depicts a quad-rotor but the reference frame definition is the same for a generic N-rotor vehicle.

2.2 Forces acting on the N-rotor vehicle

The four types of forces acting on the N-rotor vehicle are:

1. Gravity, acting in the negative $z^{(\mathrm{I})}$ direction at the location of the vehicles centre of gravity, denoted as mg in Figure 2.2.
2. Propeller thrust force, acting in the positive $z^{(\mathrm{B})}$ direction at the location of the propeller centre of thrust, denoted as f_i in Figure 2.2.
3. Torque due to aerodynamic drag on the propeller, acting about the positive $z^{(\mathrm{B})}$ axis with opposite sense from the direction of the propeller's rotation, denoted as τ_i in Figure 2.2.
4. Aerodynamic drag on the body of the vehicle as it moves through space. The nature of this force depends heavily on the geometry of the vehicle, not shown in Figure 2.2.

The gravity, propeller thrust and torque forces are shown in Figure 2.2. The gravity force is self explanatory and only requires knowledge of the vehicle mass to evaluate. The remaining forces on the vehicle all stem from aerodynamic phenomena and can be modelled in more or less detail depending on the objective of the modelling exercise.

For the propeller thrust force, we will assume this to be an input to our system that is directly represented as f_i for the purpose of deriving the equations of motion. Practically however, the actuator used to rotate the propeller is an electric motor, and the input applied to the motor is current. It would be possible to derive equations of motion that characterise the rotational speed of the propeller for a current signal, but that is beyond the scope of this course. Each propeller thrust force produces a moment about the body frame $x^{(\mathrm{B})}$ and $y^{(\mathrm{B})}$ axes. As shown in Figure 2.3, the moment produced is proportional to the distance of the propeller's centre from the respective axis.

For the purpose of modelling, we assume that the torque induced on the propeller by aerodynamic drag is linearly proportional to the thrust force. To understand this assumption, consider that each half of the propeller has an aerofoil cross-section and as it rotates through the air the aerofoil shape, much like the wing of an aeroplane, produces a pressure induced reaction force. If we assume that both halves of the propeller experience the same reaction force, then the lift components sum up to provide the thrust force described above, while the drag components together produce a torque that counters the direction of rotation. See Figure 2.4 for a rudimentary representation of these forces. We make the assumption for a propeller on our N-rotor vehicle that the lift-to-drag ratio remains constant over the operating range of the motor. Thus we model the torque produced as proportional to the thrust force, $\tau_i = c_i f_i$, where c_i is the constant of proportionality and needs to be identified for each propeller. The constant of proportionality is determined experimentally.

The aerodynamic drag on the body can be reasonably neglected when the linear velocity of the N-rotor vehicle is low. We will include a term for this force in the equations of motion derived in §2.4 but neglect it for all other purposes. In some settings the aerodynamic drag on the body should be adequately identified and included to achieve acceptable flight performance, see [9] for example.

2.2.1 Blade Flapping

Blade flapping is a phenomena that occurs when a propeller experiences a non-zero relative air speed which results in the resultant thrust force produced by the propeller being no longer aligned with its axis of rotation. As highlighted in Figure 2.5, at each instant during the cycle of the propellers rotation, each half of the propeller experiences a different relative air speed, this changes the magnitude of the total reaction force, labelled as f_{TOTAL}. Assuming that the lift to drag ratio remains constant we see that this difference in the total reaction force leads to an asymmetry in the lift and drag forces on each half of the propeller. Thus the total propeller thrust force is no longer aligned with the propeller's axis of rotation, and the torque also acts about an axis different from the axis of rotation. This explanation is simplified, but it serves to highlight what situations gives rise to blade flapping.

Blade flapping is an important phenomena that must be correctly analysed and accounted for when designing and flying a full scale helicopter that regularly fly at air speeds in excess of 100 km/h. For achieving stable hover of a quad-rotor in a controller laboratory environment, the effects of blade flapping are negligible and can be reasonably ignored.

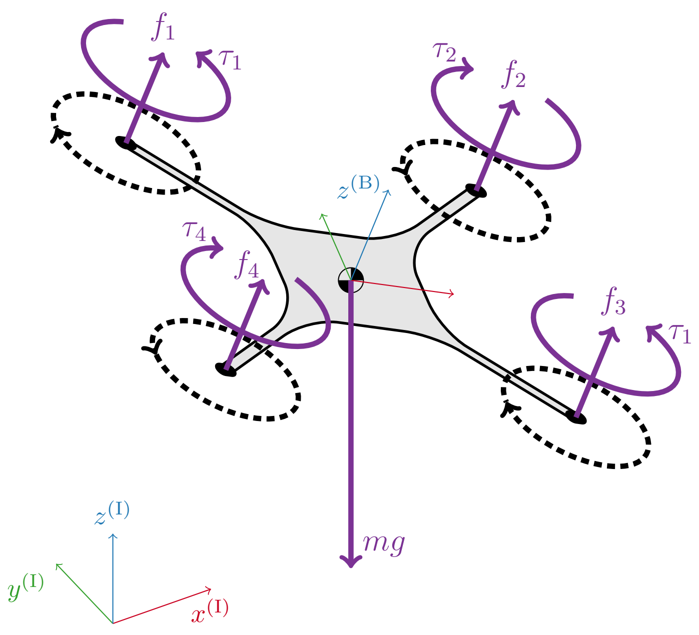


Figure 2.2: Showing the forces acting on a quad-rotor vehicle. The gravity force mg acts in the negative $z^{(\mathrm{I})}$ direction at the centre of gravity of the vehicle. The propeller thrust forces f_i all act in the $z^{(\mathrm{B})}$ direction at the respective propeller axis of rotation. The aerodynamic drag on the rotating propeller induces the torques τ_i that all act about the $z^{(\mathrm{B})}$ and with a sense that opposes the rotation of the respective propeller.

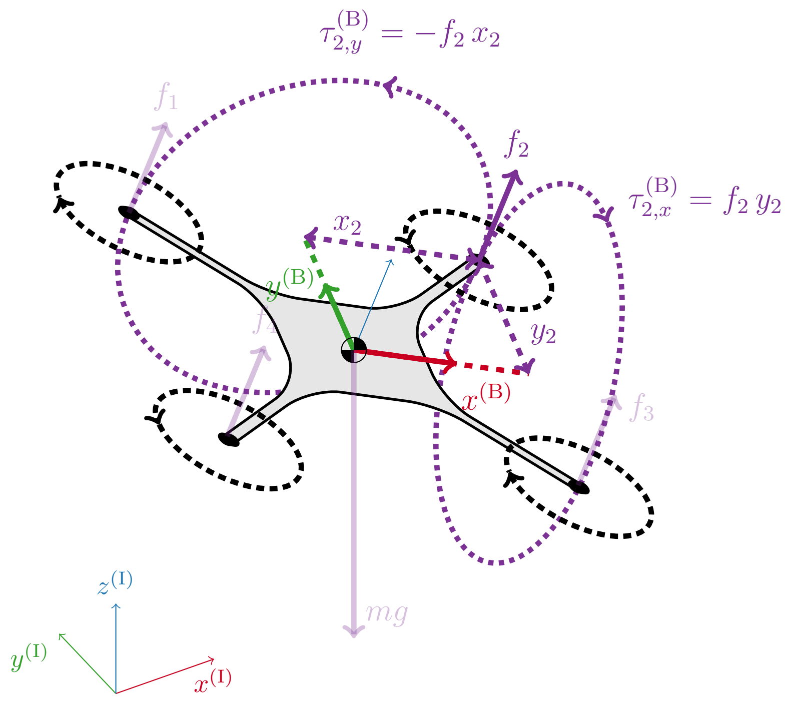


Figure 2.3: Showing the moment about the body frame $x^{(\mathrm{B})}$ and $y^{(\mathrm{B})}$ axes due to a propeller thrust force. The diagram highlights the moments arising from f_2 and is readily derived for the other thrust forces. In general, the moment about the $x^{(\mathrm{B})}$ axis is $f_i y_i$ and about the $y^{(\mathrm{B})}$ axis is $-f_i x_i$, where (x_i, y_i) is the location of the propeller axis relative to the vehicle's centre of gravity.

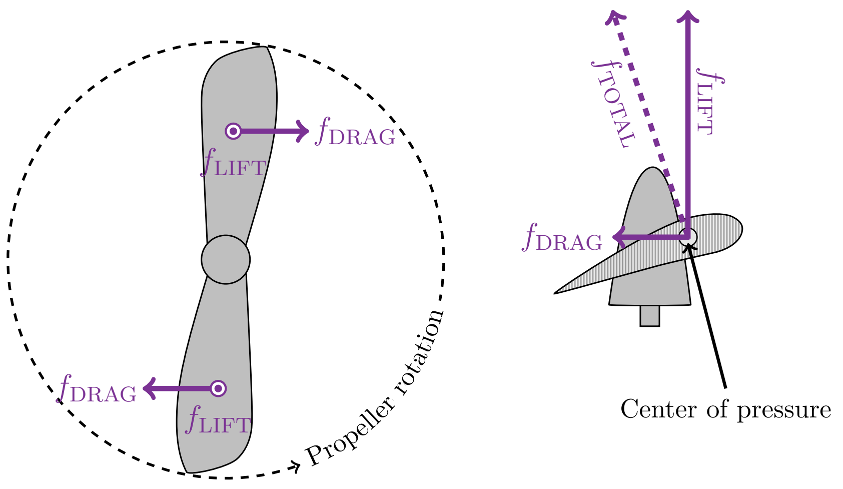


Figure 2.4: Showing the aerodynamic forces acting on a propeller. Assuming that both halves of the propeller are identical and produce the same f_{TOTAL} force vector, then the total thrust produced by the lift component can be modelled as acting at the propeller's centre, and the drag forces modelled as a single torque that acts about the axis of the propeller's rotation.

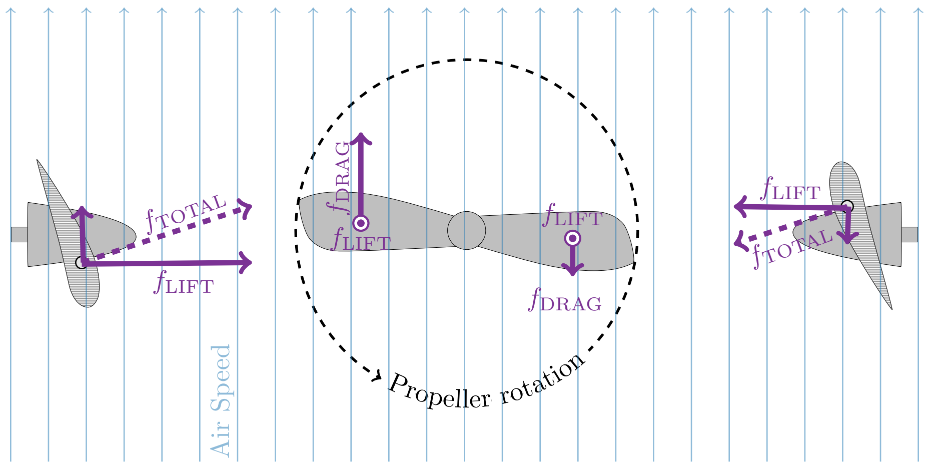


Figure 2.5: Showing the aerodynamic forces acting on a propeller which is moving through the air and hence experiences the relative wind indicated by the blue lines. At the position in the propeller's rotation pictured, the right half experiences a lower relative airspeed than the left half. Thus the total aerodynamic reaction force is less for the right half of the propeller than for the left half. This asymmetry in the aerodynamic reaction forces produces a propeller thrust force that is no longer aligned with the axis of rotation. This phenomena is referred to as *blade flapping*.

2.3 Frame transformations

In order to write down the full non-linear equations of motion we need to express all the forces acting on the N-rotor vehicle in the same reference frame. In §2.2 above we observed that some forces are naturally expressed in the body frame, while others are naturally expressed in the inertial frame. Expressed in words, the transformation we need to mathematically describe is: *when the origin of frames* I *and* B *are co-located, what rotation needs to be applied such that their coordinate axes are aligned.*

As described in Section 2.1 there are three degrees of freedom related to the orientation of the body frame relative to the inertial frame. Thus it is natural to construct a sequence of three rotations that allow the description of any relative orientation, and in this script we use what is referred to as the intrinsic (z, y', x'') Euler angles.

2.3.1 Intrinsic (z, y', x'') Euler angles

The intrinsic (z, y', x'') Euler angles describe a general transformation from frame I to frame B by the following sequence of three coordinate axis rotations:

1. A rotation about the $z^{(\mathrm{I})}$ axis by α, called the *yaw* angle. The new frame is denoted A_1, see Figure 2.6.
2. A rotation about the $y^{(A_1)}$ axis by β, called the *pitch* angle. The new frame is denoted A_2, see Figure 2.6.
3. A rotation about the $x^{(A_2)}$ axis by γ, called the *roll* angle. The new frame is now frame B, see Figure 2.6.

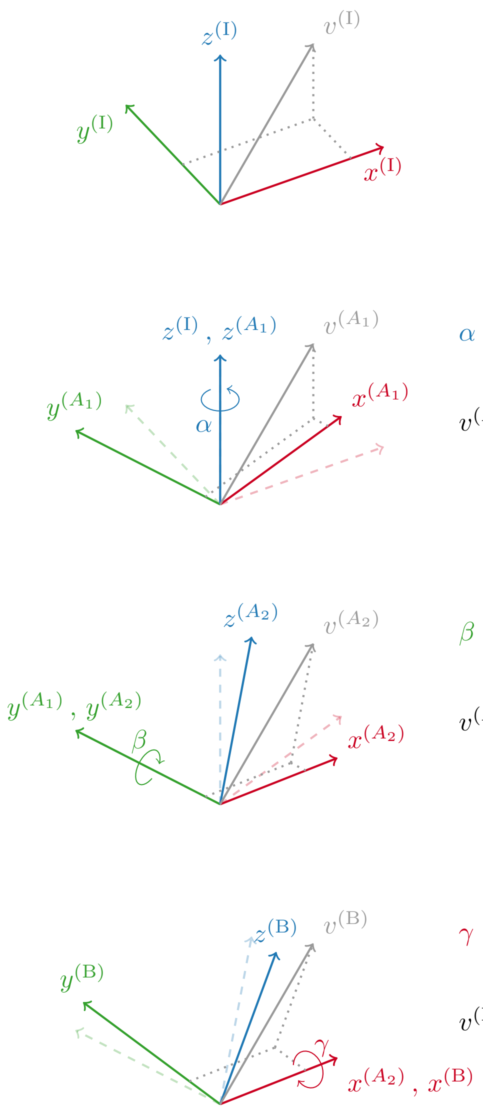
Each step is a rotation about a coordinate axis, thus we can directly write down the rotation matrix for each step. For step (i), a vector $v^{(I)}$ expressed in the inertial frame, is transformed to the equivalent vector expressed in the A_1 frame by the following rotation matrix,

$$v^{(A_1)} = \begin{bmatrix} \cos(\alpha) & sin(\alpha) & 0 \\ -\sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} v^{(I)} = \underbrace{\begin{bmatrix} c_\alpha & s_\alpha & 0 \\ -s_\alpha & c_\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{{}_{(A_1)}R^{(\mathrm{I})}} v^{(I)} = {}_{(A_1)}R^{(I)}\, v^{(I)} \,. \tag{2.1}$$

We have introduced the short-hand notation s_θ and c_θ for the sine and cosine of an angle θ, and the notation ${}_{(A_1)}R^{(\mathrm{I})}$ for a rotation matrix indicating that when right multiplied by a vector in frame I the result is the same vector expressed with respect to the coordinates of frame A_1. The three steps are shown in Figure 2.6 also with the rotation matrices given.

(Proper) Rotation matrices have the following properties:

- Determinant equal to 1, hence always invertible.
- The inverse of a rotation matrix is a rotation of the same angle about the same axis but with the opposite sense.
- The inverse of a rotation matrix is equal to its transpose.
- The rotation matrix for a sequence of rotations is constructed by matrix-multiplication of the individual rotation matrices.



α := the *yaw* angle

$$v^{(A_1)} = {}_{(A_1)}R^{(\mathrm{I})}\, v^{(\mathrm{I})} = \begin{bmatrix} c_\alpha & s_\alpha & 0 \\ -s_\alpha & c_\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} v^{(\mathrm{I})}$$

β := the *pitch* angle

$$v^{(A_2)} = {}_{(A_2)}R^{(A_1)}\, v^{(A_1)} = \begin{bmatrix} c_\beta & 0 & -s_\beta \\ 0 & 1 & 0 \\ s_\beta & 0 & c_\beta \end{bmatrix} v^{(A_1)}$$

γ := the *roll* angle

$$v^{(\mathrm{B})} = {}_{(\mathrm{B})}R^{(A_2)}\, v^{(A_2)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\gamma & s_\gamma \\ 0 & -s_\gamma & c_\gamma \end{bmatrix} v^{(A_2)}$$

Figure 2.6: The sequence of rotations (from top to bottom) defined by the intrinsic (z, y', x'') Euler angles that describe the orientation of the body frame B relative to the inertial frame I. The intermediate frames A_1 and A_2 have been introduced to simplify the exposition of the sequence of rotations. In each diagram the dashed coordinate axes are identical to the solid axes from the diagram above.

The rotation matrix from the inertial frame I to the body B is constructed as follows,

$$\begin{aligned}
v^{(\mathrm{B})} &= \quad {}_{(\mathrm{B})}R^{(A_2)} \qquad {}_{(A_2)}R^{(A_1)} \qquad {}_{(A_1)}R^{(\mathrm{I})} \qquad v^{(\mathrm{I})} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\gamma & s_\gamma \\ 0 & -s_\gamma & c_\gamma \end{bmatrix} \begin{bmatrix} c_\beta & 0 & -s_\beta \\ 0 & 1 & 0 \\ s_\beta & 0 & c_\beta \end{bmatrix} \begin{bmatrix} c_\alpha & s_\alpha & 0 \\ -s_\alpha & c_\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} v^{(\mathrm{I})} \\
&= \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\gamma & s_\gamma \\ 0 & -s_\gamma & c_\gamma \end{bmatrix} \begin{bmatrix} c_\beta\, c_\alpha & c_\beta\, s_\alpha & -s_\beta \\ -s_\alpha & c_\alpha & 0 \\ s_\beta\, c_\alpha & s_\beta\, s_\alpha & c_\beta \end{bmatrix} v^{(\mathrm{I})} \\
&= \begin{bmatrix} c_\beta\, c_\alpha & c_\beta\, s_\alpha & -s_\beta \\ (-c_\gamma\, s_\alpha + s_\gamma\, s_\beta\, c_\alpha) & (c_\gamma\, c_\alpha + s_\gamma\, s_\beta\, s_\alpha) & s_\gamma\, c_\beta \\ (s_\gamma\, s_\alpha + c_\gamma\, s_\beta\, c_\alpha) & (-s_\gamma\, c_\alpha + c_\gamma\, s_\beta\, s_\alpha) & c_\gamma\, c_\beta \end{bmatrix} v^{(\mathrm{I})} \\
&= {}_{(\mathrm{B})}R^{(\mathrm{I})}\; v^{(\mathrm{I})}
\end{aligned} \tag{2.2}$$

The rotation matrix to transform a vector in the body frame B to its equivalent in the inertial frame I is simply the inverse of this,

$$\begin{aligned}
v^{(\mathrm{I})} &= \left({}_{(\mathrm{B})}R^{(\mathrm{I})}\right)^{-1} v^{(\mathrm{B})} \\
&= \left({}_{(\mathrm{B})}R^{(\mathrm{I})}\right)^{\top} v^{(\mathrm{B})} \\
&= \left({}_{(\mathrm{B})}R^{(A_2)}\;{}_{(A_2)}R^{(A_1)}\;{}_{(A_1)}R^{(\mathrm{I})}\right)^{\top} v^{(\mathrm{B})} \\
&= \left({}_{(A_1)}R^{(\mathrm{I})}\right)^{\top} \left({}_{(A_2)}R^{(A_1)}\right)^{\top} \left({}_{(\mathrm{B})}R^{(A_2)}\right)^{\top} v^{(\mathrm{B})} \\
&= {}_{(\mathrm{I})}R^{(A_1)}\;{}_{(A_1)}R^{(A_2)}\;{}_{(A_2)}R^{(\mathrm{B})}\; v^{(\mathrm{B})} \\
&= \begin{bmatrix} c_\alpha\, c_\beta & (-s_\alpha\, c_\gamma + c_\alpha\, s_\beta\, s_\gamma) & (s_\alpha\, s_\gamma + c_\alpha\, s_\beta\, c_\gamma) \\ s_\alpha\, c_\beta & (c_\alpha\, c_\gamma + s_\alpha\, s_\beta\, s_\gamma) & (-c_\alpha\, s_\gamma + s_\alpha\, s_\beta\, c_\gamma) \\ -s_\beta & c_\beta\, s_\gamma & c_\beta\, c_\gamma \end{bmatrix} v^{(\mathrm{B})} \\
&= {}_{(\mathrm{I})}R^{(\mathrm{B})}\; v^{(\mathrm{B})}
\end{aligned} \tag{2.3}$$

Now if we are given a measurement of the intrinsic (z, y', x'') Euler angles (i.e., roll, pitch, and yaw) that describe the orientation of frame B relative to frame I, then we can readily construct the matrices that rotate vectors between the two frames.

2.3.2 Gimbal lock

In §-2.3.1 above we introduced the intrinsic (z, y', x'') Euler angle representations that allow the orientation of the body frame B to be constructed by a sequence of three coordinate axes rotations starting from the inertial frame I. The concept of Gimbal lock relates to the inverse problem: *given two frames, what set(s) of Euler angles describe their relative orientation?*. When the answer is that the Euler angles representation is not unique, then this is a point of Gimbal lock. This is also observed as a loss of a degree of freedom in the rotation matrix.

An additional downside of Euler angle representation is that the map from Euler angles to rotations (i.e., the formula for ${}_{(\mathrm{I})}R^{(\mathrm{B})}\; v^{(\mathrm{B})}$ derived above) is not a *local homeomorphism*. In other words, at certain points in the space of rotations, some continuous changes in the rotation cannot be described by a continuous change in the space of Euler angles. This downside is why 3D modelling software and games must take care to avoid the pitfalls of Gimbal lock. The YouTube video "Euler (gimbal lock) Explained" [4] provides some nice animations to highlight gimbal lock along trajectories.

2.3.3 Alternatives for frame transformations

The (z, y', x'') ordering of Euler angles used in §2.3.1 above is not the only choice. In fact 6 permutations of the order provide a valid set of Euler angles for describing the rotation of an object, i.e.,

$$\begin{array}{ccc} (z, y', x'') & (y, x', z'') & (x, z', y'') \\ (z, x', y'') & (y, z', x'') & (x, y', z'') \end{array}$$

It is also possible to construct a set of Euler angles as a sequence of three rotations each about one of the inertial frame coordinate axes. These are referred to as extrinsic (or Proper) Euler angles. For example, the extrinsic (ZXZ) Euler angles from frame I to B describe the following sequence of three coordinate axis rotations:

1. A rotation about the $z^{(\mathrm{I})}$.
2. A rotation about the $x^{(\mathrm{I})}$.
3. A rotation about the $z^{(\mathrm{I})}$.

A valid set of Euler angles is obtained for the following sequences of extrinsic rotations,

$$\begin{array}{ccc} (Z, X, Z) & (Y, X, Y) & (Z, Y, Z) \\ (X, Z, X) & (X, Y, X) & (Y, Z, Y) \end{array}$$

Each of these choices for the Euler angles experiences Gimbal lock at different points in the rotation space.

In order to avoid the phenomena of Gimbal lock, many applications use either *quaternions* or rotation matrices. Loosely speaking, a quaternion is a vector of 4 elements used to represent a single rotation axis and angle of rotation. The space of rotation matrices is the space of $\mathbb{R}^{3\times 3}$ orthogonal matrices with determinant $+1$ and is called the *special orthogonal group*, denoted as SO(3). See, for example, [5], [6], and [2] for further details about quaternions and SO(3) rotation matrices.

2.4 Non-linear equations of motion

Now that we have established the frames of reference (§2.1), the forces acting on the vehicle (§2.2), and an Euler angle convention for describing the attitude of the vehicle (§2.3), we have all the ingredients needed to derive the equations of motion. We cover first the derivations for the equations of motion describing the translation of the N-rotor vehicle because these are a direct application of the concepts introduced in the previous sections. We then cover the derivations of the angular equations of motion, introducing a few new concepts required for those derivations.

2.4.1 Equations of motion for translation

Recall from §2.2 above that the linear forces acting on the N-rotor vehicle are gravity, propeller thrust forces, and aerodynamic drag on the body of the vehicle. We denote each of these as follows, with the frame of the vector indicated,

$$\vec{f}^{(\mathrm{B})}_{\mathrm{thrust}} \;=\; \begin{bmatrix} 0 \\ 0 \\ \sum\limits_{i=1}^{N} f_i \end{bmatrix} ,\quad \vec{f}^{(\mathrm{I})}_{\mathrm{gravity}} \;=\; \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} ,\quad \vec{f}^{(\mathrm{B})}_{\mathrm{body\ drag}} \;=\; \begin{bmatrix} -c_{d,x}\,\dot{p}_x^{(\mathrm{B})2} \\ -c_{d,y}\,\dot{p}_y^{(\mathrm{B})2} \\ -c_{d,z}\,\dot{p}_z^{(\mathrm{B})2} \end{bmatrix} . \tag{2.4}$$

As a brief side comment on notation, the subscripts for these forces are not concise, we make this choice to ease the amount of notation the reader needs to remember and make the equations more readable. Although the literature on flying vehicles has some consistent notation, we find that differences in the subtleties and exact meaning of x, y, z, f, τ, g quickly become confusing. It is for this reason that we favour less concise and more self-explanatory notation at times.

