TÉCNICO LISBOA

# Incremental Nonlinear Dynamic Inversion applied to Quadrotor UAV Control

**Ricardo Filipe Encarnação Almeida**

Thesis to obtain the Master of Science Degree in

## Aerospace Engineering

Supervisor: Prof. José Raúl Carreira Azinheira

### Examination Committee

Chairperson: Prof. José Fernando Alves da Silva
Supervisor: Prof. José Raúl Carreira Azinheira
Member of the Committee: Prof. Alexandra Bento Moutinho

**November 2017**

To my mum, for whom I have the greatest admiration

# Acknowledgments

The conclusion of this academic stage would have not been possible without the help of a few very special people. In first place I would like to thank my supervisor, Prof. José Azinheira, for sharing with me his knowledge about the topic and for all the helpful advise that he gave me.

I also want to thank Afonso, for all the useful ideas that arose from the several conversations we had for this thesis, as well as for all the learning that I took from the common projects and study times that we had together in Delft.

I want to thank this Master's Degree for the opportunity of encountering some very interesting people, to which I have the pleasure of calling friends.

At last, I would like to express my deepest gratitude to my family, for the constant and very important support that they provided during the last five years.

# Resumo

O trabalho de pesquisa tem como fim o desenvolvimento de uma lei de controlo utilizando o método de Inversão de Dinâmica Não Linear (NDI) em ambas as variantes, clássica e incremental. Em primeiro lugar é apresentada uma revisão de literatura relevante relacionada com controlo de multirotores, bem como a teoria por detrás do NDI, seguidas pela apresentação de um modelo para o quadrirotor. Estas três componentes serviram como bases para a implementação de um controlador de atitude e velocidade vertical utilizando ambas as técnicas, NDI clássico e incremental. No controlador incremental a estimativa da derivada de estado é feita com um filtro passa banda de segunda ordem, que não só estima a derivada, mas também filtra algum ruído de alta frequência oriundo das medidas dos sensores. A plataforma de simulação foi construída em *MATLAB/Simulink*, incluindo um modelo do quadrirotor e seus sensores. Ambas as leis de controlo foram comparadas com um controlador base, pertencente ao sistema de piloto automático Ardupilot, que utiliza controladores PID, provando a viabilidade destas técnicas não lineares para o controlo de um quadrirotor. A versão incremental foi posteriormente implementada em código *Ardupilot* e comparada com o controlador original desta plataforma no ambiente de simulação incluído no *Ardupilot* (*Software in the Loop (SITL)*). Os resultados da simulação provam que o desempenho desta nova técnica de controlo é semelhante ao do controlador original, apresentando como vantagens: correr a uma frequência inferior, o que requere menos poder computacional numa aplicação de tempo real; apresenta um número consideravelmente menor de parâmetros a ajustar; posui robustez a incertezas de modelo.

**Palavras-chave:** Inversão Incremental de Dinâmica Não Linear, Quadrirotor, Controlo não linear, Ardupilot, Controlo de atitude

# Abstract

The research work aims to develop a feasible control law using Nonlinear Dynamic Inversion (NDI) methodology, both in its classical and incremental variants. In first place a review of relevant literature related with nonlinear multirotor control is presented, as well as a background theory in NDI, followed by a modulation of the quadrotor. This three components served as a basis for the implementation of an attitude and vertical velocity controller using both techniques, NDI and Incremental Nonlinear Dynamic Inversion (INDI). In INDI controller the state derivate estimate was performed with a second order bandpass filter, that not only estimates the derivative but also filters part of the high frequency noise from sensor measurements. A simulation platform was built in MATLAB/Simulink, including a model of the quadrotor and of its sensors. Both control laws were compared with a baseline controller from Ardupilot autopilot system that uses PID loops, proving the feasibility of this methods for quadrotor control. The INDI version of the controller was then implemented in Ardupilot code and compared with the original controller from this platform in the simulation environment of Ardupilot (SITL). The simulation results proved that the performance of this new control technique can match the original controller, having the advantages of: running at lower frequency, which requires a smaller computational power in real time applications; presenting a considerably smaller number of parameters to be adjusted; having robustness to model uncertainties.