Using Newton's second law of motion we directly write down the equations of motion for translation of the vehicle, with the rotation matrix ${}_{(\mathrm{I})}R^{(\mathrm{B})}$ being used as necessary to rotate body frame forces into the inertial frame.

$$\ddot{\vec{p}}^{(\mathrm{I})} \;=\; \begin{bmatrix} \ddot{p}_x^{(\mathrm{I})} \\ \ddot{p}_y^{(\mathrm{I})} \\ \ddot{p}_z^{(\mathrm{I})} \end{bmatrix} \;=\; \frac{1}{m}\left({}_{(\mathrm{I})}R^{(\mathrm{B})}\,\vec{f}^{(\mathrm{B})}_{\mathrm{thrust}} \;+\; \vec{f}^{(\mathrm{I})}_{\mathrm{gravity}} \;+\; {}_{(\mathrm{I})}R^{(\mathrm{B})}\,\vec{f}^{(\mathrm{B})}_{\mathrm{body\ drag}} \right) . \tag{2.5}$$

If we now assume the drag forces on the body to be negligible, and substitute in the expressions for the rotation matrix and the forces, the equations of motion for translation become,

$$\ddot{\vec{p}}^{(\mathrm{I})} \;=\; \begin{bmatrix} \ddot{p}_x^{(\mathrm{I})} \\ \ddot{p}_y^{(\mathrm{I})} \\ \ddot{p}_z^{(\mathrm{I})} \end{bmatrix} \;=\; \frac{1}{m}\begin{bmatrix} \left(s_\alpha\, s_\gamma \;+\; c_\alpha\, s_\beta\, c_\gamma\right)\left(\sum\limits_{i=1}^{N} f_i\right) \\ \left(-c_\alpha\, s_\gamma \;+\; s_\alpha\, s_\beta\, c_\gamma\right)\left(\sum\limits_{i=1}^{N} f_i\right) \\ \left(c_\beta\, c_\gamma\right)\left(\sum\limits_{i=1}^{N} f_i\right) \;-\; mg \end{bmatrix} . \tag{2.6}$$

Recall from the descriptions in §2.2 that we use the individual forces f_i as the input to our system description, and that equations of motion for translation of the vehicle depend only on the sum of these inputs. This makes intuitive sense because we have assumed that the N propellers on the vehicle produce individual thrust vectors that are aligned with each other.

If we design a vehicle which can adjust the direction of each force vector f_i as additional inputs to the system, then the resulting equations of motion for translation will be more complex. See for example the "Voliro" hexa-roter project that is a collaboration between ETHZ and ZHDK aiming to build and control such a vehicle [1].

2.4.2 Equations of motion for rotation

The angular equations of motion describe the angular acceleration of the Euler angles we have chosen for describing the attitude of the body frame relative to the inertial frame. One may also attempt to derive angular equations of motion directly via Newton's second law of motion, i.e.,

$$\frac{d^2}{dt^2}\left(J\,\vec{\psi} \right) \;=\; \frac{d^2}{dt^2}\left(J \begin{bmatrix} \gamma \\ \beta \\ \alpha \end{bmatrix} \right) \;=\; \begin{bmatrix} x^{(\mathrm{B})} \text{ component of torques} \\ y^{(A_1)} \text{ component of torques} \\ z^{(\mathrm{I})} \text{ component of torques} \end{bmatrix} ,$$

where $J \in \mathbb{R}^{3\times 3}$ is the rotational inertia matrix of the vehicle. However, this approach is complicated by the fact that we need the torques about axes from three different coordinate systems and two of the coordinate systems are rotating differently relative to the inertial frame so the time derivatives need to be correctly handled.

In §2.2 we saw that when considering the torques about the centre of gravity of the vehicle, the gravity force produces zero moment, and the remaining torques and moments are all naturally expressed in the body frame coordinates. In other words, the induced torque due to drag on the propeller is always about the body frame $z^{(\mathrm{B})}$ axis and proportional to the propeller's thrust force. Similarly, the moment produced about the body frame $x^{(\mathrm{B})}$ and $y^{(\mathrm{B})}$ axes are proportional to the propeller's thrust force. Thus we will first write down the equations of motion for rotation of the body frame.

In order to write the equations of motion for rotation of the body frame, we introduce the following notation for the angular rotation rate of the body frame about its own coordinate axes,

$$\vec{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = \begin{bmatrix} \text{Angular velocity of frame B about } x^{(\mathrm{B})} \\ \text{Angular velocity of frame B about } y^{(\mathrm{B})} \\ \text{Angular velocity of frame B about } z^{(\mathrm{B})} \end{bmatrix} . \tag{2.7}$$

The equations of motion for $\vec{\omega}$ are thus,

$$\frac{d}{dt}\left(J\,\vec{\omega}\right) = J\,\dot{\vec{\omega}} + \left(\vec{\omega} \times J\,\vec{\omega}\right) = \vec{\tau}^{(\mathrm{B})} = \begin{bmatrix} \sum_{i=1}^{N} & f_i\,y_i \\ \sum_{i=1}^{N} & -f_i\,x_i \\ \sum_{i=1}^{N} & f_i\,c_i \end{bmatrix} , \tag{2.8}$$

where the term $(\vec{\omega} \times J\vec{\omega})$ arises because the vector $\vec{\omega}$ of which we take the time derivative is in a rotating frame, and $\vec{\tau}^{(\mathrm{B})}$ simply represents the total torque vector acting about the origin of the body frame. The expressions for the body frame torques were described in §2.2 above. Equation (2.8) is sometimes referred to as *Euler's equations for rigid body dynamics*. If the body frame is aligned with the principle axes of rotation of the vehicle, then the inertia matrix, J will be diagonal and the equations of motion for $\vec{\omega}$ simplify to,

$$\begin{bmatrix} J_x\,\dot{\omega}_x + (J_z - J_y)\,\omega_y\omega_z \\ J_y\,\dot{\omega}_y + (J_x - J_z)\,\omega_x\omega_z \\ J_z\,\dot{\omega}_z + (J_y - J_x)\,\omega_x\omega_y \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N} & f_i\,y_i \\ \sum_{i=1}^{N} & -f_i\,x_i \\ \sum_{i=1}^{N} & f_i\,c_i \end{bmatrix} , \tag{2.9}$$

where J_x, J_y, and J_z are the components of the rotational inertia about the respective principle axes of rotation.

Given that we have derived the equations of motion for rotation of the body frame, it remains to derive the relation between the angular acceleration of the Euler angles $\ddot{\vec{\psi}}$, and the body rates $\vec{\omega}$. We start from the geometrical relationship between the angular velocities of the Euler angles $\dot{\vec{\psi}}$, and the body rates, i.e.,

$$\vec{\omega} = T(\vec{\psi})\,\dot{\vec{\psi}}, \tag{2.10}$$

where the transformation matrix $T(\vec{\psi})$ is given by,

$$T(\vec{\psi}) = \begin{bmatrix} 1 & 0 & -\sin(\beta) \\ 0 & \cos(\gamma) & \sin(\gamma)\cos(\beta) \\ 0 & -\sin(\gamma) & \cos(\gamma)\cos(\beta) \end{bmatrix} . \tag{2.11}$$

This transformation matrix is derived by rotating each component of $\dot{\vec{\psi}}$ into the body frame, i.e.,

$$\vec{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = {}_{(\mathrm{B})}R^{(\mathrm{B})} \begin{bmatrix} \dot{\gamma} \\ 0 \\ 0 \end{bmatrix} + {}_{(\mathrm{B})}R^{(A_2)} \begin{bmatrix} 0 \\ \dot{\beta} \\ 0 \end{bmatrix} + {}_{(\mathrm{B})}R^{(A_1)} \begin{bmatrix} 0 \\ 0 \\ \dot{\alpha} \end{bmatrix} . \tag{2.12}$$

The inverse of $T(\vec{\psi})$ can be analytically derived and is the following,

$$T^{-1}(\vec{\psi}) = \begin{bmatrix} 1 & \sin(\gamma)\tan(\beta) & \cos(\gamma)\tan(\beta) \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma)\sec(\beta) & \cos(\gamma)\sec(\beta) \end{bmatrix} . \tag{2.13}$$

It is left as an exercise for the reader to derive the expression for $T(\vec{\psi})$, and to show that the expression for $T^{-1}(\vec{\psi})$ is actually its inverse.

Finally, by taking the time derivation of relation (2.10) we can derive the equation of motion for the Euler angles,

$$\begin{aligned}\ddot{\vec{\psi}} &= \frac{d}{dt}\dot{\vec{\psi}} \\ &= \frac{d}{dt}\left(T^{-1}(\vec{\psi})\,\vec{\omega}\right) \\ &= \left(T^{-1}(\vec{\psi})\,\dot{\vec{\omega}}\right) + \left(\dot{T}^{-1}(\vec{\psi},\dot{\vec{\psi}})\,\vec{\omega}\right) \\ &= \left(T^{-1}(\vec{\psi})\,\dot{\vec{\omega}}\right) + \left(\dot{T}^{-1}(\vec{\psi},\dot{\vec{\psi}})\,T(\vec{\psi})\,\dot{\vec{\psi}}\right) \\ &= \left(T^{-1}(\vec{\psi})\,\dot{\vec{\omega}}\right) - \left(T^{-1}(\vec{\psi})\,\dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\dot{\vec{\psi}}\right) \\ &= T^{-1}(\vec{\psi})\,\left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\dot{\vec{\psi}}\right)\,,\end{aligned} \tag{2.14}$$

where the third equality is an application of the product rule, the fourth equality used the identity (2.10) to remove the dependence on $\vec{\omega}$, and the fifth equality uses the identity that $\frac{d}{dt}(I) = \frac{d}{dt}\left(T(\vec{\psi})T^{-1}(\vec{\psi})\right) = 0$.

In summary, equations (2.5), (2.8), and (2.14) together are the non-linear equations of motion that describe the evolution of our N-rotor vehicle under the influence of the motor thrusts $\{f_i\}_{i=1}^{N}$,

$$\ddot{\vec{p}} = \begin{bmatrix}\ddot{p}_x\\ \ddot{p}_y\\ \ddot{p}_z\end{bmatrix} = \frac{1}{m}\left({}_{(\mathrm{I})}R^{(\mathrm{B})}(\vec{\psi})\begin{bmatrix}0\\0\\ \sum\limits_{i=1}^{N} f_i\end{bmatrix} + \begin{bmatrix}0\\0\\-mg\end{bmatrix}\right)\,, \tag{2.15a}$$

$$\ddot{\vec{\psi}} = \begin{bmatrix}\ddot{\gamma}\\ \ddot{\beta}\\ \ddot{\alpha}\end{bmatrix} = T^{-1}(\vec{\psi})\,\left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\dot{\vec{\psi}}\right)\,, \tag{2.15b}$$

$$\dot{\vec{\omega}} = \begin{bmatrix}\dot{\omega}_x\\ \dot{\omega}_y\\ \dot{\omega}_z\end{bmatrix} = J^{-1}\left(\begin{bmatrix}\sum_{i=1}^{N} & f_i\,y_i\\ \sum_{i=1}^{N} & -f_i\,x_i\\ \sum_{i=1}^{N} & f_i\,c_i\end{bmatrix} - (\vec{\omega}\times J\,\vec{\omega})\right)\,. \tag{2.15c}$$

These equations of motion can now be implemented using a software that allows for numerical integration of ordinary differential equations, for example Simulink [8]. Chapter 3 covers the details of simulating these equations of motion using Simulink.

2.5 Actuators

Despite the N-rotor vehicle having N separate propellers, each capable of producing a thrust force, the $\{f_i\}_{i=1}^N$ forces only enter the equations of motion (2.15) through four terms, namely:

- $\sum\limits_{i=1}^{N} f_i$, the total thrust force
- $\sum_{i=1}^{N} \; f_i \, y_i$, the resultant torque about the body frame $x^{(\mathrm{B})}$ axis,
- $\sum_{i=1}^{N} -f_i \, x_i$, the resultant torque about the body frame $y^{(\mathrm{B})}$ axis,
- $\sum_{i=1}^{N} \; f_i \, c_i$, the resultant torque about the body frame $z^{(\mathrm{B})}$ axis.

For this reason it is natural, and common for controller design, to consider as inputs to our system only these four terms, and then compute the N separate propeller thrust forces that provide the desired total thrust and body frame torques. We refer to each of these terms as an *actuator* to our system, and notationally define them as follows,

$$f_{\mathrm{total}} = \sum_{i=1}^{N} f_i \,, \tag{2.16a}$$

$$\tau_x^{(\mathrm{B})} = \sum\nolimits_{i=1}^{N} \; f_i \, y_i \,, \tag{2.16b}$$

$$\tau_y^{(\mathrm{B})} = \sum\nolimits_{i=1}^{N} -f_i \, x_i \,, \tag{2.16c}$$

$$\tau_z^{(\mathrm{B})} = \sum\nolimits_{i=1}^{N} \; f_i \, c_i \,. \tag{2.16d}$$

See Figure 2.7 for a visualisation of these actuators on a quad-rotor airframe. Thus we can compute the actuator values using a "matrix-vector" multiplication,

$$\begin{bmatrix} f_{\mathrm{total}} \\ \tau_x^{(\mathrm{B})} \\ \tau_y^{(\mathrm{B})} \\ \tau_z^{(\mathrm{B})} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \cdots & 1 \\ y_1 & \cdots & y_N \\ -x_1 & \cdots & -x_N \\ c_1 & \cdots & c_N \end{bmatrix}}_{M_{\mathrm{layout}}} \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix} , \tag{2.17}$$

where this M_{layout} matrix contain all the relevant information about how the N propellers are laid out on the airframe of the vehicle.

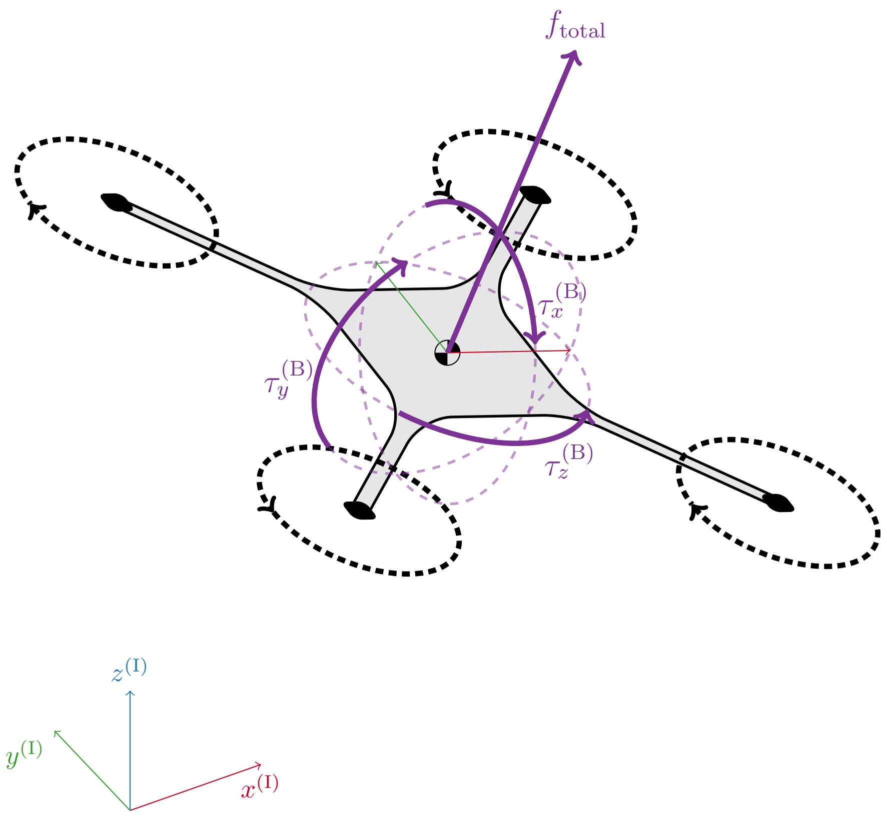


Figure 2.7: Visualisation of the four actuators that act on the N-rotor vehicle, namely the total thrust force f_{total}, and the torque about each axis of the body frame $(\tau_x^{(B)}, \tau_y^{(B)}, \tau_z^{(B)})$. These four actuators represent the four system input terms that enter the equations of motion derived in §2.4 and summarised in equation (2.15).

2.6 Discussion Points

2.6.1 Frames of reference:

a) For the Crazyflie 2.0 quad-rotor that we will use in experiments, is the rigid body assumption valid? Is it reasonable?

b) What is the minimum number of rotors to sustain stable flight?

2.6.2 Forces acting on the N-rotor vehicle:

a) Consider a symmetrical quad-rotor vehicle, if the Centre of Gravity does not coincide with its geometrical centre, how is this modelled in terms of the specifying the forces acting on the vehicle?

b) The modelling theory presented throughout this chapter was for an N-rotor vehicle, but all the explanatory diagrams were presented with symmetric quad-rotor. Figure 2.8 provides an example of an octo-rotor vehicle and the accompanying table specifies the details. If the total vehicle weight is 0.9 kilograms, scrutinise (i.e., provide a critical analysis of) the design. Consideration should be given to, but is not limited to, the following: (i) characterisation of equilibrium thrust, (ii) actuation available for roll, pitch, and yaw manoeuvres, (iii) redundancy, i.e., ability to maintaining stable flight in the case that a propeller ceases to function.

c) As described above, we neglect the dynamics for the electrical motor used to rotate the propellers, what is the assumption we are making by neglecting this term?

d) Design a layout for your own N-rotor vehicle, guidance will be provided in class.

2.6.3 Frame Transformations:

a) The angles (α, β, γ) are defined as a positive rotation of the coordinate system about the specified axis, however, the rotation matrix ${}_{(A_1)}R^{(\mathrm{I})}$ used is a negative rotation about this axis. Explain why this is consistent with the convention.

b) Is the order of rotation important? Do rotation matrices commute?

c) Let one of the intrinsic (z, y', x'') Euler angles be $\frac{\pi}{2}$ and show that the rotation matrix also indicates gimbal lock. Hint: use the trigonometric identities of the form $\cos(\theta)\,\cos(\phi) = \cos(\theta - \phi) + \cos(\theta + \phi)$.

d) Think of a trajectory in the rotation space for which the trajectory in the Euler angle space is not continuous.

e) If we use Euler angles in a control design for an N-rotor vehicle, explain whether the possibility of gimbal lock needs to be adequately addressed for the following situations: (i) hovering, (ii) performing a pirouette (i.e., a 360 degree rotation about $z^{(\mathrm{I})}$), (iii) performing a "back flip".

f) If we use Euler angles in the equations of motion of an N-rotor vehicle, explain why passing through a gimbal lock orientation will cause the equations of motion to predict non-physical behaviour of the vehicle.

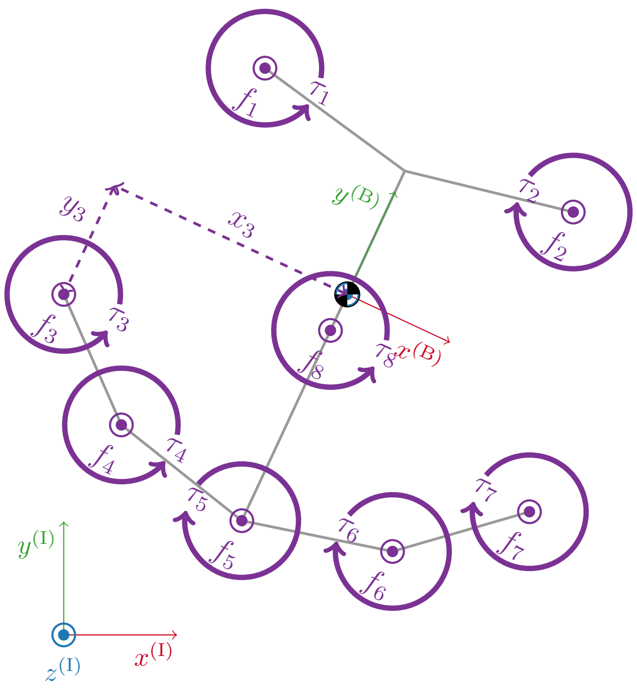
2.6.4 Full non-linear equations of motion:

a) Derive the expression for $T(\vec{\psi})$ and show that the expression given for $T^{-1}(\vec{\psi})$ is its inverse.

b) At what attitudes $\vec{\psi}$ does the transformation matrix $T(\vec{\psi})$ become singular, i.e., not invertible? Explain how this relates to Gimbal lock as discussed in §2.3.2.

c) In the derivations for $\dot{\vec{\omega}}$ the term $(\vec{\omega} \times J\vec{\omega})$ was necessary to account for taking a time derivative in a rotating frame. Explain why a similar term was not included in equation (2.14) where we take the time derivative of a similar quantity.

d) Is it possible to relax the assumption that all the propeller thrust vectors are aligned? Which parts of the equations of motion derivation change in this case? Do these changes to the equations of motion complicate the controller design?



Propeller #	x	y	Handedness	Thrust [N]	
				Min	Max
1	0.2	a	+	0	1
2	b	a	−	0	1
3	c	d	+	0	1
4	c	d	+	0	1
5	c	d	−	0	1
6	c	d	−	0	1
7	c	d	−	0	1
8	c	d	+	0	10

Figure 2.8: The "smiley face" octo-rotor vehicle, see discussion point 2.6.2 (b) for the context.

2.6.5 Actuators:

a) Do we loose some generality by condensing the N separate propeller thrust forces into the four actuators described?

b) Is it always possible to find a set of propeller thrust forces that produces any desired actuation? Is it unique? If no for either question, then provide an example.

c) For a symmetric quad-rotor, find an expression for computing the propeller thrust forces to apply to achieve a desired set of total thrust and body torque actuation values.

Chapter 3

Simulation

For (almost) all practical systems, it is prohibitively costly and time-consuming to test and tune a control algorithm on the real-world system directly. Thus, the usual approach is to test and tune a controller via a simulation-based analysis. In order that the control performance achieved in simulation translates to a similar performance on the real-world system, the simulation-environment used must represent the system sufficiently well. In this chapter we describe techniques for simulation of the non-linear, continuous-time, equations of motion derived in Chapter 2 for an N-rotor vehicle. The simulation for aerodynamic forces on the body of the vehicle are not covered in this script because their influence is negligible for hover and low-speed manoeuvres.

The equations of motion to be simulated for an N-rotor vehicle are non-linear ODE's (Ordinary Differenial Equations). Thus simulation can be thought of as "just" integrating forward the ODE's from a particular initial condition. Unless the ODE's can be analytically integrated, this requires numerical methods that approximate the continuous time integration. The most common technique used to numerically integrate forward an ODE is Runge-Kutta [3]. The accuracy of Runge-Kutta numerical integration is affected by the time step of integration and order of the time derivative approximation, where a shorter time step and higher order generally increase both the accuracy and the computational burden of the integration.

Most programming languages have a library that implements Runge-Kutta numerical integration techniques with various options, and many software packages exist that are dedicated to simulation of ODE's. The Simulink software produced by MathWorks is one such example and is chosen as the focus for this script due to it availability at many tertiary institutions. Simulink uses a Graphical User Interface (GUI) for entering the ODE and then compiles dedicated code that numerically integrates the ODE. By default, Simulink performs Runge-Kutta numerical integration of both fourth and fifth order, comparing the result to estimate the integration error and adapt the size of the integration time-step.

Section 3.1 provides all the details required to simulate an N-rotor vehicle using Simulink with sufficient accuracy for the controller design to be transferable to the real-world vehicle. An introduction to Simulink is provided in §3.1.1, followed in §3.1.2 by an explanation of the Simulink template that is provided for simulating an N-rotor vehicle. Entering the full N-rotor vehicle equations of motion is left as an exercise and §3.1.4 provides hints for how to get started on this exercise.

Important note for how to read this chapter

This is a "tutorial style" chapter in the sense that step-by-step instructions are provided and the reader is expected to complete each step before moving to the next step. Many of the steps are accompanied by a screen-shot, and for a reader that is unfamiliar with the simulation software it is useful to look at the screen-shots for performing each step correctly. However, the instructions are presented with multiple steps stated one-after-the-other, and the relevant screen-shots provided on the subsequent pages, hence some flicking back-and-forth is expected. The reason for presenting the material in this manner is that once the reader has developed basic competency with the simulation software, then it is more user-friendly to have multiple steps written down together.

3.1 Simulation with Simulink

This section assumes no prior knowledge or use of the Simulink software, it does however assume that the reader has a basic level of competency with MATLAB . For introductory MATLAB tutorials see the MathWorks website [7]. The following screen-shots and menu descriptions were produced for Simulink Release 2017b, though likely to be very similar for previous and future releases.

3.1.1 Introduction to Simulink

This introduction to Simulink will cover the basics of creating a simulation from scratch. We will use a pendulum as the motivating application and simulate the movement of the pendulum forward from a fixed initial condition. Figure 3.1 shows a schematic of the pendulum to be simulated, with the variables and quantities defined as follows:

θ	specifies the angular position of the pendulum in radians, with the convention that positive angle is measured as anti-clock-wise from the vertical-down position,
m	denotes the mass of the pendulum, measured in kilograms
g	denotes acceleration due to gravity, i.e., $9.81\,\mathrm{kg/s^2}$
l	denotes the length from the pivot to the centre of gravity of the pendulum, measured in metres
c	denotes the angular kinetic friction constant of proportionality, measured in units of Nms/radian.

The equations of motion of the pendulum are readily derived as:

$$\left(ml^2\right)\,\ddot{\theta} = -m\,g\,l\,\sin(\theta)\;-\;c\,\dot{\theta}\,, \tag{3.1}$$

where the quantity $\left(ml^2\right)$ on the left-hand-side is the mass moment of inertia of the pendulum, the first quantity on the right-hand-side is the torque due to the gravitational force, and the second quantity on the right-hand-side is the torque due to friction at the pivot that opposes the direction of rotation.

Opening a new Simulink model

Simulink is accessed through MATLAB , thus to launch Simulink you must first launch MATLAB . Simulink is launched by either clicking the “Simulink” button in the home ribbon, see Figure 3.2, or by entering the command `simulink` in the MATLAB “Command Window”. Simulink generally launches with a “Start Page”, click the “Blank Model” button to start a new Simulink model file, see Figure 3.3.

Accessing Simulink blocks from the “Library Browser”

Simulink is a graphical tool where a model is built by connecting various blocks, in other words it can be considered as a graphical programming software. The various blocks are accessed via the “Library Browser” that can be opened by clicking the appropriate button in the top bar of your Simulink model, see Figure 3.4. Alternatively, the “Library Browser” can be opened by entering the command `slLibraryBrowser` in the MATLAB Command Window.

The “Library Browser” is shown in Figure 3.5. The left-hand pane has a list of categories that are used to locate a particular block, with the blocks in the highlighted category being shown in the right-hand pane. The four blocks highlighted in Figure 3.5 are required to simulate the pendulum example.

A block is added to a model by either dragging-and-dropping from the “Library Browser” into the model file, or by right-clicking the block and selecting “Add block to model” from the pop-up menu.

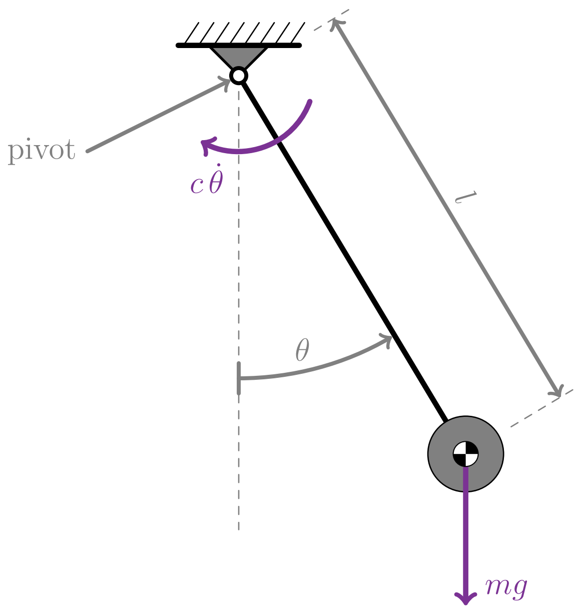


Figure 3.1: Schematic of the pendulum model that will simulated as part of the Simulink introduction. The arm of the pendulum is a massless rod of length l and with angular position denoted as θ. The forces acting on the pendulum are the gravity force mg and the friction force $c\dot{\theta}$

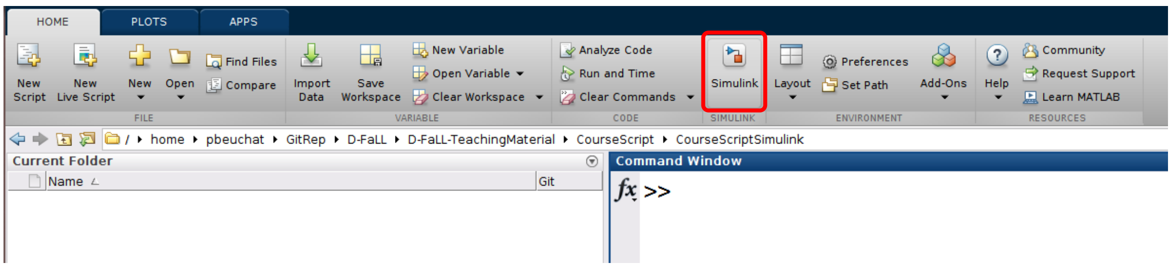


Figure 3.2: After opening MATLAB , click the “Simulink” button highlighted to launch Simulink. Alternatively, enter the command `simulink` in the MATLAB “Command Window”.

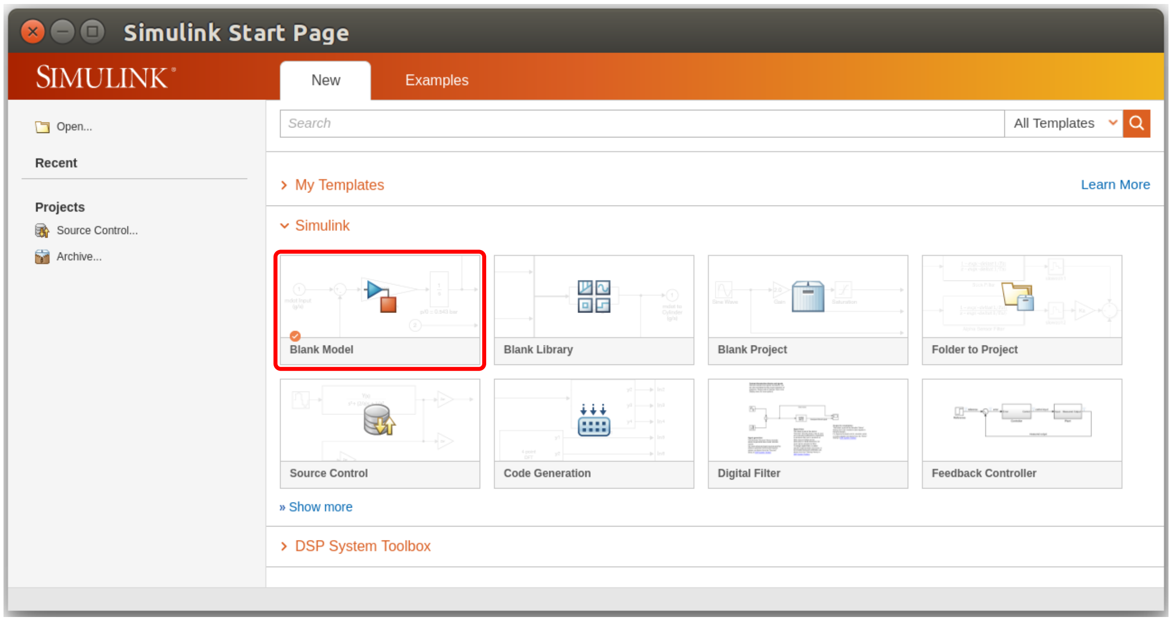


Figure 3.3: From the "Simulink Start Page", click the "Blank Model" button highlighted to start with a clean slate.

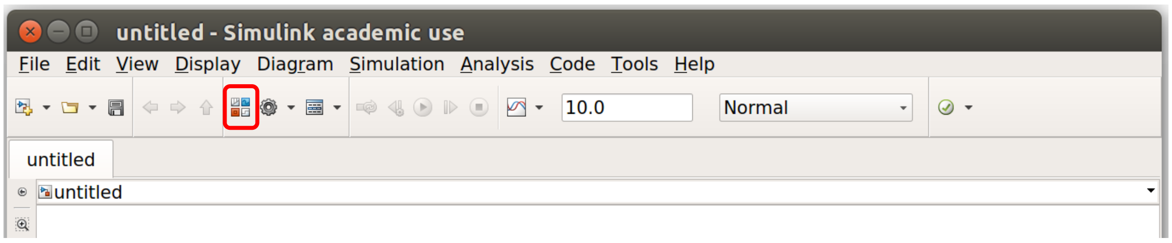

untitled - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
untitled
untitled

Figure 3.4: From the top bar of a Simulink model, the various blocks are accessed by clicking the highlighted button to open the "Library Browser". Alternatively, entering the command `slLibraryBrowser` in the MATLAB Command Window will also open the "Library Browser".

Build the pendulum equations of motion

The equation of motion for the pendulum, i.e., equation (3.1), shows that the angular acceleration, i.e., $\ddot{\theta}$, is equal to the sum of two terms that depend only on lower order derivatives of the angle, i.e., the terms depend on $\dot{\theta}$ and θ. Thus we start building the Simulink model by adding two "Integrator" blocks connected sequentially, see Figure 3.6 (top). The blocks are connected by clicking-and-dragging from the input or output port of a block. The connections can then be labeled by double-clicking on a connection line. Labeling the connections is not necessary but can assist with readability of the model. Reading from left-to-right on Figure 3.6 (top) we have that $\ddot{\theta}$ will be the input to the first integrator, $\dot{\theta}$ is thus the output of the first integrator, and θ is the output of the second integrator.

Next, we need to construct the equation for $\ddot{\theta}$, thus we add a "Trigonometric Function" block to compute $\sin(\theta)$, In addition, we add two "Gain" blocks to the model that will multiply $\dot{\theta}$ and $\sin(\theta)$ by the respective coefficient from equation (3.1). Figure 3.6 (bottom) shows how these blocks should be connected, to make an intersection along a connection line, hold down the *Ctrl* key while clicking-and-dragging from some point along the connection line. The "Trigonometric Function" block can be found under the "Math Operations" category. To avoid the connection lines becoming a mess it can be beneficial to flip the blocks. To flip a block, first right-click on the block, and then from the pop-up menu select "Rotate & Flip > Flip Block".

Next, we sum together the two terms by adding a "Sum" block using `--` for the "List of signs" and connecting the output of the "Gain" blocks to the inputs of the "Sum" block. The output of the "Sum" block thus represents $\ddot{\theta}$ once the gain parameters are correctly set. Figure 3.7 (top) shows how to connect the "Gain" and "Sum" blocks.

Finally, a quick way to analyse the results is to use a "Scope" block. This block can be useful for debugging why a particular value is not as expected, or for gaining quick insight into the system behaviour. As shown in Figure 3.7 (bottom), a "Scope" block is added with two input ports, one of which is connected to the θ signal and the other connected to the $\dot{\theta}$ signal.

Connecting parameters to a MATLAB script

Entering the parameter values for each block separately has two disadvantages: (i) it is cumbersome when multiple parameters need to be updated, especially on larger models, and (ii) longer expressions have poor readability. Instead, it is more convenient to use variable names for each parameter and assign these names using a MATLAB script.

When a Simulink model is compiled and run, any variables that exist in the MATLAB "Workspace" are available to be used within the Simulink model. To be more clear, Figure 3.8 shows a MATLAB "Workspace" with a variable named `example_parameter` that was assigned the value 2. Thus if a Simulink model is compiled when the MATLAB "Workspace" is like this, then any occurrence of the variable name `example_parameter` in a Simulink block will be replaced with the value 2.

A Simulink model of any appreciable size will contain many parameters, and many of these would be calculated from more fundamental parameters that describe the physical system being modeled. Thus the easiest way to put the necessary variables into the MATLAB "Workspace" is to use a `*.m` script that defines the variables necessary for the Simulink model and puts them into the MATLAB "Workspace". Figure 3.9 shows a code snippet that puts the pendulum paramaters into the MATLAB "Workspace", in this case the mass, length, gravitational acceleration, angular friction co-efficient, and initial angle.

A common pitfall of using a MATLAB script to define the parameters is that the user often forgets to run the script after changing a parameter value in the script. To overcome this, it is possible to specify in the Simulink model that the MATLAB script should be automatically run before the Simulink model is compiled. This is specified via the "Model Properties" of a Simulink model, which are accessed from settings menu in the top bar of the Simulink model window, see Figure 3.10.

In the "Model Properties" window, the "Model initialization function" is specified on "Callbacks" tab via the item "InitFnc", see Figure 3.11. A list of functions entered here will all automatically be run prior to compilation of the Simulink model. If a function cannot be found in the current working directory or on the MATLAB path, then an error message will appear and compilation will be cancelled. Thus, saving the code snippet in Figure 3.9 as a file named `simulink_intro_pendulum_paramters.m` and entering this as the "Model initialization function" will make the variables `m`, `l`, `g`, `c`, and `initial_theta` available for use in the Simulink model.

Setting the block parameters in the Simulink model

The parameters of any block can be are specified by double-clicking on a block, and then filling in the fields available for that block. The block parameters can also be accessed by right-clicking on a block and selecting "Block Parameters" from the pop-up menu.

The parameters to enter are shown in Figure 3.12, or equivalently:

- For the integrator from $\ddot{\theta}$ to $\dot{\theta}$, the "Initial condition" block parameter should be set to zero. This specifies that when the simulation starts the pendulum has zero angular velocity.
- For the integrator from $\dot{\theta}$ to θ, the "Initial condition" block parameter should be set to the variable named `initial_theta`. This specifies that when the simulation starts the pendulum is at the angle `initial_theta` radians.
- For the "Trigonometric Function" block, ensure that the "Function" parameter is set to be `sin`.
- For the "Gain" block connected to output of the trigonometric function, set the "Gain" parameter to `g/l`.
- For the "Gain" block connected to $\dot{\theta}$, set the "Gain" parameter to `c/(m*l^2)`.
- For the "Sum" block, set the "List of signs" to `--`.

Note: setting the initial condition parameter of an integrator block should always be carefully considered. This is especially true when the signal being integrated is a vector signal, in which case the initial condition should be specified as a vector so that Simulink knows the number of elements in the vector signal. For example, in the N-rotor simulation we will integrate vector signals that have three elements, thus `[0;0;0]` would be a valid initial condition parameter.

Run a simulation and view results

To compile and run a Simulink model, click the "Run" button in the top bar of the model window, as highlighted in Figure 3.13. This first generates and compiles dedicated code for performing a simulation of the model. If any errors are encountered during compilation, a pop-up window will appear with details about the source of the error. As with any programming language, the error messages can be more or less helpful depending on the cause of the error.

The duration of the simulation is specified by the field in the top bar of the model window, also highlighted in Figure 3.13 and is set as 10.0 seconds in that screen-shot.

To analyse the results of the simulation double-click on the "Scope" block. The pendulum behaviour simulated should be similar to that shown in Figure 3.14. By default the scope has a black background and no legend, the various options in the "Style..." settings allow you to customise the appearance to your liking.

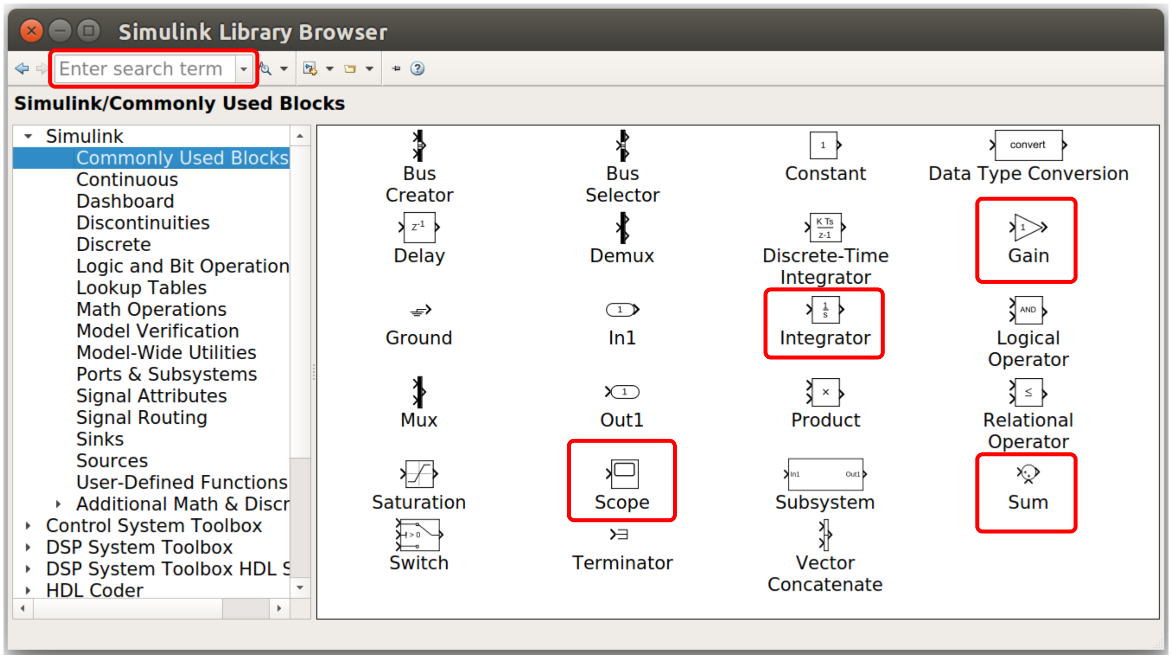
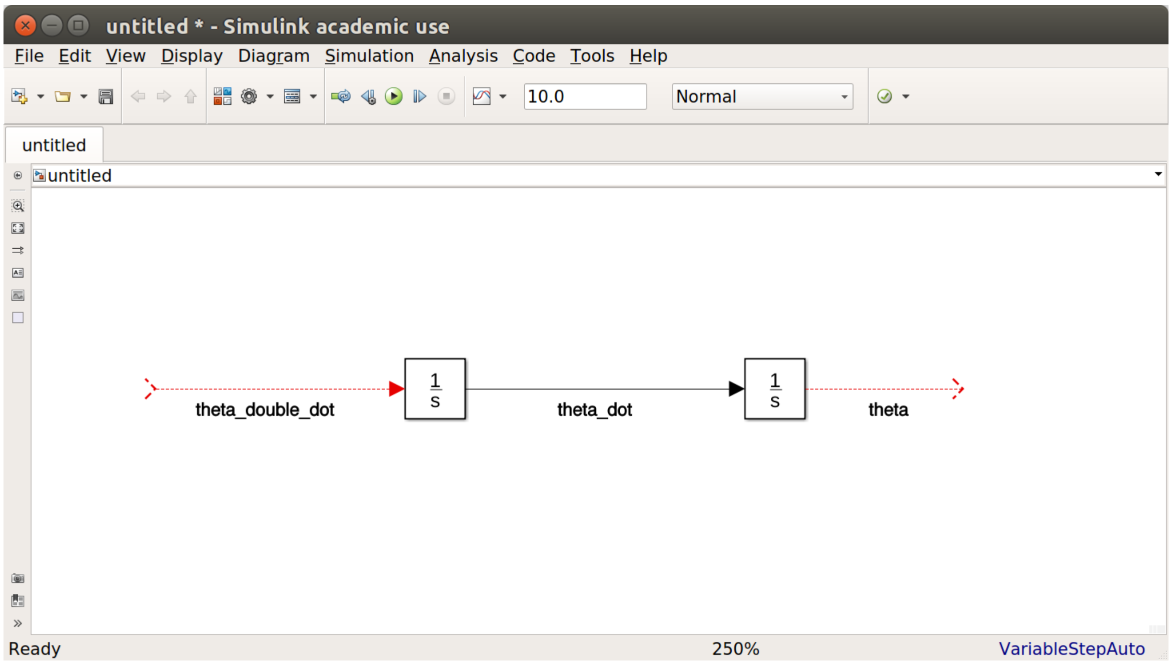


Figure 3.5: The “Library Browser” is where all the blocks are accessed and those required are dragged into the model file. The “Commonly Used Blocks” category shown in this figure does legitimately contain the most useful blocks. Those blocks highlighted are required to simulate the pendulum example, plus one additional block from the “Math Operations” category as explained later. The sub-categories of ”Simulink” accessed via the left pane of the window contain enough blocks to create complex models. The search bar, also highlighted, can be quite useful for finding a desired block.


untitled * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
untitled
untitled
1/s
1/s
theta_double_dot
theta_dot
theta
Ready
250%
VariableStepAuto

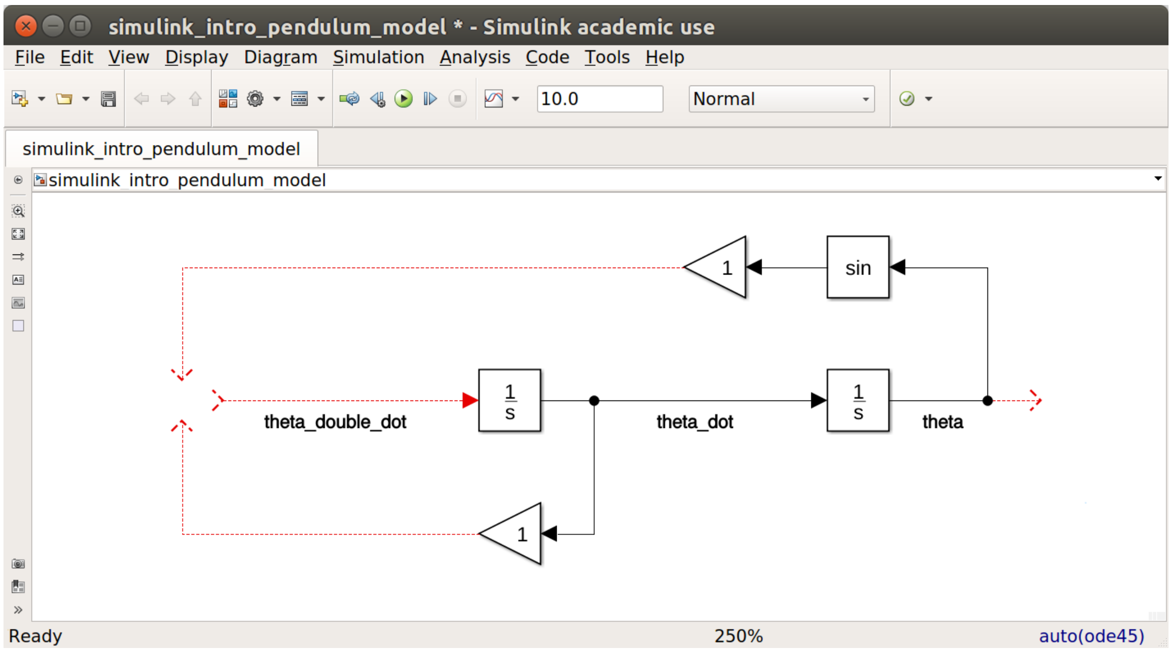
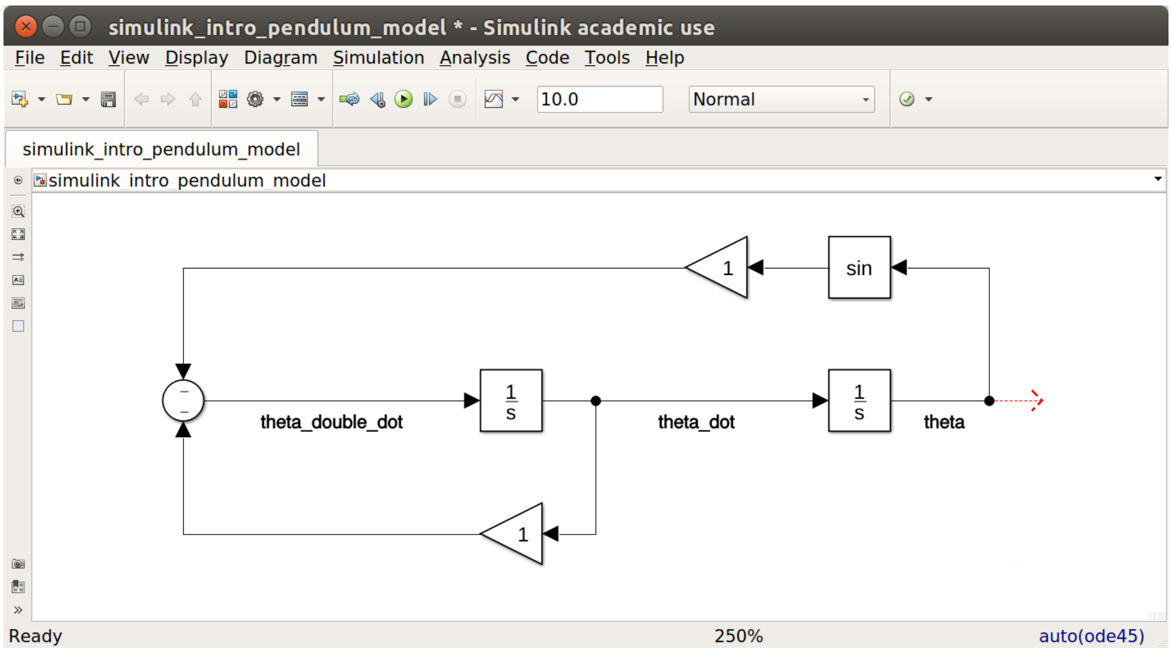

simulink_intro_pendulum_model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
simulink_intro_pendulum_model
simulink_intro_pendulum_model
1
sin
1/s
1/s
theta_double_dot
theta_dot
theta
1
Ready
250%
auto(ode45)

Figure 3.6: Top: add two “Integrator” blocks and connect them as shown by clicking-and-dragging from the input or output port of a block. The connections can be labeled by double-clicking at some point along the connection line. Bottom: add a “Trigonometric Function” and two “Gain” blocks and connect as shown. The “Trigonometric Function” block can be found in the “Math Operations” category and should be sinus by default. The gains will later be set to the appropriate coefficient from the equation of motion.


simulink_intro_pendulum_model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
simulink_intro_pendulum_model
simulink_intro_pendulum_model
1
sin
$\frac{1}{s}$
theta_double_dot
theta_dot
$\frac{1}{s}$
theta
1
Ready
250%
auto(ode45)

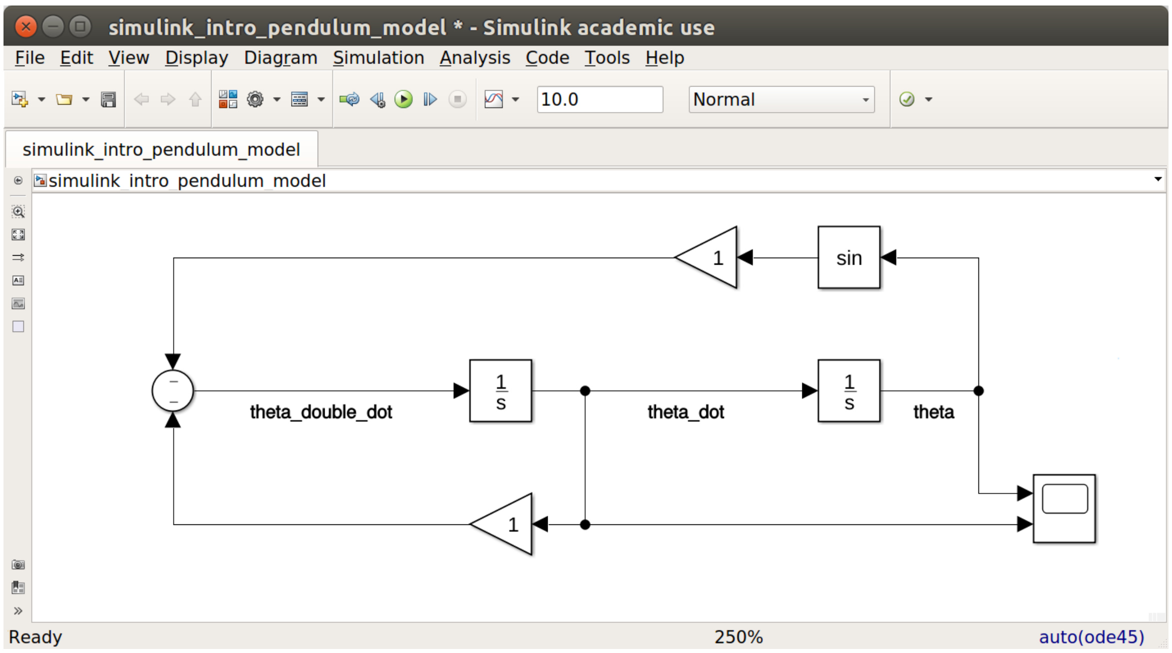
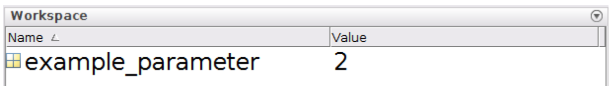

simulink_intro_pendulum_model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
simulink_intro_pendulum_model
simulink_intro_pendulum_model
1
sin
$\frac{1}{s}$
theta_double_dot
theta_dot
$\frac{1}{s}$
theta
1
Ready
250%
auto(ode45)

Figure 3.7: Top: add a “Sum” block as shown to construct the equation for $\ddot{\theta}$ by summing together the output of the two “Gain” blocks already present in the model. Bottom: add a “Scope” block as shown to analyse the θ and $\dot{\theta}$ behaviour of the pendulum.