**Keywords:** Incremental Nonlinear Dynamic Inversion, Quadrotor, Nonlinear control, Ardupilot, Attitude control

# Contents

# List of Tables

# List of Figures

# Nomenclature

### Greek symbols

| | |
|---|---|
| $\alpha$ | Linear controller parameter. |
| $\beta$ | Linear controller parameter. |
| $\nu$ | Virtual input. |
| $\omega$ | Angular velocity vector. |
| $\phi$ | Roll angle. |
| $\psi$ | Yaw angle. |
| $\theta$ | Pitch angle. |
| $\omega_n$ | Second order system natural frequency. |
| $\tau$ | First order system time constant. |
| $\xi$ | Second order system damping coefficient. |

### Roman symbols

| | |
|---|---|
| $\mathbf{I}$ | Inertia matrix. |
| $\mathbf{R_i^b}$ | Direct Cosine Matrix. |
| $\mathbf{u}$ | Motor input vector. |
| $\mathbf{V}$ | Velocity vector. |
| $\mathbf{x}$ | State vector. |
| $C_{D_0}$ | Aerodynamic drag coefficient. |
| $D$ | Aerodynamic drag force. |
| $F$ | Sum of forces applied to the system. |
| $F_g$ | Gravity force. |
| $F_i$ | Thrust force from propeller $i$. |

$F_t$ Sum of thrust forces.

$g$ Gravity acceleration.

$I_x$ $x$ axis moment of inertia.

$I_y$ $y$ axis moment of inertia.

$I_z$ $z$ axis moment of inertia.

$K_d$ Derivative gain.

$K_i$ Integral gain.

$K_m$ Propeller proportional gain for thrust moment.

$K_p$ Proportional gain.

$K_t$ Propeller proportional gain for thrust force.

$K_{\dot{\psi}}$ Yaw rate controller proportional gain.

$K_{v_z}$ Vertical velocity controller proportional gain.

$l$ Arm length of quadrotor.

$M$ Sum of moments applied to the system.

$m$ Mass.

$p, q, r$ Body frame angular velocity components.

$t_s$ Second order system sampling time.

$u, v, w$ Body frame velocity components.

$V_a$ Air velocity.

$V_w$ Wind velocity.

$v_x, v_y, v_z$ Inertial frame velocity components.

$x, y, z$ Position vector components.

$T_s$ Sampling time.

**Subscripts**

$b$ Body frame of reference.

$d$ Desired value.

**Superscripts**

$'$ First order time derivative.

| | |
|---|---|
| $''$ | Second order time derivative. |
| $(n)$ | Nth order time derivative. |
| $-1$ | Inverse. |
| $T$ | Transpose. |

**Other Notation**

| | |
|---|---|
| $\frac{d}{dt}$ | First order time derivative. |
| $\dot{x}$ | First order time derivative of x. |
| $\|.\|$ | Euclidean norm. |
| $\nabla$ | Gradient operator. |
| $\times$ | Cross product operator. |

# Acronyms

| | |
|---|---|
| **APM** | Ardupilot Mega |
| **BKS** | Backstepping |
| **DCM** | Direct Cosine Matrix |
| **EKF** | Extended Kalman Filter |
| **GCS** | Ground Control Station |
| **IBKS** | Incremental Backstepping |
| **INDI** | Incremental Nonlinear Dynamic Inversion |
| **LFR** | Linear Fractional Representation |
| **LLA** | Latitude-Longitude-Altitude |
| **LQR** | Linear Quadratic Regulator |
| **MIMO** | Multiple Input Multiple Output |
| **NDI** | Nonlinear Dynamic Inversion |
| **NED** | North-East-Down |
| **PINDI** | Predictive Incremental Nonlinear Dynamic Inversion |
| **RASCLE** | Robustness Analysis for Simulation based Control Law Evaluation |
| **SI** | International System of Units (from the French *Système international (d'unités)*) |
| **SISO** | Single Input Single Output |
| **SITL** | Software in the Loop |

**SMC** Sliding Mode Control

**UAV** Unmanned Aerial Vehicle

**VTOL** Vertical Takeoff and Landing

# Chapter 1

# Introduction

This chapter introduces the research work on the topic of ”Incremental Nonlinear Dynamic Inversion applied to Quadrotor UAV Control”. The introduction presents the motivation behind the relevance of this topic, a topic overview, the objectives that the thesis aims to accomplish and an outline of the chapters that constitute it.