Name	Value
example_parameter	2

Figure 3.8: Showing the MATLAB “Workspace” with a variable named `example_parameter` that was assigned the value 2. When a Simulink model is compiled and run, any use of the variable name `example_parameter` will be replaced with the value 2.

```
% Clear the "Workspace"
clear;
% Specify the mass of the pendulum, in units of [kilograms]
m = 1.5;
% Specify the length of the pendulum, in units of [meters]
l = 0.8;
% Specify the acceleration due to gravity, in units of [m/s^2]
g = 9.81;
% Specify the angular friction constant of proportionality,
% in units of [Nms/radian]
c = 0.5;
% Specify the initial angle of the pendulum, in units of [radians]
initial_theta = 60 * (pi / 180);
```

Figure 3.9: Code snippet that first clears the MATLAB "Workspace", and then puts the parameters of the pendulum into the "Workspace", i.e., `m`, `l`, `g`, `c`, and `initial_theta`. After running this script, these variable are available when a Simulink model is compiled and run.

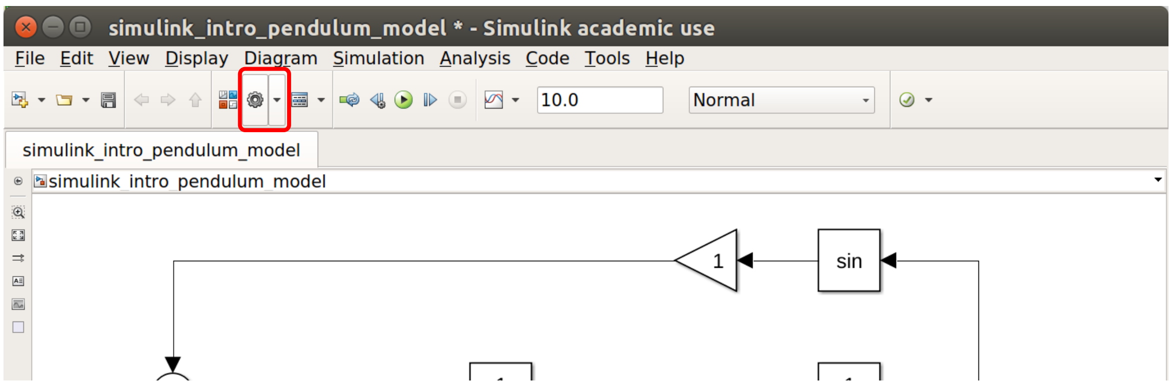
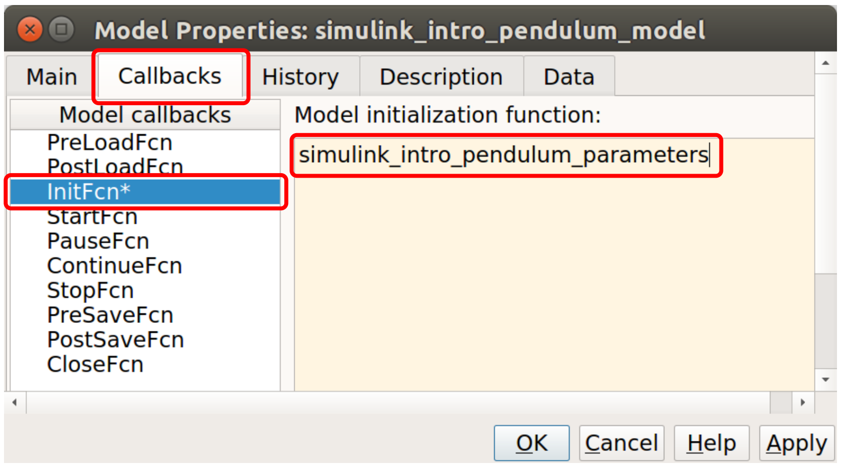

simulink_intro_pendulum_model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
simulink_intro_pendulum_model
simulink_intro_pendulum_model
1
sin

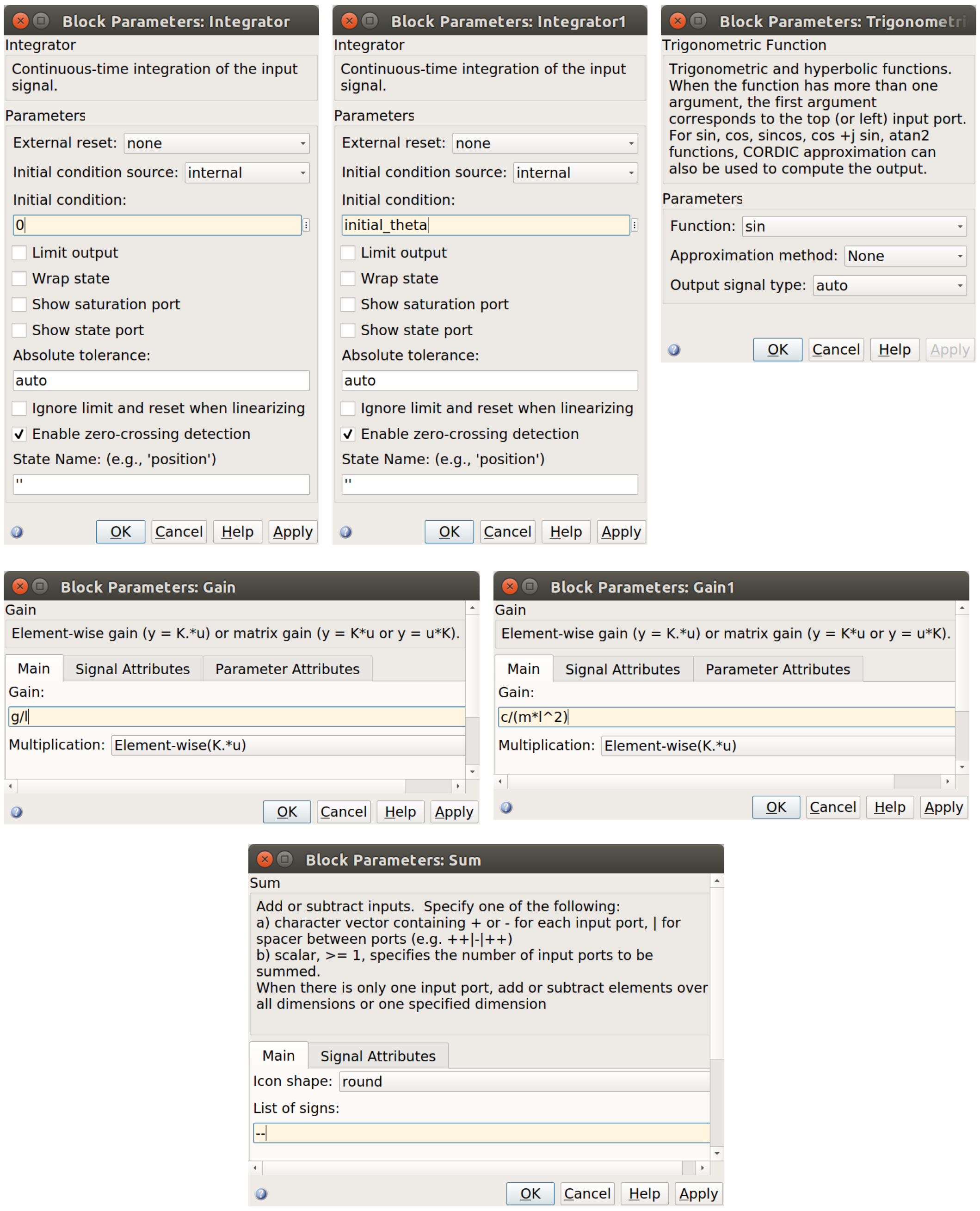
Figure 3.10: To access the "Model Properties" of a Simulink model, click on the arrow beside the settings icon, as highlighted, and from the pop-up menu that appears select the "Model Properties" item.



Model initialization function:

simulink_intro_pendulum_parameters

Figure 3.11: To specify a script to be automatically run prior to compilation of the Simulink model, change to the “Callbacks” tab, select the “InitFnc” menu item in the left-hand pane, and enter the name of the script in the right-hand pane. This screen-shot shows that the `simulink_intro_pendulum_paramters.m` script will be run prior to Simulink model compilation if it is on the MATLAB path. Note importantly that the “`.m`” extension is not included when specifying the model initialization function. If the model initialization function cannot be found in the current working directory or on the MATLAB path then an error message will appear and compilation will be cancelled.



Block Parameters: Integrator

Integrator

Continuous-time integration of the input signal.

Parameters

External reset: none

Initial condition source: internal

Initial condition:

0

☐ Limit output

☐ Wrap state

☐ Show saturation port

☐ Show state port

Absolute tolerance:

auto

☐ Ignore limit and reset when linearizing

✓ Enable zero-crossing detection

State Name: (e.g., 'position')

''

OK Cancel Help Apply

Block Parameters: Integrator1

Integrator

Continuous-time integration of the input signal.

Parameters

External reset: none

Initial condition source: internal

Initial condition:

initial_theta

☐ Limit output

☐ Wrap state

☐ Show saturation port

☐ Show state port

Absolute tolerance:

auto

☐ Ignore limit and reset when linearizing

✓ Enable zero-crossing detection

State Name: (e.g., 'position')

''

OK Cancel Help Apply

Block Parameters: Trigonometri

Trigonometric Function

Trigonometric and hyperbolic functions. When the function has more than one argument, the first argument corresponds to the top (or left) input port. For sin, cos, sincos, cos +j sin, atan2 functions, CORDIC approximation can also be used to compute the output.

Parameters

Function: sin

Approximation method: None

Output signal type: auto

OK Cancel Help Apply

Block Parameters: Gain

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Main | Signal Attributes | Parameter Attributes

Gain:

g/l

Multiplication: Element-wise(K.*u)

OK Cancel Help Apply

Block Parameters: Gain1

Gain

Element-wise gain (y = K.*u) or matrix gain (y = K*u or y = u*K).

Main | Signal Attributes | Parameter Attributes

Gain:

c/(m*l^2)

Multiplication: Element-wise(K.*u)

OK Cancel Help Apply

Block Parameters: Sum

Sum

Add or subtract inputs. Specify one of the following:
a) character vector containing + or - for each input port, | for spacer between ports (e.g. ++|-|++)
b) scalar, >= 1, specifies the number of input ports to be summed.
When there is only one input port, add or subtract elements over all dimensions or one specified dimension

Main | Signal Attributes

Icon shape: round

List of signs:

--

OK Cancel Help Apply

Figure 3.12: Screen-shots of the block parameters for each of the six blocks in the pendulum model (excluding the Scope). See the the list in the body of the script explaining why the block parameters are set to these values.

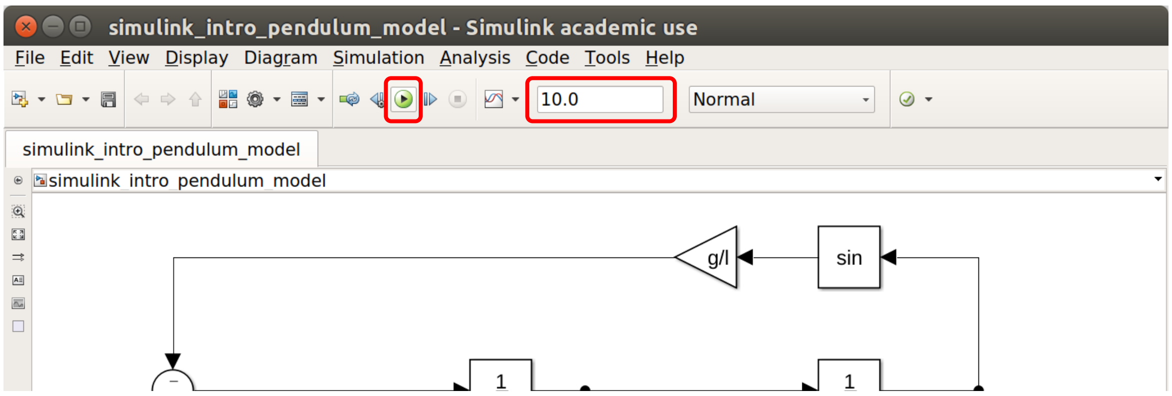

simulink_intro_pendulum_model - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10.0
Normal
simulink_intro_pendulum_model
simulink_intro_pendulum_model
g/l
sin
1
1

Figure 3.13: To compile and run a simulation of the Simulink model, click the highlighted button. The simulation is performed for the duration entered in the highlighted field, i.e., for 10 seconds in the case of this screen-shot. If errors arise during compilation, then a box will appear with information about the error(s) that occurred. As with all programming languages, sometimes the error message points directly to the cause of the error, and other times the error message can be rather misleading.

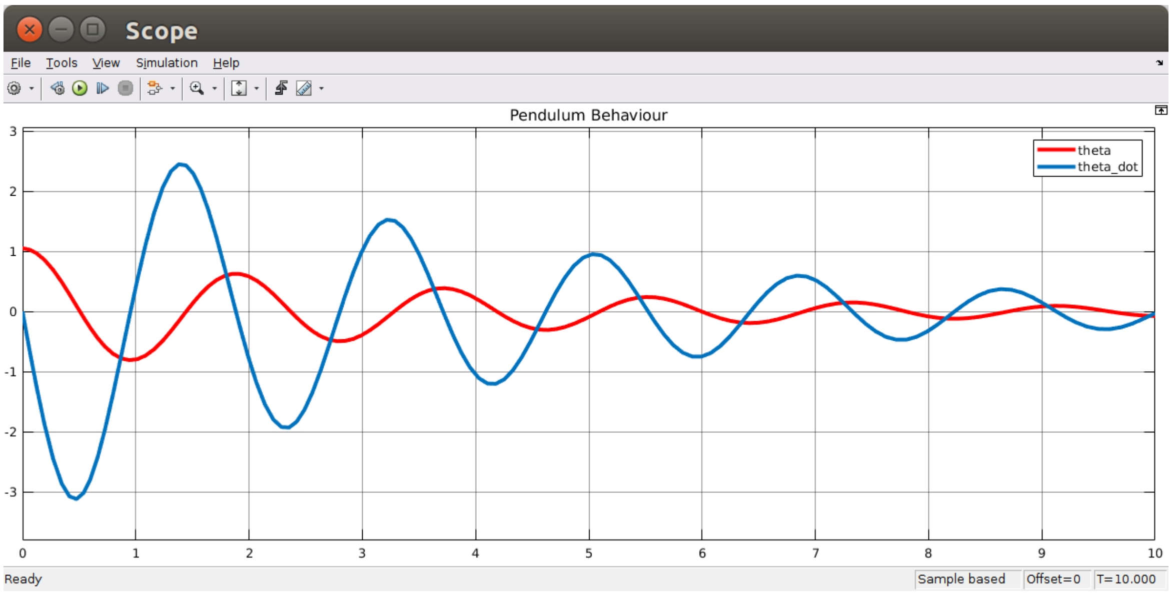


Figure 3.14: This “Scope” screen-shot shows the pendulum behaviour starting from an initial angle of $\theta = 60°$ ($\pi/3$ radians) and oscillating back and forth for 10 seconds, with the amplitude of oscillation dying out due to the angular kinetic friction term. By default, the scope has a black background, no legend, and different colours for the traces. The appearance can be changed via the “Style...” settings that are found under the “View” menu of the scope window.

3.1.2 Explanation of N-rotor Vehicle Template

To allow students to achieve the learning objective of simulating a general N-rotor vehicle in a more timely manner, a Simulink template is provided with the dynamics left as empty and a "black-box" controller whose design details are not revealed. This template consist of two files:

i) a Simulink model file named `exercise01_simulation_model_template.slx`
ii) a MATLAB script file named `exercise01_simulation_parameters.m`

The Simulink model has the MATLAB script set as the "Model Initialization Function", in the same way that this is described for the pendulum tutorial of §3.1.1 above and shown in Figure 3.11. Hence, if you change the file name of the MATLAB script provided, then you will need to update the "Model Initialization Function" setting accordingly.

When the Simulink model is first opened it should appear similar to Figure 3.15. The model has four boxes connected together, in Simulink terminology these boxes are called "sub-systems" and the details of what is modelled within a sub-system is viewed and edited by double-clicking on the respective box. From left-to-right, the four sub-systems are:

i) **Reference generator.** This sub-system outputs a vector signal with four elements which specify the desired p_x, p_y, p_z position and α yaw angle of the N-rotor vehicle. This sub-system contains blocks to generate step-change, sinusoidal, and saw-tooth reference signals.
ii) **Controller.** This sub-system takes as input signals the desired position and yaw angle from the "Reference generator" and the current full state vector of the N-rotor vehicle, and outputs a vector signal with N elements that is the desired thrust in Newtons from each rotor of the vehicle. The convention used in this model is that the "full state" is a vector signal with 12 elements that are stacked as follows:

$$\texttt{full state} = \begin{bmatrix} \vec{p} \\ \dot{\vec{p}} \\ \vec{\psi} \\ \dot{\vec{\psi}} \end{bmatrix}$$

iii) **N-rotor vehicle model.** This sub-system is empty and it is left as the exercise for the student to fill in this block with the equations of motion of the N-rotor vehicle. Section 3.1.4 provides hints to get started with this exercise.
iv) **Plotting.** This sub-system combines and scales the signals so that the behaviour of the N-rotor vehicle can be concisely observed on a single scope.

The MATLAB script that accompanies this Simulink template is extensively commented and those comments are the best source of explanation. In summary, the script places the necessary variables into the workspace that are needed for the four sub-systems described above. It also defines variables that describe the layout of the N-rotor vehicle and the initial condition of the full state. To complete the exercise of adding the N-rotor vehicle equations of motion to the Simulink model, it is useful to define extra variables in this MATLAB script. The feedback gain of a default controller are hard-coded in the MATLAB script both for continuous-time and discrete-time control, hence the user should be careful to choose the controller gains appropriate to the simulation being run.

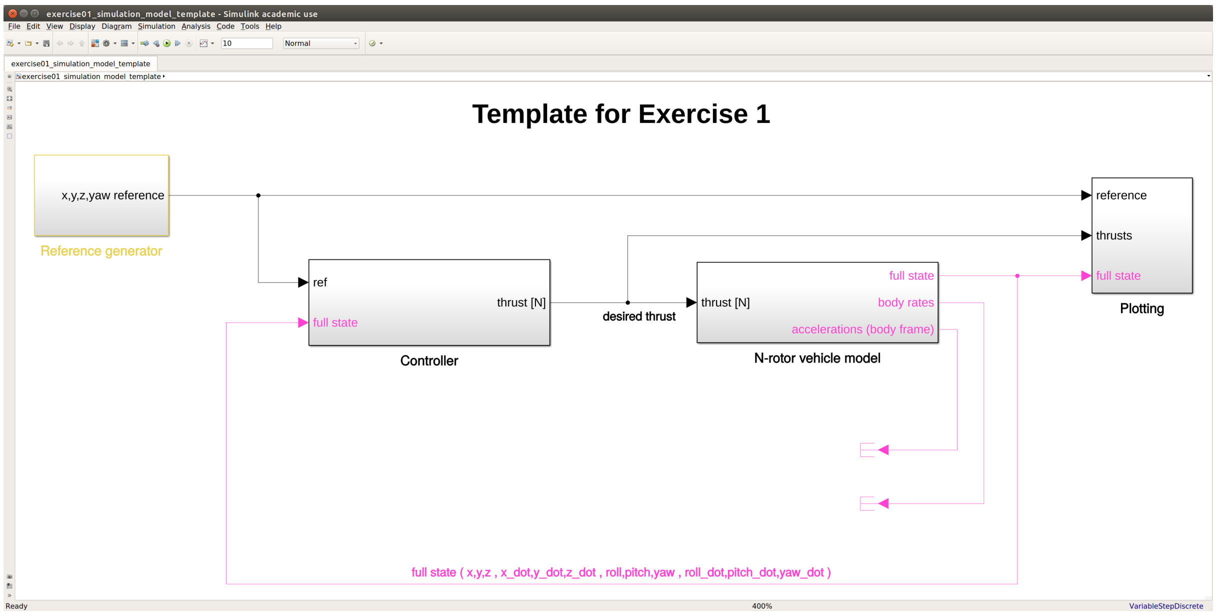

exercise01_simulation_model_template - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10
Normal
exercise01_simulation_model_template
exercise01_simulation_model_template
Template for Exercise 1
x,y,z,yaw reference
Reference generator
ref
full state
thrust [N]
Controller
desired thrust
thrust [N]
full state
body rates
accelerations (body frame)
N-rotor vehicle model
reference
thrusts
full state
Plotting
full state (x,y,z , x_dot,y_dot,z_dot , roll,pitch,yaw , roll_dot,pitch_dot,yaw_dot)
Ready
400%
VariableStepDiscrete

Figure 3.15: Screen-shot of the `exercise01_simulation_model_template.slx` Simulnk model when it is first opened. From left-to-right, the four main boxes are labelled as: Reference Generator, Controller, N-rotor vehicle model, and Plotting. In Simulink terminology these boxes are called "sub-systems" and internal construction of a sub-system is interrogated by double-clicking on the respective box. The colour coding of sub-systems and connection lines is automatically applied by Simulink when the model is compiled and run. Black means that a signal is continuous, pink means that a signal is constant, and yellow means that the sub-system contains a mixture of continuous and constant signals.

3.1.3 Hints for Simulating an N-rotor Vehicle "all-in-one-block"

This section provides hints for building the equations of motion "all-in-one-block". The result is a cleaner model that is easier to debug. In contrast, the hints given in Section 3.1.4 offer the benefit that by building the equations of motion "block-by-block" you gain more insight about how information propagates through the model. The disadvantage of building the equations of motion "block-by-block" is that many different blocks must be connected and it can be difficult to locate errors that may occur.

Start by adding an "Integrator" block that goes from the time derivative of the `full state` to the `full state` itself, and label the connection lines to assist with readability, see Figure 3.16 (top). Set the initial condition property of the "Integrator" block to be a column vector of length 12. Next, add a "MATLAB Function" block, found in the "User-Defined Functions" category. This block allows the use of normal MATLAB syntax for computing the block outputs from the block inputs. All the equations of motions will be computed inside this one block. To use a "MATLAB Function" block follow these steps:

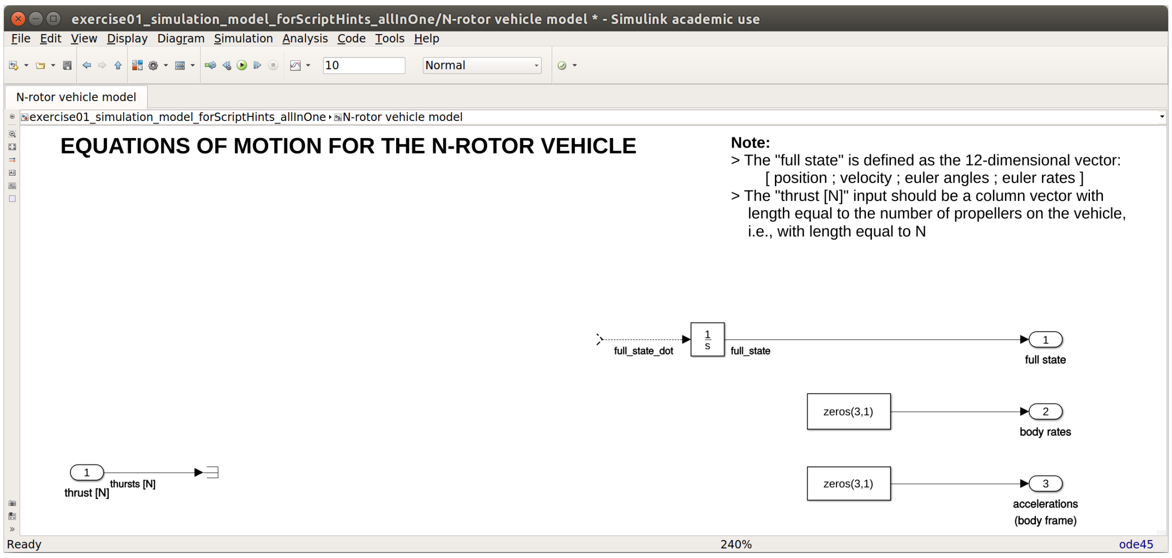
- Add a "MATLAB Function" block to the model,
- Double-click on the added "MATLAB Function" block to edit the script that it runs,
- Enter the code snippet shown in Figure 3.16 (middle). This code snippet is just a place-holder so that the block inputs and outputs are labelled as desired.
- Save the script and return to the Simulink model, the "MATLAB Function" block will now appear with 7 inputs and 1 output labelled accordingly. Connect the "Integrator" block, "MATLAB Function" block, and `thrusts[N]` input as shown in Figure 3.16 (bottom).

To connect the various parameter values from the MATLAB workspace into the Simulink model, use a separate "Constant" block for each parameter (5 in total). The required parameters are:

`g`	Acceleration due to gravity,
`nrotor_vehicle_mass_true`	mass of the vehicle,
`nrotor_vehicle_inertia_true`	the mass moment of inertia matrix,
`nrotor_vehicle_inertia_true_inverse`	the inverse for convenience,
`nrotor_vehicle_layout_true`	the propeller locations and thrust-to-torque ratios.

Note that the "`_true`" used in the variable names is to indicate that this parameters is used to specify a property of the actual N-rotor vehicle being simulated. This naming is used to provide a clear distinction from the parameters that are used in the controller, whose variable names contain "`_for_controller`". This separation between "`_true`" and "`_for_controller`" parameters allows for simulating the effect of a mismatch between the properties of the actual system, and those used when designing and implementing the controller.

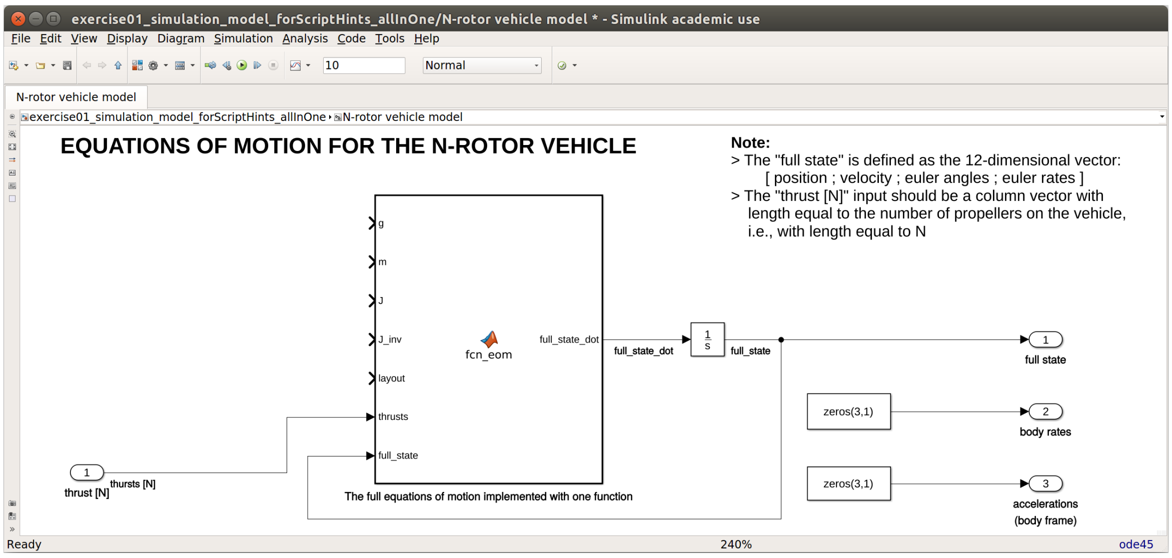
Connect the "Constant" blocks to the "MATLAB Function" block as as shown in Figure 3.17 (top). To simulate the N-rotor vehicle you need to complete the necessary calculations in the script of the "MATLAB Function" block. Figure 3.17 (bottom) provides some skeleton code to get you started. Once completed correctly the scope will show a behaviour similar to Figure 3.18 when the parameters as per the `exercise01_simulation_parameters.m` script provided.


exercise01_simulation_model_forScriptHints_allInOne/N-rotor vehicle model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10
Normal
N-rotor vehicle model
exercise01_simulation_model_forScriptHints_allInOne ▸ N-rotor vehicle model
full_state_dot
1/s
full_state
1 full state
zeros(3,1)
2 body rates
zeros(3,1)
3 accelerations (body frame)
1 thrust [N]
thursts [N]
Ready
240%
ode45

EQUATIONS OF MOTION FOR THE N-ROTOR VEHICLE

Note:
> The "full state" is defined as the 12-dimensional vector:
[position ; velocity ; euler angles ; euler rates]
> The "thrust [N]" input should be a column vector with length equal to the number of propellers on the vehicle, i.e., with length equal to N

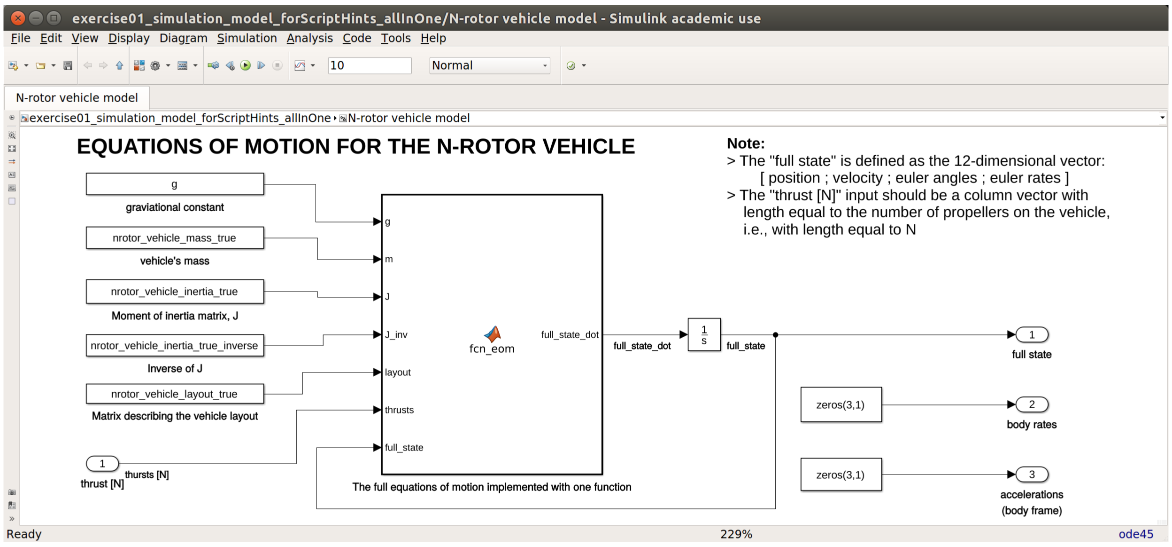
```
function full_state_dot = fcn_eom(g,m,J,J_inv,layout,thrusts,full_state)
% The "full_state_dot" return argument is a column vector with 12 elements
full_state_dot = zeros(12,1)
end
```


exercise01_simulation_model_forScriptHints_allInOne/N-rotor vehicle model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10
Normal
N-rotor vehicle model
exercise01_simulation_model_forScriptHints_allInOne ▸ N-rotor vehicle model
g
m
J
J_inv
layout
thrusts
full_state
fcn_eom
full_state_dot
full_state_dot
1/s
full_state
1 full state
zeros(3,1)
2 body rates
zeros(3,1)
3 accelerations (body frame)
1 thrust [N]
thursts [N]
The full equations of motion implemented with one function
Ready
240%
ode45

EQUATIONS OF MOTION FOR THE N-ROTOR VEHICLE

Note:
> The "full state" is defined as the 12-dimensional vector:
[position ; velocity ; euler angles ; euler rates]
> The "thrust [N]" input should be a column vector with length equal to the number of propellers on the vehicle, i.e., with length equal to N

Figure 3.16: Top: add an "Integrator" that goes from `full_state_dot` to the `full_state`. Middle: the place-holder code snippet to enter into the "MATLAB Function" block, first add the block into the model, then double-click on the block to enter this code snippet. Bottom: Showing how to connect the "MATLAB Function" block to the "Integrator" block, and to the `thrusts[N]` input of this "N-rotor vehicle model" sub-system.


exercise01_simulation_model_forScriptHints_allInOne/N-rotor vehicle model - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10
Normal
N-rotor vehicle model
exercise01_simulation_model_forScriptHints_allInOne › N-rotor vehicle model
EQUATIONS OF MOTION FOR THE N-ROTOR VEHICLE
g
graviational constant
nrotor_vehicle_mass_true
vehicle's mass
nrotor_vehicle_inertia_true
Moment of inertia matrix, J
nrotor_vehicle_inertia_true_inverse
Inverse of J
nrotor_vehicle_layout_true
Matrix describing the vehicle layout
1
thrust [N]
thursts [N]
g
m
J
J_inv
layout
thrusts
full_state
fcn_eom
full_state_dot
The full equations of motion implemented with one function
full_state_dot
1/s
full_state
1
full state
zeros(3,1)
2
body rates
zeros(3,1)
3
accelerations (body frame)
Ready
229%
ode45

Note:
> The "full state" is defined as the 12-dimensional vector:
[position ; velocity ; euler angles ; euler rates]
> The "thrust [N]" input should be a column vector with length equal to the number of propellers on the vehicle, i.e., with length equal to N

```
function full_state_dot = fcn_eom(g,m,J,J_inv,layout,thrusts,full_state)
% Extract the components of the full state
current_position     = full_state(1:3);
current_position_dot = full_state(4:6);
current_psi          = full_state(7:9);
current_psi_dot      = full_state(10:12);
% Extract each Euler angles
roll   =  current_psi(1);  % Roll
pitch  =  current_psi(2);  % Pitch
yaw    =  current_psi(3);  % Yaw
% Extract each Euler anglular rates
roll_dot   =  current_psi_dot(1);  % Roll  angular velocity
pitch_dot  =  current_psi_dot(2);  % Pitch angular velocity
yaw_dot    =  current_psi_dot(3);  % Yaw   angular velocity
% PERFORM THE EQUATION OF MOTION COMPUTATIONS HERE
% Put together the "full_state_dot" return variable
full_state_dot = [...
    current_position_dot ;...
    zeros(3,1)           ;...
    current_psi_dot      ;...
    zeros(3,1)            ...
];
end
```

Figure 3.17: Top: Use a separate "Constant" block to get each parameter required from the MATLAB workspace and connect to the "MATLAB Function" block as shown. Bottom: skeleton code to get you started with implementing the equations of motion in the "MATLAB Function" block.

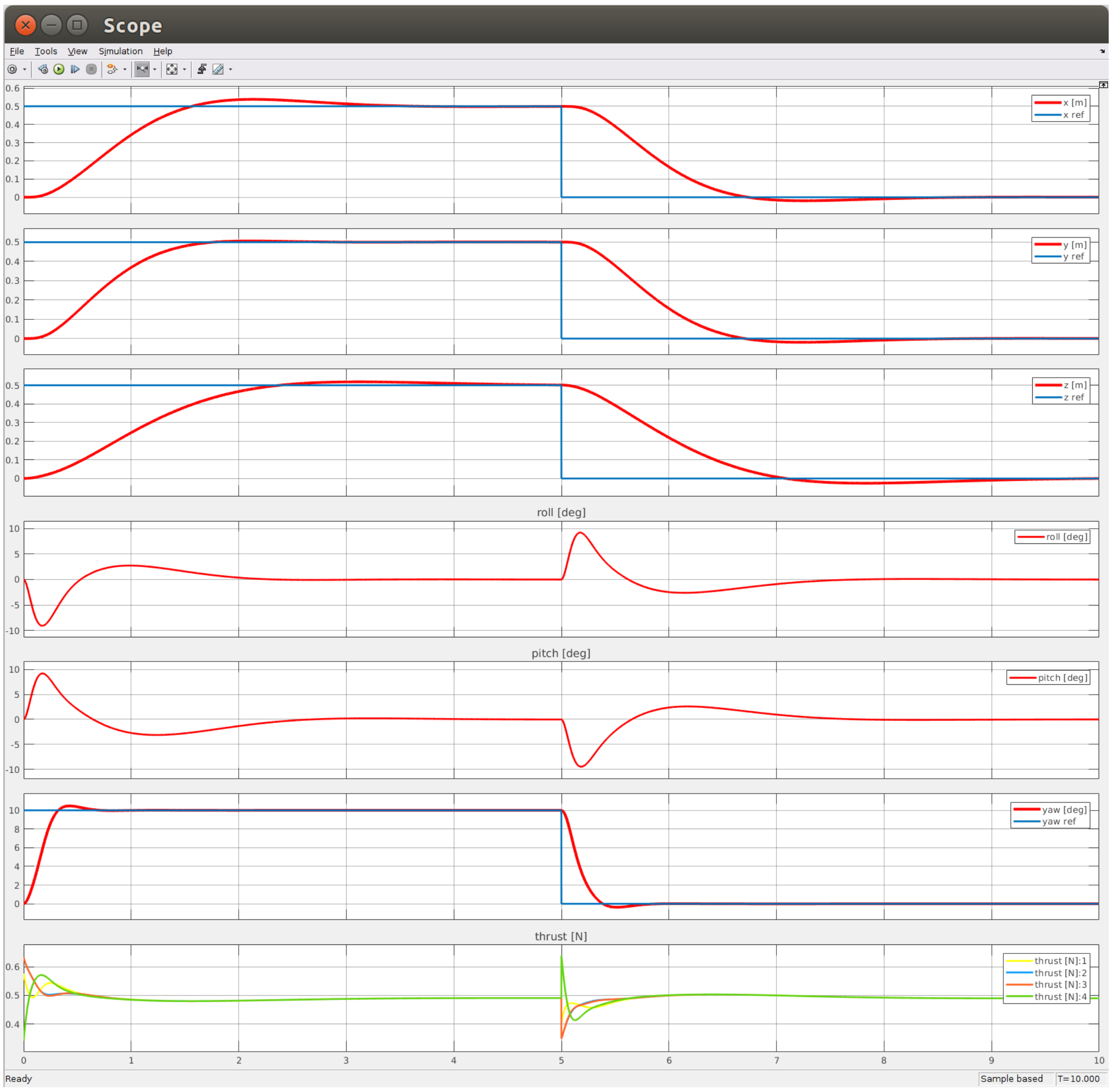


Figure 3.18: Showing the expected behaviour as viewed on the scope of the template Simulink model provided, with the step reference and controller gains as per the parameter file provided.

3.1.4 Hints for Simulating an N-rotor Vehicle "block-by-block"

To start with modelling the N-rotor vehicle equations of motion, first look at the equations of motion for the body rates $\vec{\omega}$, repeated here for convenience:

$$\dot{\vec{\omega}} \;=\; \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \;=\; J^{-1} \left(\begin{bmatrix} \sum_{i=1}^{N} & f_i\, y_i \\ \sum_{i=1}^{N} & -f_i\, x_i \\ \sum_{i=1}^{N} & f_i\, c_i \end{bmatrix} \;-\; \left(\vec{\omega} \,\times\, J\,\vec{\omega}\right) \right) . \tag{3.2}$$

We observe that the body accelerations $\dot{\vec{\omega}}$ only depend on the lower order derivative $\vec{\omega}$, the inputs to the system f_i, and parameters that describe the N-rotor vehicle, namely J, x_i, y_i, and c_i. Hence we can build this equation of motion in a similar way to the introduction tutorial of §3.1.1 above.

Start by adding an "Integrator" block that goes from $\dot{\vec{\omega}}$ to $\vec{\omega}$ and label the connection lines to assist with readability, see Figure 3.19 (top). To construct the $\vec{\omega} \times J\,\vec{\omega}$ term:

- use a "Constant" block to get the `nrotor_vehicle_inertia` variable from the MATLAB workspace into the Simulink model,
- use a "Product" block to perform the multiplication $J\,\vec{\omega}$, ensuring that the "Multiplication" parameter of the block is set to "Matrix(*)" so that the operation is actually matrix multiplication and not element-wise multiplication. It is also important that J is the first input to the "Product" block and $\vec{\omega}$ is the second input so that the multiplication is performed in the correct order,
- use a "Cross Product" block to compute the full term $\vec{\omega} \times J\,\vec{\omega}$. This block is best found in the Library Browser by using the search bar. It is important that $\vec{\omega}$ is the first input to the block, and that $J\,\vec{\omega}$ is the second input.

Figure 3.19 (bottom) shows how these blocks should be connected. Next, to build the term:

$$\begin{bmatrix} \sum_{i=1}^{N} & f_i\, y_i \\ \sum_{i=1}^{N} & -f_i\, x_i \\ \sum_{i=1}^{N} & f_i\, c_i \end{bmatrix}$$

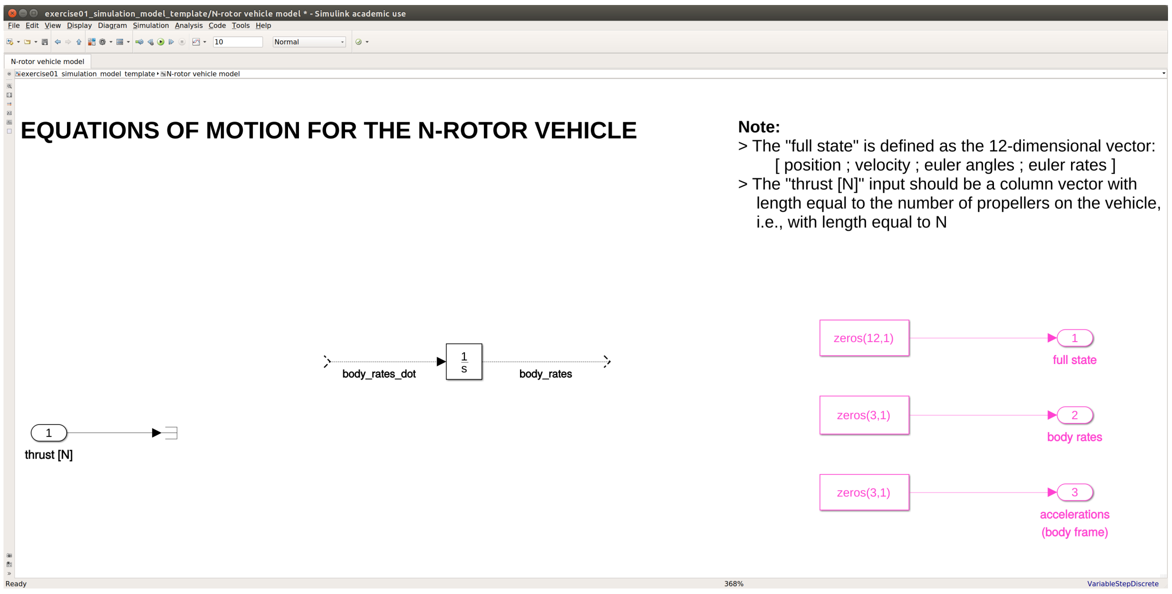
there are multiple options that yield the same result. One of the neatest and more readable options is to use a "MATLAB Function" block that is found in the "User-Defined Functions" category. This block allows the use of normal MATLAB syntax for computing the block outputs from the block inputs. To use a "MATLAB Function" block follow these steps:

- Add a "MATLAB Function" block to the model,
- Double-click on the added "MATLAB Function" block to edit the script that it runs,
- Enter the code snippet shown in Figure 3.20 (top). This code performs the necessary calculations and returns the variable `body_torques` as a vector with 3 elements,
- Save the script and return to the Simulink model, the "MATLAB Function" block will now appear with two inputs labelled `layout` and `thrusts` and one output labelled `body_torques`, connect these as shown in 3.20 (bottom).

Now the two terms can be summed together and multiplied by J^{-1} to compute the $\dot{\vec{\omega}}$ signal that is the input to the integrator, see Figure 3.21 (top). Finally, to keep you Simulink model neat and readable, it can be beneficial to combine a group of blocks into a sub-system. To do this:

- Highlight all the block to be combined into a sub-system, i.e., all the blocks shown in 3.21 (top),
- Right-click on any part of the selection and from the pop-up menu choose "Create Subsystem from Selection". The result will be similar to Figure 3.21 (bottom),
- When creating a sub-system, any connection line entering or exiting the selection is automatically made as an input or output respectively of the sub-system block,
- To add additional inputs to or outputs from the sub-system, use the "In1" and "Out1" blocks from the Library Browser.