## 1.1 Motivation

In recent decades the interest in Unmanned Aerial Vehicles (UAVs) has considerably increased, due to the ability of executing their actions without an onboard pilot, being controlled by an airborne control and guidance system, commonly known as an autopilot, or even remotely. Nowadays a combination of airborne control and remote control is commonly used, with the autopilot addressing the problem of keeping the UAV stable while a remote human pilot navigates a certain trajectory. Initial applications were exclusively related with fixed wing configurations used for military operations, since they greatly reduce the risk of human loss and reach the objectives of the operations with higher efficiency.

More recently multirotor UAVs, also known as multicopters, have become a relevant solution for performing a wide range of applications, thanks to their Vertical Takeoff and Landing (VTOL) and high manoeuvrability capabilities. With these characteristics it is possible to operate them in scenarios with spacial constraints where fixed wing configurations cannot fly, namely, in indoor operations. The most common multirotor configuration is the quadrotor, with four motors and propellers distributed in a square shape. Other configurations are also available, such as tricopters, hexacopters or even octocopters.

Among the numerous applications in which a multicopter can be used one can stand out two categories, namely, the applications where a multicopter is used with a camera, capturing images and video, and the applications where it is used as a transportation device.

The first category includes search and rescue operations [Bowers, 2017], where they can search a large area much faster than humans and with a fraction of the cost of using an helicopter. The camera is also used in agriculture [Mazur, 2017], where multicopters can monitor the state of crops, detecting diseases or damages provoked by bad weather conditions. Another relevant application is related with

their possibility of taking aerial footage for cinema, commercials, music concerts and festivals, or even a large variety of sports [Verrier, 2017]. In this scenarios multirotors are useful due to their ability to film at different angles and perspectives from traditional equipment. Some applications are also related with monitoring the state of large constructions, such as bridges, pipelines, wind turbines and buildings [Sanchez-Cuevas et al., 2017].

In the second category one can include the transportation of medical supplies or vital goods. This kind of UAVs are being used to transport donated organs much faster than by ambulance, allowing the transportation to further locations, which can in some scenarios be the only way of saving patients lives [Strange, 2017]. Multirotors are also starting to be used in package deliveries. Amazon presented in past June their vision for the future in this field, where they aim to build completely autonomous structures similar to beehives where the multicopters will charge and grab the packages to be delivered [Vincent, 2017].

All these applications can present several challenges in terms of controlling quadrotor UAVs, due to the high number of unpredicted disturbances that can be present in some flight situations. Since the control system is usually responsible for stabilizing the quadrotor, with a pilot only having to navigate a given trajectory, a control system that ensures good stabilization performance in the presence of disturbances must be considered as an important objective to be achieved.

Some control systems have full control over the quadrotor, i.e. they possess a completely autonomous control system that does not require a human pilot. In this case the problem of providing a control system that guarantees good performance and robustness to disturbances is even more important.

Besides being able to cope with the disturbances, the control system must also be capable of providing a flight envelope that includes some more aggressive manoeuvres, that may be required during one of the applications that are described above. These problems were addressed with several different methodologies, that will be described and compared in the following section.

## 1.2 Topic Overview

The first solutions for quadrotor control systems use linear controllers, due to their simplicity in implementation. However a quadrotor is an underactuated six degree of freedom nonlinear system, and linear control can only be applied near a specific stable flight condition, i.e. a trimming point. To extend the flight envelope one can design several linear controllers, which one operating at a specific trimming point, including also some functions that choose the best controller for the current flight condition. In other versions these functions can interpolate between the controllers [Stilwell and J. Rugh, 1998]. This technique is known as gain scheduling ([Oosterom and Babuška, 2006], [Milhim, 2010], [Poksawat et al., 2017]) and, even though it has a simple implementation, its design represents a time consuming process without stability guarantees, so that alternatives have been proposed using inherent nonlinear control design methods in recent years, that would provide robust solutions to quadrotor control.