It is left as an exercise to complete building the equations of motion for the N-rotor vehicle model. The steps will be similar to those explained in these hints, and once completed correctly the scope will show a behaviour similar to Figure 3.18 when the parameters as per the `exercise01_simulation_parameters.m` script provided.


exercise01_simulation_model_template/N-rotor vehicle model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
N-rotor vehicle model
exercise01_simulation_model_template ▸ N-rotor vehicle model
body_rates_dot
1/s
body_rates
1
thrust [N]
zeros(12,1)
1
full state
zeros(3,1)
2
body rates
zeros(3,1)
3
accelerations (body frame)
Ready
368%
VariableStepDiscrete

EQUATIONS OF MOTION FOR THE N-ROTOR VEHICLE

Note:
> The "full state" is defined as the 12-dimensional vector:
[position ; velocity ; euler angles ; euler rates]
> The "thrust [N]" input should be a column vector with length equal to the number of propellers on the vehicle, i.e., with length equal to N

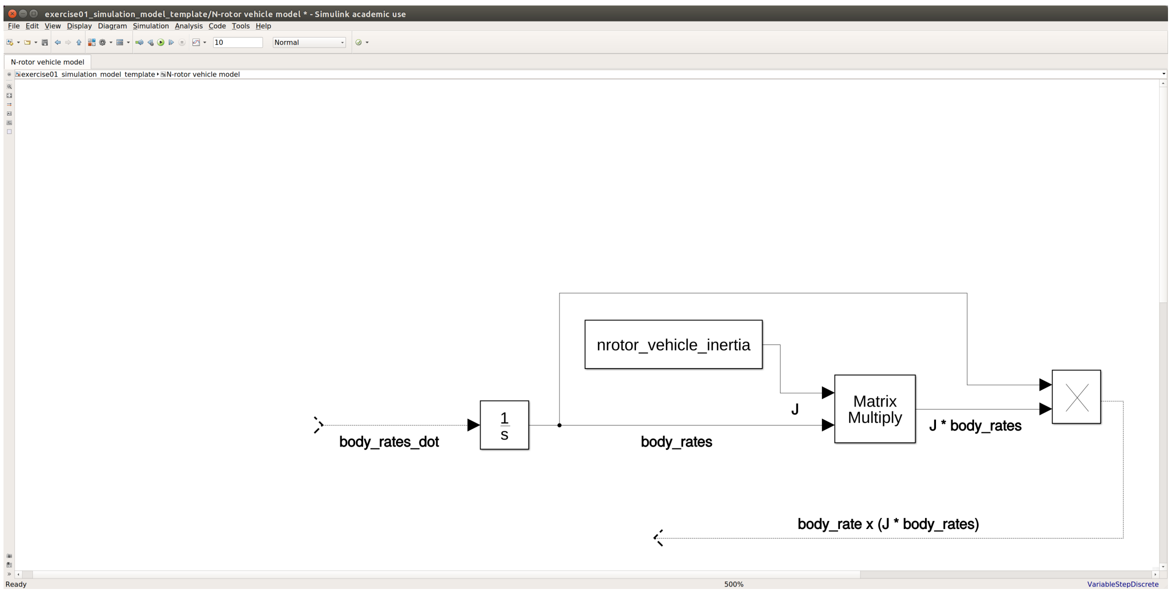

exercise01_simulation_model_template/N-rotor vehicle model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
N-rotor vehicle model
exercise01_simulation_model_template ▸ N-rotor vehicle model
nrotor_vehicle_inertia
J
Matrix Multiply
J * body_rates
body_rates_dot
1/s
body_rates
body_rate x (J * body_rates)
Ready
500%
VariableStepDiscrete

Figure 3.19: Top: add an "Integrator" that goes from $\dot{\vec{\omega}}$ to $\vec{\omega}$. Bottom: build the term $\vec{\omega} \times J\vec{\omega}$ by adding the blocks as described and connecting as shown, the names of the blocks used are: "Constant" block, "Product" block, and "Cross Product" block.

```
function body_torques = fcn(layout,thrusts)
% The "body_torques" return argument is a vector with 3 elements
body_torques = [...
layout(2,:) * thrusts ;...
-layout(1,:) * thrusts ;...
layout(3,:) * thrusts  ...
];
end
```

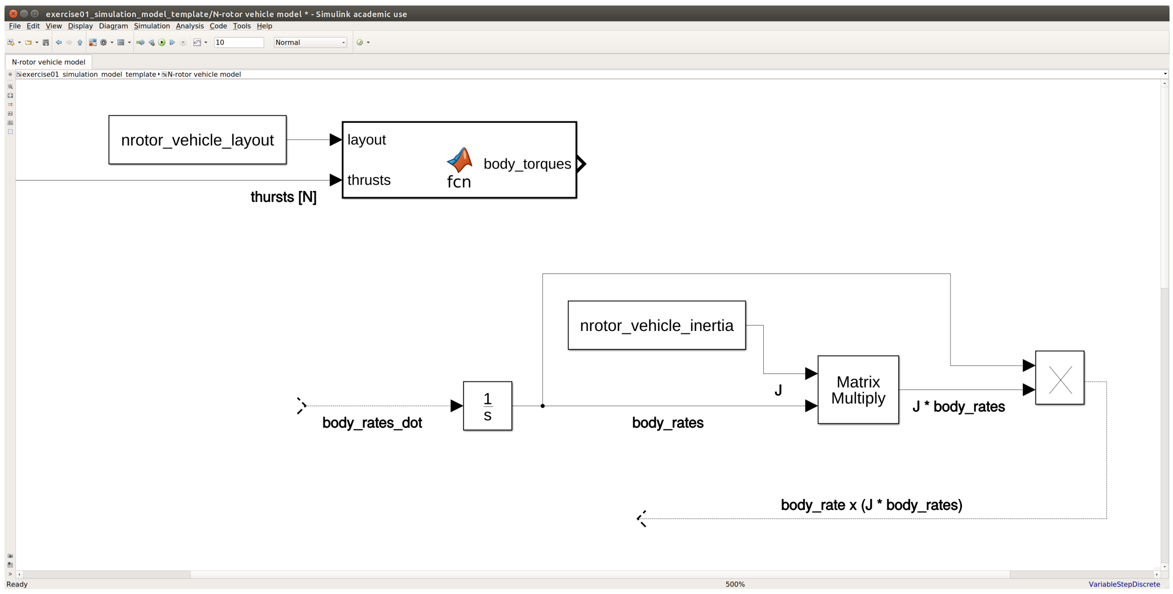
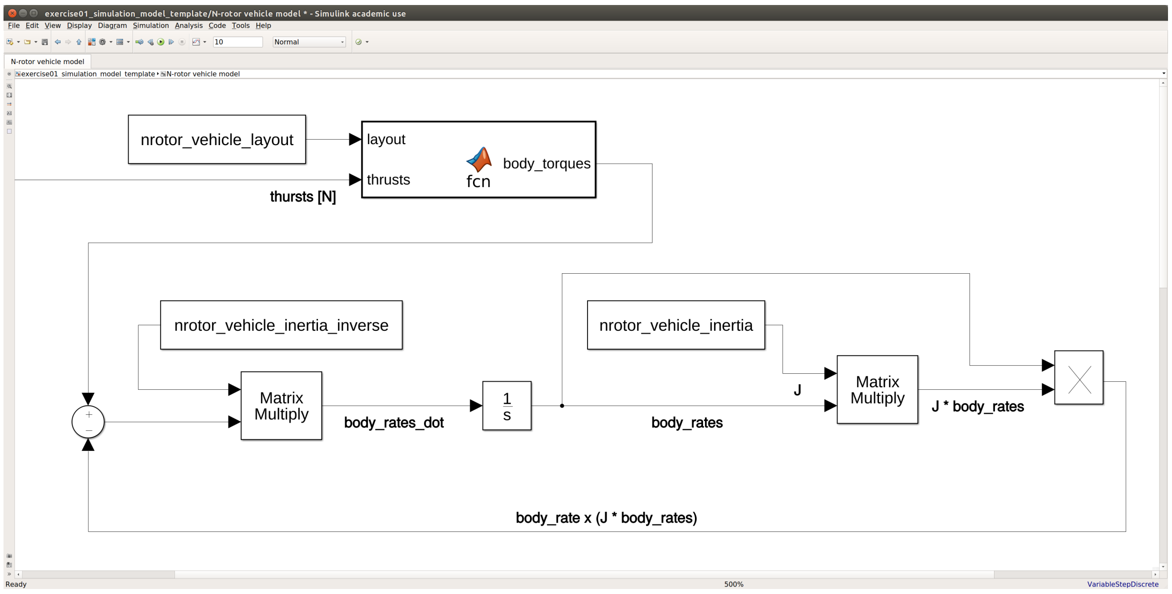

exercise01_simulation_model_template/N-rotor vehicle model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
10
Normal
N-rotor vehicle model
exercise01_simulation_model_template ▸ N-rotor vehicle model
nrotor_vehicle_layout
layout
thrusts
fcn
body_torques
thursts [N]
nrotor_vehicle_inertia
J
Matrix Multiply
J * body_rates
1/s
body_rates_dot
body_rates
body_rate x (J * body_rates)
Ready
500%
VariableStepDiscrete

Figure 3.20: Top: the code snippet to enter into the "MATLAB Function" block, first add the block into the model, then double-click on the block to enter this code snippet. Bottom: Showing how to connect the "MATLAB Function" block, use a "Constant" block to get the `nrotor_vehicle_layout` variable from the MATLAB workspace, and connect the `thrusts [N]` from the input of this "*N*-rotor vehicle model" sub-system.


exercise01_simulation_model_template/N-rotor vehicle model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
nrotor_vehicle_layout
layout
thrusts
fcn
body_torques
thursts [N]
nrotor_vehicle_inertia_inverse
nrotor_vehicle_inertia
Matrix Multiply
body_rates_dot
1/s
body_rates
J
Matrix Multiply
J * body_rates
body_rate x (J * body_rates)
Ready
500%
VariableStepDiscrete

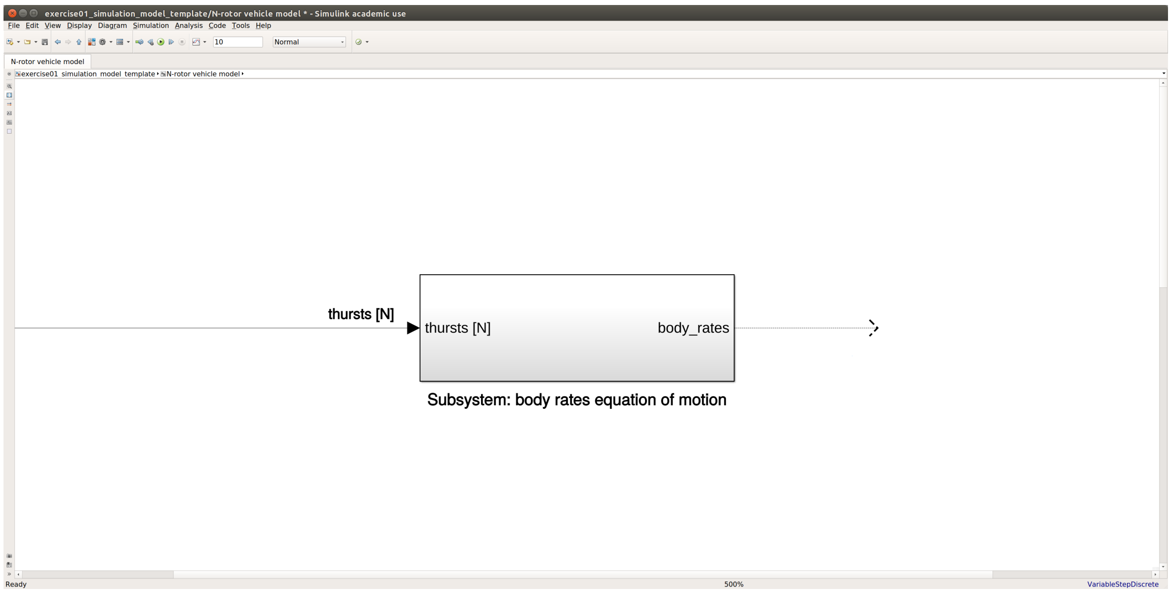

exercise01_simulation_model_template/N-rotor vehicle model * - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
thursts [N]
thursts [N]
body_rates
Subsystem: body rates equation of motion
Ready
500%
VariableStepDiscrete

Figure 3.21: Top: add a “Sum” block to respectively add and subtract the terms together, then add a “Product” block with the “Multiplication” parameter set to “Matrix(*)” and a “Constant” block to get the `nrotor_vehicle_inertia_inverse` variable from the MATLAB workspace. Bottom: the collection of blocks from the top figure were combined into a sub-system by first selecting all the blocks and then choosing “Create Subsystem from Selection” from the right-click pop-up menu. Using sub-systems keeps the Simulink model neat and readable.

Chapter 4

Controller Design

In Chapter 2 we derived the high-fidelity non-linear, continuous-time equations of motion that were used in Chapter 3 for simulating the N-rotor vehicle. Using these same equations of motion for designing and implementing a controller to autonomously fly the N-rotor vehicle is complicated due to the non-linearities. As established in the introduction to this script, the goal is to achieve stable flight of the N-rotor vehicle using controller design techniques taught in the under-graduate control system classes. As under-graduate control classes focus primarily on control and analysis of linear systems, we need to first derive a linearisation of the equation of motion about a suitable operating condition.

This chapter builds up in a logical sequence the tools necessary to synthesise a controller. In Section 4.1 we establish the equilibrium point about which the equations of motion will be linearised. In Section 4.2 we briefly review the theory of linearisation, state the linearised system matrices, and provide hints for their derivation. In Section 4.3 we describe the basic control architecture that will be used to achieve stable hover, motivating both the cascaded and decoupled aspects of the architecture. Descriptions and rules-of-thumb for PID and LQR design are not currently included in this script, and instead the reader is encouraged to make connections with their under-graduates control theory material for implementing the required PID and LQR controllers.

4.1 Equilibrium state and input

The natural guess for an equilibrium point of the non-linear equations of motion is when the N-rotor vehicle is hovering flat at an arbitrary position and yaw angle but without changing its position or yaw angle. The natural choice of input that would maintain the N-rotor vehicle at this state is with the sum of the f_i rotor forces summing to weight of the vehicle, i.e., summing to mg, and chosen such that there is zero net torque about each of the body frame axes. This deduction for the equilibrium state is expressed as:

- $\vec{p}$ and α arbitrary, with $\dot{\vec{p}} = 0$ and $\dot{\alpha} = 0$,
- $\gamma = \beta = 0$ to be flat, with $\dot{\gamma} = \dot{\beta} = 0$,
- The f_i rotor forces chosen to satisfy:

$$\begin{bmatrix} f_{\text{total}} \\ \tau_x^{(\text{B})} \\ \tau_y^{(\text{B})} \\ \tau_z^{(\text{B})} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & \cdots & 1 \\ y_1 & \cdots & y_N \\ -x_1 & \cdots & -x_N \\ c_1 & \cdots & c_N \end{bmatrix}}_{M_{\text{layout}}} \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix} = \begin{bmatrix} mg \\ 0 \\ 0 \\ 0 \end{bmatrix} . \tag{4.1}$$

Indeed, substituting these choices into the equations of motion will yield zero linear and rotational accelerations, i.e., $\ddot{\vec{p}} = \ddot{\vec{\psi}} = 0$, confirming that this is in fact an equilibrium point for the N-rotor vehicle.

4.2 Linearisation

4.2.1 Review of linearisation theory

For this sub-section we use the notation that $x \in \mathbb{R}^n$ is the state vector for a n-dimensional system, $u \in \mathbb{R}^m$ is the m-dimensional input to the system, and $g : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$ is the continuous-time equations of motion of the system, also referred to as the dynamics function. In other words, this general system evolves according to:

$$\dot{x} \,=\, g\left(x, u\right) \,. \tag{4.2}$$

Given a stationary (or equilibrium) point of this system, denoted as $(x_{\text{eq}}, u_{\text{eq}})$ satisfying $g\left(x_{\text{eq}}, u_{\text{eq}}\right) = 0$, the behaviour of the system around $(x_{\text{eq}}, u_{\text{eq}})$ can be approximated using a first order Taylor series expansion of the dynamics:

$$\dot{x} = g(x,u) \approxeq \underbrace{g(x_{\text{eq}}, u_{\text{eq}})}_{=0} + \underbrace{\left.\frac{\partial g}{\partial x^\top}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}}}_{A_{\text{eq}}} \left(\, x - x_{\text{eq}} \,\right) + \underbrace{\left.\frac{\partial g}{\partial u^\top}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}}}_{B_{\text{eq}}} \left(\, u - u_{\text{eq}} \,\right) \tag{4.3}$$

where $A \in \mathbb{R}^{n\times n}$ is the state evolution matrix and $B \in \mathbb{R}^{n\times m}$ is the input matrix of the linear system approximation. Defining new quantities that are the deviation of the state and input from equilibrium, i.e., $\Delta x = x - x_{\text{eq}}$ and $\Delta u = u - u_{\text{eq}}$, the linear system approximation about equilibrium is concisely written as:

$$\Delta\dot{x} \,=\, A_{\text{eq}}\,\Delta x \,+\, B_{\text{eq}}\,\Delta u \,. \tag{4.4}$$

Note that if a controller is designed based on this linear approximation of the original system, then the controller output will be Δu, and hence the equilibrium input must be added when requesting a control action from the system, i.e., $u = u_{\text{eq}} + \Delta u$.

The notation we have used for the partial derivatives of the dynamics function places a transpose on the denominator in an attempt to make it clear how to construct the A_{eq} and B_{eq} matrices. For clarity, we now write out the partial derivative in is entirety. Let $g_i : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$, $i = 1, \ldots, n$, denote each component of the dynamics function, i.e.,

$$\dot{x} \,=\, \begin{bmatrix} \dot{x}_1 \\ \vdots \\ \dot{x}_n \end{bmatrix} \,=\, \begin{bmatrix} g_1(x,u) \\ \vdots \\ g_n(x,u) \end{bmatrix} \,=\, g\left(x,u\right) \,, \tag{4.5}$$

where the subscript x_i refers to the i^{th} component of the n-dimensional state vector. The A_{eq} and B_{eq} matrices are thus constructed by stacking together the scalar partial derivatives into the following matrices:

$$A_{\text{eq}} \,= \left.\frac{\partial g}{\partial x^\top}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} \,=\, \begin{bmatrix} \left.\frac{\partial g_1}{\partial x_1}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} & \cdots & \left.\frac{\partial g_1}{\partial x_n}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} \\ \vdots & \ddots & \vdots \\ \left.\frac{\partial g_n}{\partial x_1}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} & \cdots & \left.\frac{\partial g_n}{\partial x_n}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} \end{bmatrix} \,, \tag{4.6a}$$

$$B_{\text{eq}} \,= \left.\frac{\partial g}{\partial u^\top}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} \,=\, \begin{bmatrix} \left.\frac{\partial g_1}{\partial u_1}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} & \cdots & \left.\frac{\partial g_1}{\partial u_m}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} \\ \vdots & \ddots & \vdots \\ \left.\frac{\partial g_n}{\partial u_1}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} & \cdots & \left.\frac{\partial g_n}{\partial u_m}\right|_{\substack{x=x_{\text{eq}}\\u=u_{\text{eq}}}} \end{bmatrix} \,. \tag{4.6b}$$

The notation defined for the state, input, and dynamics in this brief review of linearisation theory does not apply elsewhere in this script unless re-defined.

4.2.2 Linear system about hover for the full state

Let the choice of equilibrium state and input of the N-rotor vehicle at hover, as per Section 4.1, be denoted:

$$\begin{bmatrix}\vec{p}\\ \dot{\vec{p}}\\ \vec{\psi}\\ \dot{\vec{\psi}}\end{bmatrix}_{\text{hover}} = \begin{bmatrix}\vec{p}_{\text{hover}}\\ 0\\ 0\\ 0\end{bmatrix}, \quad \{f_i\}_{i=1}^{N} \text{ such that } \begin{bmatrix}1 & \cdots & 1\\ y_1 & \cdots & y_N\\ -x_1 & \cdots & -x_N\\ c_1 & \cdots & c_N\end{bmatrix}\begin{bmatrix}f_1\\ \vdots\\ f_N\end{bmatrix}_{\text{hover}} = \begin{bmatrix}mg\\ 0\\ 0\\ 0\end{bmatrix}$$

where we have fixed the choice that $\alpha_{\text{hover}} = 0$. Further, let the deviations from this stationary point be denoted:

$$\begin{bmatrix}\Delta\vec{p}\\ \Delta\dot{\vec{p}}\\ \Delta\vec{\psi}\\ \Delta\dot{\vec{\psi}}\end{bmatrix} = \begin{bmatrix}\vec{p}\\ \dot{\vec{p}}\\ \vec{\psi}\\ \dot{\vec{\psi}}\end{bmatrix} - \begin{bmatrix}\vec{p}\\ \dot{\vec{p}}\\ \vec{\psi}\\ \dot{\vec{\psi}}\end{bmatrix}_{\text{hover}}, \quad \begin{bmatrix}\Delta f_1\\ \vdots\\ \Delta f_N\end{bmatrix} = \begin{bmatrix}f_1\\ \vdots\\ f_N\end{bmatrix} - \begin{bmatrix}f_1\\ \vdots\\ f_N\end{bmatrix}_{\text{hover}} \tag{4.7}$$

Then carrying out the linearisation as described in Section 4.2.1, the linear system approximation of the full non-linear, continuous-time equations of motion about the zero yaw hover equilibrium condition is:

$$\begin{bmatrix}\Delta\dot{\vec{p}}\\ \Delta\ddot{\vec{p}}\\ \Delta\dot{\vec{\psi}}\\ \Delta\ddot{\vec{\psi}}\end{bmatrix} = A_{\text{hover}}\begin{bmatrix}\Delta\vec{p}\\ \Delta\dot{\vec{p}}\\ \Delta\vec{\psi}\\ \Delta\dot{\vec{\psi}}\end{bmatrix} + B_{\text{hover}}\begin{bmatrix}\Delta f_1\\ \vdots\\ \Delta f_N\end{bmatrix}, \tag{4.8}$$

$$\underbrace{\begin{bmatrix}
0&0&0&1&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&1&0&0&0&0&0&0\\
0&0&0&0&0&0&0&g&0&0&0&0\\
0&0&0&0&0&0&-g&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&1&0&0\\
0&0&0&0&0&0&0&0&0&0&1&0\\
0&0&0&0&0&0&0&0&0&0&0&1\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0&0&0
\end{bmatrix}}_{A_{\text{hover}}}
\quad
\underbrace{\begin{bmatrix}
0 \;\cdots\; 0\\
0 \;\cdots\; 0\\
0 \;\cdots\; 0\\
0 \;\cdots\; 0\\
0 \;\cdots\; 0\\
\frac{1}{m} \;\cdots\; \frac{1}{m}\\
0 \;\cdots\; 0\\
0 \;\cdots\; 0\\
0 \;\cdots\; 0\\
J^{-1}\begin{bmatrix}y_1 & \cdots & y_N\\ -x_1 & \cdots & -x_N\\ c_1 & \cdots & c_N\end{bmatrix}
\end{bmatrix}}_{B_{\text{hover}}}$$

The reader is encouraged to carry through the linearisation computations, following hints given in Section 4.2.3, and show that these A_{hover} and B_{hover} matrices are in fact correct. The linearisation computations are provided in Appendix A for reference if required.

4.2.3 Hints for deriving the linearisation about hover

If one attempts to compute all the partial derivatives before substituting in the equilibrium state and input, then deriving the linear system matrices will require quite some pages of algebra. The following are a few hints to assist with carrying through the linearisation computations:

- It can be beneficial to perform the linearisation in blocks that corresponds to $\vec{p}$, $\dot{\vec{p}}$, $\vec{\psi}$, $\dot{\vec{\psi}}$, and the vector $[f_1, \ldots, f_N]^\top$.
- First "compute" the linearisation for those blocks that are trivially zero because there is no dependence of the equations of motion for a particular block on the variable of differentiation of that block.

- For the remaining blocks, while carrying through the computations of the partial derivatives, substitute in the equilibrium state and input at convenient steps to conclude that certain terms are identically zero.
- For example, if during the derivation the term $\frac{d}{d\vec{\psi}} \left(\dot{T}(\vec{\psi}) \right) \dot{\vec{\psi}}$ arises, then there is no need to compute the partial derivative $\frac{d}{d\vec{\psi}} \left(\dot{T}(\vec{\psi}) \right)$ because the term $\dot{\vec{\psi}}$ is identically zero at equilibrium.

4.3 Control Architecture

Based on the linearisation presented in Section 4.2, there are three distinct control architecture choices described in this section. The first is a feed-forward term that is essentially the f_{hover} equilibrium required to opposed gravity and that applies zero net torque about the body frame axes. The second is a cascaded control structure with two nested loops, and finally a decoupling into a separate controller for each coordinate axis direction and the yaw angle.

4.3.1 Feed-forward

When designing a controller based linear system matrices obtained via linearisation, it is important to remember that the equilibrium point chosen is a combination of:

(1) an equilibrium state of the system, and

(2) the equilibrium input that keeps the system at that equilibrium state.

For an N-rotor vehicle, the inputs to the vehicle are the thrust for each propeller, and therefore a controller designed for a linearised N-rotor vehicle system is a function that:

- should receive the **error** between the current state of the system and the **reference** equilibrium state, and
- returns the **thrust adjustments** that should be made to the **equilibrium thrusts** of each propeller.

This equilibrium input is commonly referred to as the feed-forward input, and the feed-forward architecture for an N-rotor vehicle is shown in Figure 4.1. The control function can therefore be described in words as:

$$\begin{array}{c}\text{propeller}\\\text{thrusts}\end{array} = \begin{array}{c}\text{equilibrium}\\\text{thrusts}\end{array} + \begin{array}{c}\text{thrust}\\\text{adjustments}\end{array}. \tag{4.9}$$

For stabilising the N-rotor vehicle about hover, the equilibrium thrusts are any set of N propeller thrusts, $\{f_i\}_{i=1}^{N}$, that satisfy equation (4.1), repeated here for convenience,

$$\underbrace{\begin{bmatrix} 1 & \cdots & 1 \\ y_1 & \cdots & y_N \\ -x_1 & \cdots & -x_N \\ c_1 & \cdots & c_N \end{bmatrix}}_{M_{\text{layout}}} \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix} = \begin{bmatrix} mg \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Two important factors that arise when solving this equation for the equilibrium thrusts are:

(1) The chosen equilibrium state of the system does not appear in the equation, and hence the equilibrium thrusts are the same for all positions and yaw angles in the state space. Hence in Figure 4.1 the connection between the reference and the feed-forward block is not required.

(2) If the vehicle has $N \geq 5$ rotors in a non-redundant configuration, then there are infinite possible equilibrium thrusts. As the controller makes both positive and negative adjustments to the equilibrium thrusts, a sensible choice is one for which each equilibrium thrust is sufficiently far from the minimum and maximum thrust that can be delivered by the respective propeller. Note that using the pseudo-inverse of M_{layout} is generally not desirable when $N \geq 5$. This is because the pseudo-inverse computes one or more of the equilibrium thrusts to be zero, and thus for a normal propeller and motor design, a negative thrust adjustment request from the controller cannot be physically delivered.

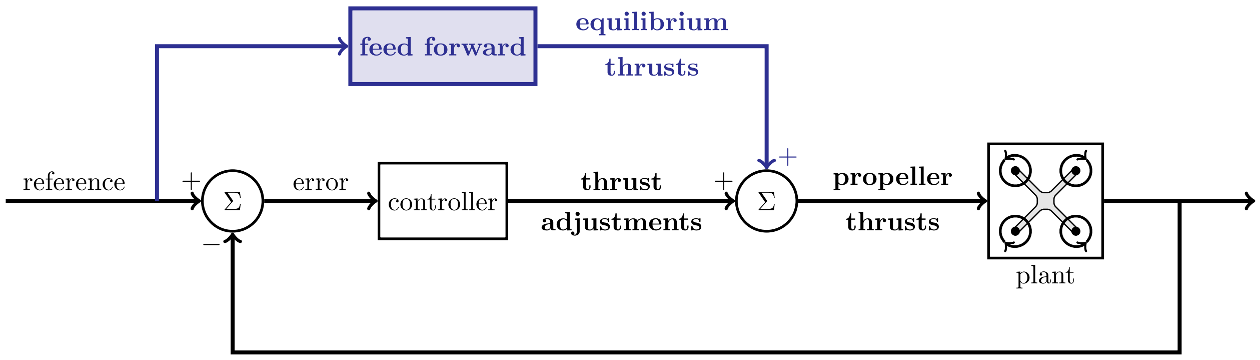


Figure 4.1: Schematic of the feed-forward architecture for an N-rotor vehicle. When the controller design is based on a linearisation of the plant, then the computed controller actions are adjustments to the equilibrium inputs that would keep that plant at the chosen equilibrium state, and equilibrium inputs are commonly referred to as the **feed-forward** term. For an N-rotor vehicle this feed-forward term is the N equilibrium propeller thrusts that satisfy equation (4.1).

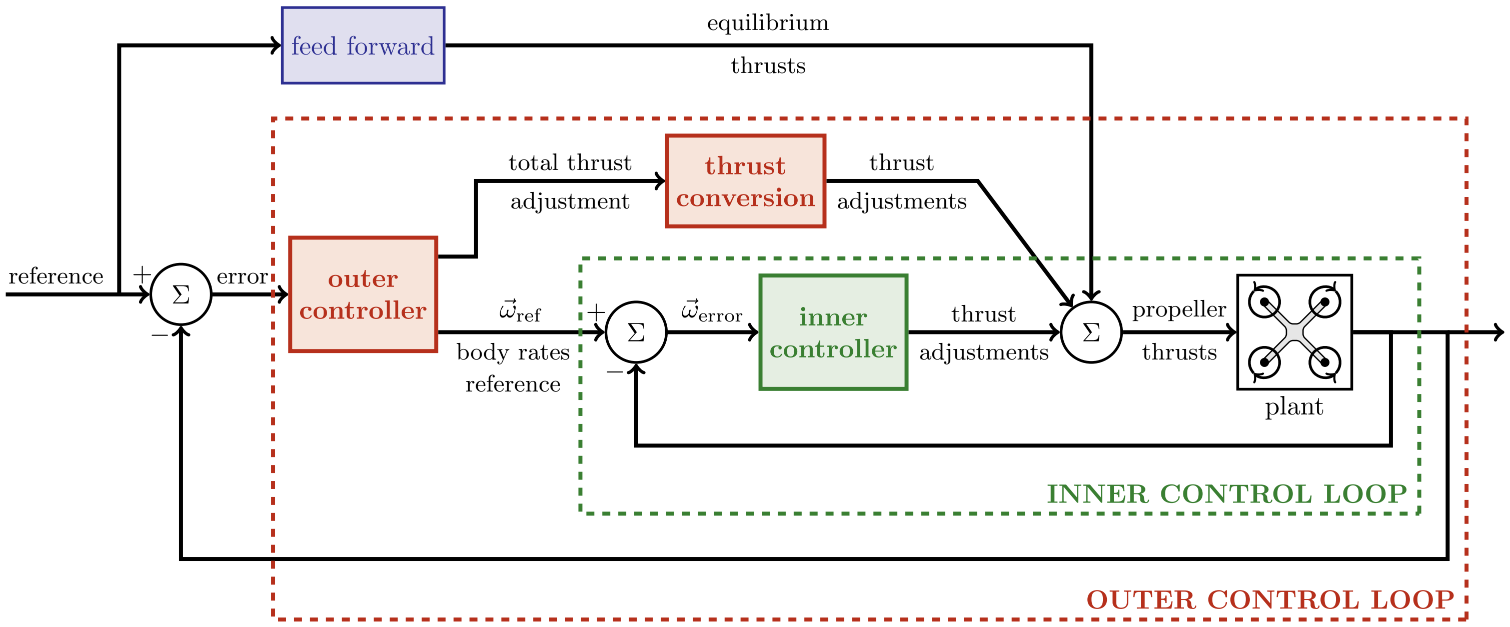


Figure 4.2: Schematic of the cascaded architecture for an N-rotor vehicle. The "thrust conversion" block is needed because the "total thrust adjustment" from the "outer controller" is a scalar quantity, and it must be converted to a vector of N "thrust adjustments" that together provide the "total thrust adjustment" requested and apply zero net torque about the body frame axes.

4.3.2 Cascaded architecture

To control an N-rotor vehicle, it is common practice to use a cascaded control architecture where the controller is separated into two nested feedback control loops. The practical motivation for this cascaded architecture is that the Inertial Measurement Unit (IMU) on-board the N-rotor vehicle has a gyroscope that provides high-frequency (up to 1000Hz) measurements of the angular rates about the body frame axes of the vehicle, i.e., measurements of $\vec{\omega}$. This is in contrast to the position and attitude measurements that are generally available at a much lower frequency. When using an external motion capture system, the frequency of position and attitude measurements could be as high as 200Hz, however, other types of sensors may be limited to frequencies of less than 50Hz.

Based on these two separate measurement streams, the cascaded control architecture is shown in Figure 4.2 and can be described in words as:

- the **"inner controller"** uses the body rate measurements, and is designed to regulate deviations from the body rates reference provided by the "outer controller". The control actions taken by the inner controller are thrust adjustments to each of the propellers.
- the **"outer controller"** uses the measurements of (x, y, z) position in the inertial frame, and the (γ, β, α) attitude angles, and is designed to regulate deviations from the $(x^{(\mathrm{I})}, y^{(\mathrm{I})}, z^{(\mathrm{I})}, \alpha)$ reference that is provided by an external oracle. The control actions taken by the outer controller are the body rates reference passed to the inner controller, and the also thrust adjustments to each of the propellers.

Aside from the practical aspect of different measurement frequencies, the cascaded control architecture also simplifies the controller synthesis because we can design the outer loop assuming that the inner loop tracks the requests infinitely fast, i.e., without dynamics or delay. This assumption means that the difference between when the outer loop requests desired body rates $\vec{\omega}$ from the inner loop, and the time when the inner loop tracks this request is negligible relative to the time scale of the outer loop dynamics. One method of checking whether this separation exists is, after the controllers are designed, to check that the slowest eigenvalue of the inner loop is 5-10 times faster than the fastest eigenvalue of the outer loop.

The assumption of an infinitely fast inner loop allows the equations of motion to be split into separate equations of motion for each loop. The state-space representation of the linearised equations of motion for the outer loop, linearised about the hover equilibrium state, is:

$$\begin{bmatrix} \Delta\dot{p}_x \\ \Delta\dot{p}_y \\ \Delta\dot{p}_z \\ \Delta\ddot{p}_x \\ \Delta\ddot{p}_y \\ \Delta\ddot{p}_z \\ \Delta\dot{\gamma} \\ \Delta\dot{\beta} \\ \Delta\dot{\alpha} \end{bmatrix} = \underbrace{\begin{bmatrix} 0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&g&0 \\ 0&0&0&0&0&0&-g&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0 \end{bmatrix}}_{A_{\text{outer}}} \begin{bmatrix} \Delta p_x \\ \Delta p_y \\ \Delta p_z \\ \Delta\dot{p}_x \\ \Delta\dot{p}_y \\ \Delta\dot{p}_z \\ \Delta\gamma \\ \Delta\beta \\ \Delta\alpha \end{bmatrix} + \underbrace{\begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ \frac{1}{m}&0&0&0 \\ 0&1&0&0 \\ 0&0&1&0 \\ 0&0&0&1 \end{bmatrix}}_{B_{\text{outer}}} \begin{bmatrix} \Delta f_{\text{total}} \\ \Delta\omega_{x,\text{ref}} \\ \Delta\omega_{y,\text{ref}} \\ \Delta\omega_{z,\text{ref}} \end{bmatrix}, \tag{4.10}$$

where the state is deviations from the equilibrium state of the position $\vec{p}$, velocity $\dot{\vec{p}}$, and intrinsic Euler angles $\vec{\psi}$, and the inputs are total thrust f_{total} and body rates reference $\vec{\omega}_{\text{ref}}$ that are given to the inner controller. Note the the A_{outer} matrix is identical to the upper three-by-three blocks of the A_{hover} matrix given in equation (4.8). The equations of motion for the inner loop, linearised about the hover equilibrium state, are:

$$\begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = \underbrace{\begin{bmatrix} 0&0&0 \\ 0&0&0 \\ 0&0&0 \end{bmatrix}}_{A_{\text{inner}}} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} + \underbrace{J^{-1}}_{B_{\text{inner}}} \begin{bmatrix} \tau_x^{(\mathrm{B})} \\ \tau_y^{(\mathrm{B})} \\ \tau_z^{(\mathrm{B})} \end{bmatrix}, \tag{4.11}$$

where the state is deviations from the equilibrium state of the body rates $\vec{\omega}$, and the inputs are the actuator torques $(\tau_x^{(B)}, \tau_y^{(B)}, \tau_z^{(B)})$. Note the the Δ notation is not used here because the equilibrium state is zero for all states and input. If the body frame axes are chosen such that the mass moment of inertia matrix, J, is approximately diagonal, then J^{-1} is also approximately diagonal, i.e.,

$$J^{-1} \approx \begin{bmatrix} \frac{1}{J_x} & 0 & 0 \\ 0 & \frac{1}{J_y} & 0 \\ 0 & 0 & \frac{1}{J_y} \end{bmatrix} .$$

This highlights that the equations of motions for rotations about the body frame axes are only weakly coupled and motivates the decoupled inner loop control architecture described in Section 4.3.3. Moreover, choosing the body frame axes to achieve this near diagonal structure is generally possible for an N-rotor vehicle because it is typically a near-planar device and thus one of the principle axes is closely aligned with the direction of thrust of the propellers.

4.3.3 Decoupling

Equation 4.10 shows that the linearised equations of motion of the outer controller have a sparse structure. In fact, a re-ordering of the state results in the following block diagonal system matrices,

$$\begin{bmatrix} \Delta\dot{p}_x \\ \Delta\ddot{p}_x \\ \Delta\dot{\beta} \\ \\ \Delta\dot{p}_y \\ \Delta\ddot{p}_y \\ \Delta\dot{\gamma} \\ \\ \Delta\dot{p}_z \\ \Delta\ddot{p}_z \\ \\ \Delta\dot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & & & & & & \\ 0 & 0 & g & & & & & & \\ 0 & 0 & 0 & & & & & & \\ & & & 0 & 1 & 0 & & & \\ & & & 0 & 0 & -g & & & \\ & & & 0 & 0 & 0 & & & \\ & & & & & & 0 & 1 & \\ & & & & & & 0 & 0 & \\ & & & & & & & & 0 \end{bmatrix} \begin{bmatrix} \Delta p_x \\ \Delta\dot{p}_x \\ \Delta\beta \\ \\ \Delta p_y \\ \Delta\dot{p}_y \\ \Delta\gamma \\ \\ \Delta p_z \\ \Delta\dot{p}_z \\ \\ \Delta\alpha \end{bmatrix} + \begin{bmatrix} 0 & & & \\ 0 & & & \\ 1 & & & \\ & 0 & & \\ & 0 & & \\ & 1 & & \\ & & 0 & \\ & & \frac{1}{m} & \\ & & & 1 \end{bmatrix} \begin{bmatrix} \Delta\omega_{y,\text{ref}} \\ \Delta\omega_{x,\text{ref}} \\ \Delta f_{\text{total}} \\ \Delta\omega_{z,\text{ref}} \end{bmatrix} . \tag{4.12}$$

This block diagonal structure indicates that for states near equilibrium the N-rotor vehicle is well approximated by four completely independent equations of motion:

- The equations relating to the inertial $p_x^{(\text{I})}$ position are coupled to the pitch angle β and $\omega_{y,\text{ref}}$ influences only $\dot{\beta}$.
- The equations relating to the inertial $p_y^{(\text{I})}$ position is coupled to the roll angle γ and $\omega_{x,\text{ref}}$ influences only $\dot{\gamma}$.
- The equations relating to the inertial $p_z^{(\text{I})}$ position are completely decoupled from the other states and the input f_{total} influences only $\ddot{p}_z$.
- The equations relating to the yaw α angle are completely decoupled from the other states and the input $\omega_{z,\text{ref}}$ influences only $\dot{\alpha}$.

Thus we decouple the architecture into four separate controllers, one for each of the reference signals:

- We use the pitch rate $\omega_{y,\text{ref}}$ to control an inertial frame $p_x^{(\text{I})}$ position error.
- We use the roll rate $\omega_{x,\text{ref}}$ to control an inertial frame $p_y^{(\text{I})}$ position error.
- We use the total thrust f_{total} to control an inertial frame $p_z^{(\text{I})}$ position error.
- We use the yaw rate $\omega_{z,\text{ref}}$ to control the α angle error.

A restriction of this decoupled control architecture based on the linearised equations of motion is that the yaw angle must stay near zero for the linearisation, and hence the decoupling, to be valid. If we carry out the linearisation in Section 4.2.2 for a general yaw angle we see that it introduces a coupling between the inertial frame $p_x^{(\text{I})}$ and $p_y^{(\text{I})}$ positions, while $p_z^{(\text{I})}$ and α remain decoupled. One option to address this coupling would be to design controllers for various choices of linearisation yaw angle, for example every 30 degrees, and use the controller that is “closest” to the measured yaw

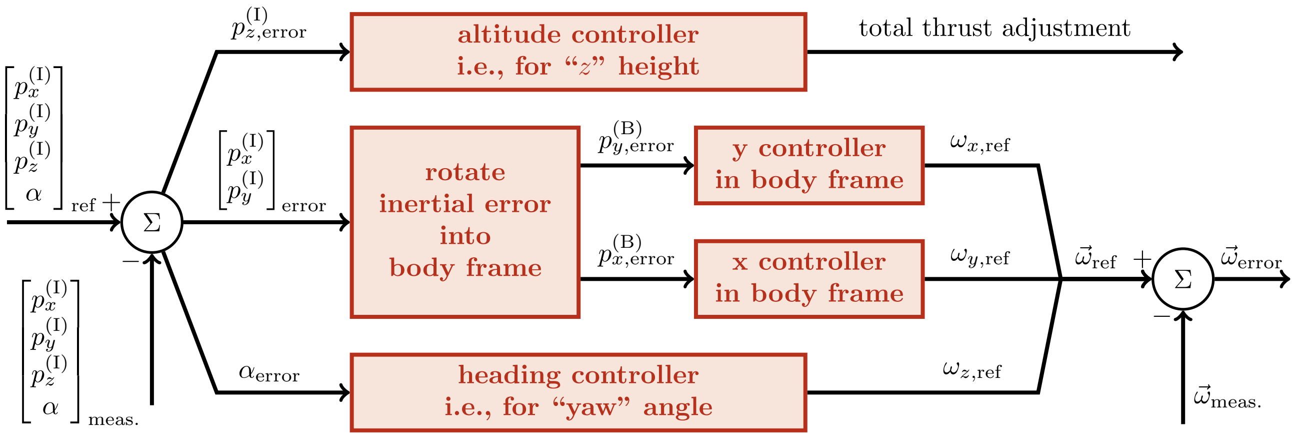

$p_{z,\text{error}}^{(I)}$
altitude controller i.e., for "z" height
total thrust adjustment
$\begin{bmatrix} p_x^{(I)} \\ p_y^{(I)} \\ p_z^{(I)} \\ \alpha \end{bmatrix}_{\text{ref}}$
+
Σ
−
$\begin{bmatrix} p_x^{(I)} \\ p_y^{(I)} \end{bmatrix}_{\text{error}}$
rotate inertial error into body frame
$p_{y,\text{error}}^{(B)}$
y controller in body frame
$\omega_{x,\text{ref}}$
$p_{x,\text{error}}^{(B)}$
x controller in body frame
$\omega_{y,\text{ref}}$
$\vec{\omega}_{\text{ref}}$
$\vec{\omega}_{\text{error}}$
$\begin{bmatrix} p_x^{(I)} \\ p_y^{(I)} \\ p_z^{(I)} \\ \alpha \end{bmatrix}_{\text{meas.}}$
α_{error}
heading controller i.e., for "yaw" angle
$\omega_{z,\text{ref}}$
$\vec{\omega}_{\text{meas.}}$

Figure 4.3: Schematic of the outer controller architecture for an N-rotor vehicle. The four controllers are essentially independent. The altitude controller regulates deviations in the height on the vehicle, while the heading controller regulates deviations in the heading direction. The (p_x, p_y) horizontal position controllers are slightly coupled through the "rotate inertial error into body frame" block. This block is required because the (p_x, p_y) controllers use body frame actuation but the measurements are inertial frame positions.

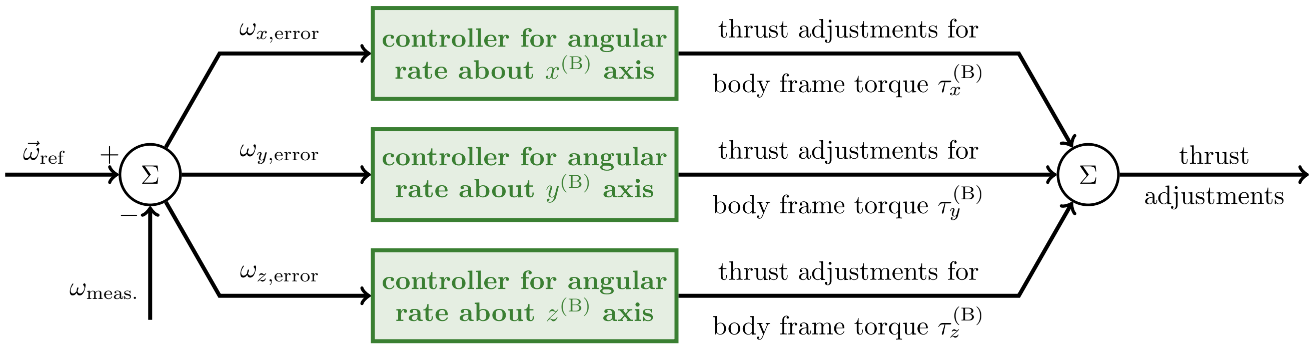

$\omega_{x,\text{error}}$
controller for angular rate about $x^{(B)}$ axis
thrust adjustments for body frame torque $\tau_x^{(B)}$
$\vec{\omega}_{\text{ref}}$
+
Σ
−
$\omega_{y,\text{error}}$
controller for angular rate about $y^{(B)}$ axis
thrust adjustments for body frame torque $\tau_y^{(B)}$
thrust adjustments
$\omega_{\text{meas.}}$
$\omega_{z,\text{error}}$
controller for angular rate about $z^{(B)}$ axis
thrust adjustments for body frame torque $\tau_z^{(B)}$

Figure 4.4: Schematic of the inner controller architecture for an N-rotor vehicle. The three controllers independently regulate angular rates deviations about each of the body frame axes.

angle at any time instant. However, this is cumbersome to implement and to choose the linearisation points. Closer inspection of the linearisation about a general yaw angle reveals that the coupling is essentially a rotation matrix. Thus a simple solution for controlling the inertial frame $p_x^{(I)}$ and $p_y^{(I)}$ positions is to first rotate them into body frame $p_x^{(B)}$ and $p_y^{(B)}$ positions, and then use a controller based on the the decoupled equations (4.12) to synthesise a controller for the body frame positions.

This decoupled architecture for the outer controller is shown in Figure 4.3. The figure shows a separate controller for the inertial frame $p_z^{(I)}$ position and α angle, and it shows that the inertial frame $p_x^{(I)}$ and $p_y^{(I)}$ controllers are coupled by first rotating the inertial frame position error into a body frame error, and then a separate controller is used for each coordinate.

Equation 4.11 shows that when the mass moment of inertia matrix is diagonal the linearised equations of motion of the inner controller are also diagonal. Thus we decouple the architecture of the inner loop into three separate controllers, one for each of the reference signals $\omega_{x,\text{ref}}$, $\omega_{y,\text{ref}}$, and $\omega_{z,\text{ref}}$. This decoupled architecture for the inner controller is shown in Figure 4.4.

Bibliography

[1] Voliro. Online: `https://www.voliro.ethz.ch`, Checked Feb 2018.

[2] Simon L. Altmann. *Rotations, quaternions, and double groups.* Dover Publications, Inc., 2005.

[3] John C. Butcher. *Numerical Methods for Ordinary Differential Equations.* John Wiley & Sons, Ltd, Chichester, West Sussex, second edition, 2008.

[4] GuerrillaCG. Euler (gimbal lock) explained. YouTube: `https://www.youtube.com/watch?v=zc8b2Jo7mno`, Published Jan 2009.

[5] William R. Hamilton. On a new species of imaginary quantities connected with a theory of quaternions. In *Proceedings of the Royal Irish Academy*, volume 2, pages 424–434, 1844.

[6] Jack B Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality.* Princeton University Press, 1999.

[7] MathWorks. Matlab tutorials. Online: `https://mathworks.com/support/learn-with-matlab-tutorials.html`, Checked Feb 2018.

[8] MathWorks. Simulink. Online: `https://ch.mathworks.com/products/simulink.html`, Checked Feb 2018.

[9] Mark W. Mueller, Michael Hamer, and Raffaello D'Andrea. Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1730–1736, Seattle, Washington, May 2015.

Appendix A

Linearisation Derivation

To keep this derivation self contained, we re-state the choice of the full state, the equations of motion, and the equation for the rotation and transformation matrix. The full state is defined as:

$$\begin{bmatrix} \vec{p} \\ \dot{\vec{p}} \\ \vec{\psi} \\ \dot{\vec{\psi}} \end{bmatrix} = \begin{bmatrix} \text{position} \\ \text{linear velocity} \\ \text{attitude} \\ \text{angular velocity} \end{bmatrix} , \tag{A.1}$$

for which the non-linear, continuous-time equations of motion are:

$$\dot{\vec{p}} = \frac{d}{dt} (\vec{p}) \tag{A.2a}$$

$$\ddot{\vec{p}} = \begin{bmatrix} \ddot{p}_x \\ \ddot{p}_y \\ \ddot{p}_z \end{bmatrix} = \frac{1}{m} \left({}_{(\mathrm{I})}R^{(\mathrm{B})}(\vec{\psi}) \begin{bmatrix} 0 \\ 0 \\ \sum\limits_{i=1}^{N} f_i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \right) , \tag{A.2b}$$

$$\dot{\vec{\psi}} = \frac{d}{dt} \left(\vec{\psi} \right) \tag{A.2c}$$

$$\ddot{\vec{\psi}} = \begin{bmatrix} \ddot{\gamma} \\ \ddot{\beta} \\ \ddot{\alpha} \end{bmatrix} = T^{-1}(\vec{\psi}) \left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi}, \dot{\vec{\psi}}) \, \dot{\vec{\psi}} \right) , \tag{A.2d}$$

$$\dot{\vec{\omega}} = \begin{bmatrix} \dot{\omega}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} = J^{-1} \left(\begin{bmatrix} \sum_{i=1}^{N} \; f_i \, y_i \\ \sum_{i=1}^{N} -f_i \, x_i \\ \sum_{i=1}^{N} \; f_i \, c_i \end{bmatrix} - (\vec{\omega} \times J \vec{\omega}) \right) . \tag{A.2e}$$

The rotation matrix, transformation matrix, its inverse, and its time derivative are given by the following expressions:

$${}_{(\mathrm{I})}R^{(\mathrm{B})}(\vec{\psi}) = \begin{bmatrix} c_\alpha \, c_\beta & (-s_\alpha \, c_\gamma + c_\alpha \, s_\beta \, s_\gamma) & (s_\alpha \, s_\gamma + c_\alpha \, s_\beta \, c_\gamma) \\ s_\alpha \, c_\beta & (c_\alpha \, c_\gamma + s_\alpha \, s_\beta \, s_\gamma) & (-c_\alpha \, s_\gamma + s_\alpha \, s_\beta \, c_\gamma) \\ -s_\beta & c_\beta \, s_\gamma & c_\beta \, c_\gamma \end{bmatrix} \tag{A.3}$$

$$T(\vec{\psi}) = \begin{bmatrix} 1 & 0 & -\sin(\beta) \\ 0 & \cos(\gamma) & \sin(\gamma)\cos(\beta) \\ 0 & -\sin(\gamma) & \cos(\gamma)\cos(\beta) \end{bmatrix} . \tag{A.4}$$

$$T^{-1}(\vec{\psi}) = \begin{bmatrix} 1 & \sin(\gamma)\tan(\beta) & \cos(\gamma)\tan(\beta) \\ 0 & \cos(\gamma) & -\sin(\gamma) \\ 0 & \sin(\gamma)\sec(\beta) & \cos(\gamma)\sec(\beta) \end{bmatrix} . \tag{A.5}$$

$$\dot{T}(\vec{\psi}, \dot{\vec{\psi}}) = \begin{bmatrix} 0 & 0 & -\dot{\beta}\cos(\beta) \\ 0 & -\dot{\gamma}\sin(\gamma) & \dot{\gamma}\cos(\gamma)\cos(\beta) - \dot{\beta}\sin(\gamma)\sin(\beta) \\ 0 & -\dot{\gamma}\cos(\gamma) & -\dot{\gamma}\sin(\gamma)\cos(\beta) - \dot{\beta}\cos(\gamma)\sin(\beta) \end{bmatrix} . \tag{A.6}$$

Finally, the equilibrium point about which we will derive the linearisation in hover, i.e., $\vec{p} = \dot{\vec{p}} = \vec{\psi} = \dot{\vec{\psi}} = 0$, and the per-rotor thrust input that maintains this equilibrium satisfies the following actuations:

$$\begin{array}{rcccl} \sum_{i=1}^{N} \; f_i & = & f_{\text{total}} & = & mg\,, \\ \sum_{i=1}^{N} \; f_i\, y_i & = & \tau_x^{(\mathrm{B})} & = & 0\,, \\ \sum_{i=1}^{N} \; -f_i\, x_i & = & \tau_y^{(\mathrm{B})} & = & 0\,, \\ \sum_{i=1}^{N} \; f_i\, c_i & = & \tau_z^{(\mathrm{B})} & = & 0\,. \end{array} \tag{A.7}$$

For a non-degenerate N-rotor vehicle with $N \geq 5$ rotors, there is an continuum of equilibrium thrusts that provide this actuation, but we will see that for the purpose of deriving the linearisation about hover, the exact choice of each f_i has no effect. Note that from an implementation point of view the choice of equilibrium thrust can be important.

To make the presentation of the derivation clearer, we first write out the structure of the linearised matrices. We will use the notation $\Delta\vec{p}$ to indicate a the deviation of $\vec{p}$ from its equilibrium value. Thus, the linearised dynamics are split into the following blocks:

$$\begin{bmatrix} \Delta\dot{\vec{p}} \\ \Delta\ddot{\vec{p}} \\ \Delta\dot{\vec{\psi}} \\ \Delta\ddot{\vec{\psi}} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{d}{d\vec{p}}\dot{\vec{p}} & \frac{d}{d\dot{\vec{p}}}\dot{\vec{p}} & \frac{d}{d\vec{\psi}}\dot{\vec{p}} & \frac{d}{d\dot{\vec{\psi}}}\dot{\vec{p}} \\ \frac{d}{d\vec{p}}\ddot{\vec{p}} & \frac{d}{d\dot{\vec{p}}}\ddot{\vec{p}} & \frac{d}{d\vec{\psi}}\ddot{\vec{p}} & \frac{d}{d\dot{\vec{\psi}}}\ddot{\vec{p}} \\ \frac{d}{d\vec{p}}\dot{\vec{\psi}} & \frac{d}{d\dot{\vec{p}}}\dot{\vec{\psi}} & \frac{d}{d\vec{\psi}}\dot{\vec{\psi}} & \frac{d}{d\dot{\vec{\psi}}}\dot{\vec{\psi}} \\ \frac{d}{d\vec{p}}\ddot{\vec{\psi}} & \frac{d}{d\dot{\vec{p}}}\ddot{\vec{\psi}} & \frac{d}{d\vec{\psi}}\ddot{\vec{\psi}} & \frac{d}{d\dot{\vec{\psi}}}\ddot{\vec{\psi}} \end{bmatrix}}_{A} \begin{bmatrix} \Delta\vec{p} \\ \Delta\dot{\vec{p}} \\ \Delta\vec{\psi} \\ \Delta\dot{\vec{\psi}} \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{d}{d\vec{f}}\dot{\vec{p}} \\ \frac{d}{d\vec{f}}\ddot{\vec{p}} \\ \frac{d}{d\vec{f}}\dot{\vec{\psi}} \\ \frac{d}{d\vec{f}}\ddot{\vec{\psi}} \end{bmatrix}}_{B} \Delta\vec{f}\,,$$

where we have introduced the notation $\vec{f} = [f_1, \ldots, f_N]^\top$ to be the column vector of the individual rotor thrust forces f_i stacked together. The size of each block in the A matrix is 3-by-3, and the size of each block in the B matrix is 3-by-N.

Looking at the dependencies in the equations of motion A.2 it is clear that many of the blocks in the linearisation matrix will be identically zero due to there being no dependency of the expression on the variable of derivation. It is also clear that the blocks $\frac{d}{d\dot{\vec{p}}}\dot{\vec{p}}$ and $\frac{d}{d\dot{\vec{\psi}}}\dot{\vec{\psi}}$ will each be a 3-by-3 identity matrix. We write out the the structure of the linearised matrices with these blocks set to zero:

$$\begin{bmatrix} \Delta\dot{\vec{p}} \\ \Delta\ddot{\vec{p}} \\ \Delta\dot{\vec{\psi}} \\ \Delta\ddot{\vec{\psi}} \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & I_{3\times3} & 0 & 0 \\ 0 & 0 & \frac{d}{d\vec{\psi}}\ddot{\vec{p}} & 0 \\ 0 & 0 & 0 & I_{3\times3} \\ 0 & 0 & \frac{d}{d\vec{\psi}}\ddot{\vec{\psi}} & \frac{d}{d\dot{\vec{\psi}}}\ddot{\vec{\psi}} \end{bmatrix}}_{A} \begin{bmatrix} \Delta\vec{p} \\ \Delta\dot{\vec{p}} \\ \Delta\vec{\psi} \\ \Delta\dot{\vec{\psi}} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{d}{d\vec{f}}\ddot{\vec{p}} \\ 0 \\ \frac{d}{d\vec{f}}\ddot{\vec{\psi}} \end{bmatrix}}_{B} \Delta\vec{f}\,.$$

Before deriving the linearisation for the non-zero blocks, we note the following points:

- the blocks that are zero or $I_{3\times3}$ will be such regardless of the state and input combination around which the dynamics is linearised.
- The blocks are denoted as vector-by-vector derivation, and the convention we use for the resulting matrix is: the vector expression being derived "goes down the rows", and the vector by which we are deriving "goes across the columns", for example:

$$\frac{d}{d\vec{\psi}}\ddot{\vec{p}} = \begin{bmatrix} \frac{d}{d\gamma}\ddot{p}_x & \frac{d}{d\beta}\ddot{p}_x & \frac{d}{d\alpha}\ddot{p}_x \\ \frac{d}{d\gamma}\ddot{p}_y & \frac{d}{d\beta}\ddot{p}_y & \frac{d}{d\alpha}\ddot{p}_y \\ \frac{d}{d\gamma}\ddot{p}_z & \frac{d}{d\beta}\ddot{p}_z & \frac{d}{d\alpha}\ddot{p}_z \end{bmatrix} \tag{A.8}$$

- To actually compute the A and B matrix, we must substitute in the hover state and input after performing the derivative.

We now derive the expression for linearising each block of the A and B matrices.

A.1 Linearisation for $\frac{d}{d\vec{\psi}}\ddot{\vec{p}}$

We first separate out the terms:

$$\begin{aligned}
\frac{d}{d\vec{\psi}}\ddot{\vec{p}} &= \frac{d}{d\vec{\psi}}\left(\frac{1}{m}\left({}_{(\mathrm{I})}R^{(\mathrm{B})}(\vec{\psi})\begin{bmatrix}0\\0\\\sum_{i=1}^{N} f_i\end{bmatrix} + \begin{bmatrix}0\\0\\-mg\end{bmatrix}\right)\right)\\
&= \frac{1}{m}\frac{d}{d\vec{\psi}}\left({}_{(\mathrm{I})}R^{(\mathrm{B})}(\vec{\psi})\begin{bmatrix}0\\0\\\sum_{i=1}^{N} f_i\end{bmatrix}\right) + \frac{d}{d\vec{\psi}}\begin{bmatrix}0\\0\\-g\end{bmatrix}\\
&= \frac{1}{m}\frac{d}{d\vec{\psi}}\left(\begin{bmatrix}(s_\alpha\, s_\gamma \,+\, c_\alpha\, s_\beta\, c_\gamma)\sum_{i=1}^{N} f_i\\(-c_\alpha\, s_\gamma \,+\, s_\alpha\, s_\beta\, c_\gamma)\sum_{i=1}^{N} f_i\\c_\beta\, c_\gamma\sum_{i=1}^{N} f_i\end{bmatrix}\right) + \begin{bmatrix}0\\0\\0\end{bmatrix}\\
&= \frac{\sum_{i=1}^{N} f_i}{m}\frac{d}{d\vec{\psi}}\left(\begin{bmatrix}(s_\alpha\, s_\gamma \,+\, c_\alpha\, s_\beta\, c_\gamma)\\(-c_\alpha\, s_\gamma \,+\, s_\alpha\, s_\beta\, c_\gamma)\\c_\beta\, c_\gamma\end{bmatrix}\right),
\end{aligned} \tag{A.9}$$

where we have avoided derivation of the full rotation matrix by first performing the matrix multiplication with the body frame thrust vector. Performing the derivatives:

$$\frac{d}{d\vec{\psi}}\ddot{\vec{p}} = \frac{\sum_{i=1}^{N} f_i}{m}\begin{bmatrix}(s_\alpha\, c_\gamma \,-\, c_\alpha\, s_\beta\, s_\gamma) & (c_\alpha\, c_\beta\, c_\gamma) & (c_\alpha\, s_\gamma \,-\, s_\alpha\, s_\beta\, c_\gamma)\\(-c_\alpha\, c_\gamma \,-\, s_\alpha\, s_\beta\, s_\gamma) & (s_\alpha\, c_\beta\, c_\gamma) & (s_\alpha\, s_\gamma \,-\, c_\alpha\, s_\beta\, c_\gamma)\\-c_\beta\, s_\gamma & -s_\beta\, c_\gamma & 0\end{bmatrix}. \tag{A.10}$$

Substituting in the equilibrium attitude of $\gamma = \beta = \alpha = 0$ and $\sum_{i=1}^{N} f_i = mg$ we get:

$$\left.\frac{d}{d\vec{\psi}}\ddot{\vec{p}}\right|_{\gamma=\beta=\alpha=0,\,\sum_{i=1}^{N} f_i = mg} = g\begin{bmatrix}0 & 1 & 0\\-1 & 0 & 0\\0 & 0 & 0\end{bmatrix} \tag{A.11}$$

A.2 Linearisation for $\frac{d}{d\vec{f}}\ddot{\vec{p}}$

We first separate out the terms:

$$\begin{aligned}
\frac{d}{d\vec{f}}\ddot{\vec{p}} &= \frac{d}{d\vec{f}}\left(\frac{1}{m}\left({}_{(\mathrm{I})}R^{(\mathrm{B})}(\vec{\psi})\begin{bmatrix}0\\0\\\sum_{i=1}^{N} f_i\end{bmatrix} + \begin{bmatrix}0\\0\\-mg\end{bmatrix}\right)\right)\\
&= \frac{1}{m}\,{}_{(\mathrm{I})}R^{(\mathrm{B})}(\vec{\psi})\,\frac{d}{d\vec{f}}\left(\begin{bmatrix}0\\0\\\sum_{i=1}^{N} f_i\end{bmatrix}\right) + \frac{d}{d\vec{\psi}}\begin{bmatrix}0\\0\\0\end{bmatrix}\\
&= \frac{1}{m}\,{}_{(\mathrm{I})}R^{(\mathrm{B})}(\vec{\psi})\begin{bmatrix}0 & \cdots & 0\\0 & \cdots & 0\\1 & \cdots & 1\end{bmatrix}
\end{aligned} \tag{A.12}$$

Substituting in the equilibrium attitude of $\gamma = \beta = \alpha = 0$ we get that the rotation matrix ${}_{(\mathrm{I})}R^{(\mathrm{B})}$ is a 3-by3 identity matrix, hence this block of the B matrix is given by:

$$\left.\frac{d}{d\vec{f}}\ddot{\vec{p}}\right|_{\gamma=\beta=\alpha=0} = \frac{1}{m}\begin{bmatrix}0 & \cdots & 0\\0 & \cdots & 0\\1 & \cdots & 1\end{bmatrix} \tag{A.13}$$

A.3 Linearisation for $\frac{d}{d\vec{\psi}}\ddot{\vec{\psi}}$

The full equation of motion for $\ddot{\vec{\psi}}$ requires us to substitute in the expression for $\dot{\vec{\omega}}$, and then the geometric relationship that $\vec{\omega} = T(\vec{\psi})\dot{\vec{\psi}}$. Thus we see that $\dot{\vec{\omega}}$ has dependence on both $\vec{\psi}$ and $\dot{\vec{\psi}}$.

To avoid the large expressions that result from performing these substitutions and computing the derivative of each term, we maintain a vector notation and conclude which terms will be identically zero when we substitute in the equilibrium state and input.

We start by first applying the product rule to the equation of motion for $\ddot{\vec{\psi}}$:

$$\begin{aligned}\frac{d}{d\vec{\psi}}\ddot{\vec{\psi}} &= \frac{d}{d\vec{\psi}}\left(T^{-1}(\vec{\psi})\left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\dot{\vec{\psi}}\right)\right)\\ &= \frac{d}{d\vec{\psi}}\left(T^{-1}(\vec{\psi})\right)\left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\dot{\vec{\psi}}\right)\\ &\quad + T^{-1}(\vec{\psi})\left(\frac{d\dot{\vec{\omega}}}{d\vec{\psi}} - \frac{d}{d\vec{\psi}}\left(\dot{T}(\vec{\psi},\dot{\vec{\psi}})\right)\dot{\vec{\psi}} - \dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\frac{d\dot{\vec{\psi}}}{d\vec{\psi}}\right).\end{aligned} \tag{A.14}$$

At equilibrium we have that $\vec{\omega} = \dot{\vec{\psi}} = 0$, and that $\dot{\vec{\omega}} = 0$, and that $T(\vec{\psi}) = T^{-1}(\vec{\psi}) = I_{3\times3}$, and that $\dot{T}(\vec{\psi},\dot{\vec{\psi}}) = 0$. Thus, after substituting in the equilibrium condition that only remaining term is:

$$\left.\frac{d}{d\vec{\psi}}\ddot{\vec{\psi}}\right|_{\gamma=\beta=\alpha=\dot{\gamma}=\dot{\beta}=\dot{\alpha}=0} = \left.\frac{d\dot{\vec{\omega}}}{d\vec{\psi}}\right|_{\gamma=\beta=\alpha=\dot{\gamma}=\dot{\beta}=\dot{\alpha}=0}. \tag{A.15}$$

We now derive an expression for $\frac{d}{d\vec{\psi}}\dot{\vec{\omega}}$:

$$\begin{aligned}\frac{d}{d\vec{\psi}}\dot{\vec{\omega}} &= \frac{d}{d\vec{\psi}}\left(J^{-1}\left(\begin{bmatrix}\sum_{i=1}^{N} & f_i\,y_i\\ \sum_{i=1}^{N} & -f_i\,x_i\\ \sum_{i=1}^{N} & f_i\,c_i\end{bmatrix} - (\vec{\omega}\times J\,\vec{\omega})\right)\right)\\ &= J^{-1}\begin{bmatrix}0\\0\\0\end{bmatrix} - \frac{d}{d\vec{\psi}}(\vec{\omega}\times J\,\vec{\omega})\\ &= J^{-1}\begin{bmatrix}0\\0\\0\end{bmatrix} - \left(\frac{d\vec{\omega}}{d\vec{\psi}}\times J\,\vec{\omega} + \vec{\omega}\times J\,\frac{d\vec{\omega}}{d\vec{\psi}}\right).\end{aligned} \tag{A.16}$$

Again, substituting in the equilibrium condition that $\vec{\omega} = 0$, we see that both terms are identically zero regardless of the expression for $\frac{d\vec{\omega}}{d\vec{\psi}}$. Hence this block of the A matrix is:

$$\left.\frac{d}{d\vec{\psi}}\ddot{\vec{\psi}}\right|_{\gamma=\beta=\alpha=\dot{\gamma}=\dot{\beta}=\dot{\alpha}=0} = 0_{3\times3} \tag{A.17}$$

A.4 Linearisation for $\frac{d}{d\dot{\vec{\psi}}}\ddot{\vec{\psi}}$

This derivation follows that same lines at $\frac{d}{d\vec{\psi}}\ddot{\vec{\psi}}$ an concludes the same result for the same reasons. Succinctly, first applying the product rule to the equation of motion for $\ddot{\vec{\psi}}$:

$$\begin{aligned}\frac{d}{d\dot{\vec{\psi}}}\ddot{\vec{\psi}} &= \frac{d}{d\vec{\psi}}\left(T^{-1}(\vec{\psi})\left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\dot{\vec{\psi}}\right)\right)\\ &= \frac{d}{d\dot{\vec{\psi}}}\left(T^{-1}(\vec{\psi})\right)\left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\dot{\vec{\psi}}\right)\\ &\quad + T^{-1}(\vec{\psi})\left(\frac{d\dot{\vec{\omega}}}{d\dot{\vec{\psi}}} - \frac{d}{d\dot{\vec{\psi}}}\left(\dot{T}(\vec{\psi},\dot{\vec{\psi}})\right)\dot{\vec{\psi}} - \dot{T}(\vec{\psi},\dot{\vec{\psi}})\,\frac{d\dot{\vec{\psi}}}{d\dot{\vec{\psi}}}\right).\end{aligned} \tag{A.18}$$

At equilibrium we have that $\vec{\omega} = \dot{\vec{\psi}} = 0$, and that $\dot{\vec{\omega}} = 0$, and that $T(\vec{\psi}) = T^{-1}(\vec{\psi}) = I_{3\times3}$, and that $\dot{T}(\vec{\psi}, \dot{\vec{\psi}}) = 0$. Thus, after substituting in the equilibrium condition that only remaining term is:

$$\left.\frac{d}{d\dot{\vec{\psi}}}\,\ddot{\vec{\psi}}\right|_{\gamma=\beta=\alpha=\dot{\gamma}=\dot{\beta}=\dot{\alpha}=0} = \left.\frac{d\dot{\vec{\omega}}}{d\dot{\vec{\psi}}}\right|_{\gamma=\beta=\alpha=\dot{\gamma}=\dot{\beta}=\dot{\alpha}=0} . \tag{A.19}$$

We now derive an expression for $\frac{d}{d\dot{\vec{\psi}}}\dot{\vec{\omega}}$:

$$\begin{aligned}
\frac{d}{d\dot{\vec{\psi}}}\dot{\vec{\omega}} &= \frac{d}{d\dot{\vec{\psi}}}\left(J^{-1}\left(\begin{bmatrix} \sum_{i=1}^{N} & f_i\, y_i \\ \sum_{i=1}^{N} & -f_i\, x_i \\ \sum_{i=1}^{N} & f_i\, c_i \end{bmatrix} - (\vec{\omega} \times J\,\vec{\omega}) \right)\right) \\
&= J^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \frac{d}{d\dot{\vec{\psi}}}(\vec{\omega} \times J\,\vec{\omega}) \\
&= J^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \left(\frac{d\vec{\omega}}{d\dot{\vec{\psi}}} \times J\,\vec{\omega} + \vec{\omega} \times J\,\frac{d\vec{\omega}}{d\dot{\vec{\psi}}} \right) .
\end{aligned} \tag{A.20}$$

Again, substituting in the equilibrium condition that $\vec{\omega} = 0$, we see that both terms are identically zero regardless of the expression for $\frac{d\vec{\omega}}{d\dot{\vec{\psi}}}$. Hence this block of the A matrix is:

$$\left.\frac{d}{d\dot{\vec{\psi}}}\,\ddot{\vec{\psi}}\right|_{\gamma=\beta=\alpha=\dot{\gamma}=\dot{\beta}=\dot{\alpha}=0} = 0_{3\times3} \tag{A.21}$$

A.5 Linearisation for $\frac{d}{d\vec{f}}\ddot{\vec{\psi}}$

Similar to the above derivations, we start directly with the equation of motion for $\ddot{\vec{\psi}}$ in terms of $\dot{\vec{\omega}}$ and check what we can conclude by substitution of the equilibrium condition.

$$\begin{aligned}
\frac{d}{d\vec{f}}\ddot{\vec{\psi}} &= \frac{d}{d\vec{f}}\left(T^{-1}(\vec{\psi}) \left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi}, \dot{\vec{\psi}})\,\dot{\vec{\psi}} \right)\right) \\
&= \frac{d}{d\vec{f}}\left(T^{-1}(\vec{\psi}) \right) \left(\dot{\vec{\omega}} - \dot{T}(\vec{\psi}, \dot{\vec{\psi}})\,\dot{\vec{\psi}} \right) \\
&\quad + T^{-1}(\vec{\psi}) \left(\frac{d\dot{\vec{\omega}}}{d\vec{f}} - \frac{d}{d\vec{f}}\left(\dot{T}(\vec{\psi}, \dot{\vec{\psi}}) \right) \dot{\vec{\psi}} - \dot{T}(\vec{\psi}, \dot{\vec{\psi}})\,\frac{d\dot{\vec{\psi}}}{d\vec{f}} \right) \\
&= T^{-1}(\vec{\psi})\,\frac{d\dot{\vec{\omega}}}{d\vec{f}},
\end{aligned} \tag{A.22}$$

where this is the only term remaining because all other terms do not depend on the rotor thrust forces $\vec{f}$. Taking the derivative of the equation of motion for $\dot{\vec{\omega}}$ with respect to $\vec{f}$ we get:

$$\begin{aligned}
\frac{d}{d\vec{f}}\dot{\vec{\omega}} &= \frac{d}{d\vec{f}}\left(J^{-1}\left(\begin{bmatrix} \sum_{i=1}^{N} & f_i\, y_i \\ \sum_{i=1}^{N} & -f_i\, x_i \\ \sum_{i=1}^{N} & f_i\, c_i \end{bmatrix} - (\vec{\omega} \times J\,\vec{\omega}) \right)\right) \\
&= J^{-1}\frac{d}{d\vec{f}}\left(\begin{bmatrix} \sum_{i=1}^{N} & f_i\, y_i \\ \sum_{i=1}^{N} & -f_i\, x_i \\ \sum_{i=1}^{N} & f_i\, c_i \end{bmatrix} \right) - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \\
&= J^{-1} \begin{bmatrix} y_1 & \cdots & y_N \\ -x_1 & \cdots & -x_N \\ c_1 & \cdots & c_N \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} ,
\end{aligned} \tag{A.23}$$

where (x_i, y_i) is the location of rotor i relative of the vehicles center of gravity, and c_i is the torque to drag ratio of rotor i with the sign indicating the direction of rotation. At equilibrium we have that $\vec{\psi} = 0$ and hence that $T^{-1}(\vec{\psi}) = I_{3\times3}$. Hence this block of the B matrix is:

$$\left.\frac{d}{d\vec{f}}\ddot{\vec{\psi}}\right|_{\gamma=\beta=\alpha=0} = J^{-1}\begin{bmatrix} y_1 & \cdots & y_N \\ -x_1 & \cdots & -x_N \\ c_1 & \cdots & c_N \end{bmatrix} \tag{A.24}$$

A.6 Linear System about hover for the full state

Combing the above derivations, the linearisation of the full non-linear, continuous-time equations of motion about the hover equilibrium condition is:

$$\begin{bmatrix} \Delta\dot{\vec{p}} \\ \Delta\ddot{\vec{p}} \\ \Delta\dot{\vec{\psi}} \\ \Delta\ddot{\vec{\psi}} \end{bmatrix} = A_{\text{hover}} \begin{bmatrix} \Delta\vec{p} \\ \Delta\dot{\vec{p}} \\ \Delta\vec{\psi} \\ \Delta\dot{\vec{\psi}} \end{bmatrix} + B_{\text{hover}} \begin{bmatrix} f_1 \\ \vdots \\ f_N \end{bmatrix}, \tag{A.25}$$

$$\underbrace{\begin{bmatrix} 0&0&0&1&0&0&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&g&0&0&0&0 \\ 0&0&0&0&0&0&-g&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&0&0&1 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0&0&0 \end{bmatrix}}_{A_{\text{hover}}} \quad \underbrace{\begin{bmatrix} 0 \;\cdots\; 0 \\ 0 \;\cdots\; 0 \\ 0 \;\cdots\; 0 \\ 0 \;\cdots\; 0 \\ 0 \;\cdots\; 0 \\ \frac{1}{m} \;\cdots\; \frac{1}{m} \\ 0 \;\cdots\; 0 \\ 0 \;\cdots\; 0 \\ 0 \;\cdots\; 0 \\ J^{-1}\begin{bmatrix} y_1 & \cdots & y_N \\ -x_1 & \cdots & -x_N \\ c_1 & \cdots & c_N \end{bmatrix} \end{bmatrix}}_{B_{\text{hover}}}$$

When the actuators of total thrust and the three body torques are considered as inputs to the system, then the linearisation about hover is readily derived as:

$$\begin{bmatrix} \Delta\dot{\vec{p}} \\ \Delta\ddot{\vec{p}} \\ \Delta\dot{\vec{\psi}} \\ \Delta\ddot{\vec{\psi}} \end{bmatrix} = A_{\text{hover}} \begin{bmatrix} \Delta\vec{p} \\ \Delta\dot{\vec{p}} \\ \Delta\vec{\psi} \\ \Delta\dot{\vec{\psi}} \end{bmatrix} + B_{\text{hover}} \begin{bmatrix} f_{\text{total}} \\ \tau_x^{(\text{B})} \\ \tau_y^{(\text{B})} \\ \tau_z^{(\text{B})} \end{bmatrix}, \tag{A.26}$$

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{A_{\text{hover}}} \quad \underbrace{\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & & & \\ 0 & & J^{-1} & \\ 0 & & & \end{bmatrix}}_{B_{\text{hover}}}$$