Some of the most common nonlinear control methods are Nonlinear Dynamic Inversion (NDI) ([Krener,

1975], [Brockett, 1978], [Reiner et al., 1996], [Juliana et al., 2005], [Lombaerts et al., 2010]), Backstepping (BKS) ([Kanellakopoulos et al., 1991], [Steinberg and Page, 1998], [Farrell et al., 2009], [Dong et al., 2012], [Gong et al., 2012], [Avram et al., 2016]) and Sliding Mode Control (SMC) ([Wang et al., 2016], [Lopez et al., 2016], [Salazar et al., 2014], [Mercado et al., 2013], [Li et al., 2016], [Chen et al., 2016]). The first two also have incremental variants known as Incremental Nonlinear Dynamic Inversion (INDI) ([Smith, 1998], [Bacon and Ostroff, 2000], [Sieberling et al., 2010], [Azinheira et al., 2015], [Smeur et al., 2015], [Acquatella et al., 2017]) and Incremental Backstepping (IBKS) ([Acquatella et al., 2013], [Lu et al., 2015], [Falconí et al., 2016], [van Gils et al., 2016]). With BKS one can prove the stability of the designed controller, which is not possible if NDI is used, but the greater complexity of the design process makes NDI a more interesting approach to obtain a simple and robust nonlinear controller for a quadrotor. The latter technique, SMC, produces a robust controller with good overall performance, but it obtains this result at the cost of a high control action, and therefore a large battery consumption. Given that battery efficiency is one of the biggest concerns for multirotor UAVs, SMC does not seem an interesting control technique for this type of UAVs.

In conclusion, NDI and INDI seem promising control techniques to solve the problem of quadrotor UAV control.

## 1.3 Objectives

The main objectives of this thesis are related with the implementation of a nonlinear controller for the purpose of controlling a quadrotor. In order to compare the new method with existing and validated control techniques Ardupilot will be used, which is an autopilot platform that has been utilized by enthusiasts and professionals in several UAV applications. For the experimental validation tests one will use an UAVision UX-Spyro quadrotor. The following proposal was made to UAVision: the company provides the quadrotor for experimental validation tests and, in exchange, they receive a developed and implemented nonlinear control algorithm with robust properties and a simpler design (i.e., less tuning parameters). In an early stage of the development, the control solution was analyzed and validated using the MATLAB/Simulink simulation environment. Considering the information of the previous sentences, it is possible to define the objectives of this thesis as:

- Develop a nonlinear control algorithm for UX-Spyro quadrotor;
- Analyze and validate the control solution in a MATLAB/Simulink environment;
- Adapt the control algorithm to Ardupilot and validate the implementation in its simulation platform;
- Validate the control solution with experimental results;

Unfortunately, the experimental tests were not done because UAVision was not available to perform them before the delivery of this document.

## 1.4 Thesis Outline

In this final section a brief explanation about the structure and content of the chapters that compose this thesis are presented.

Chapter 2 contains a review of relevant literature about the topic, including gain scheduling, as well as the main nonlinear control design methods, namely, NDI and INDI, BKS and IBKS and finally SMC. After presenting a general overview on the methods applied to quadrotor control it is important to focus on the chosen nonlinear technique.

Therefore, chapter 3 provides a background on both classical and incremental variants of Nonlinear Dynamic Inversion, concluding with a brief comparison between the advantages and disadvantages that each of them possesses.

A quadrotor model is developed in the first section of chapter 4. The remainder of the chapter presents an implementation design for an attitude and vertical velocity controller based on both NDI and INDI methods, including also an explanation about the position control from Ardupilot that will be used.

A MATLAB/Simulink implementation for both control variants is depicted in chapter 5, including a modelling of sensors, which is special relevant for INDI technique. Several simulation results show the performance of new control methods, comparing them with Ardupilot original controller. A selection of some parameters is also done using parametric variations of them, as well as some tests on the robustness of the control solution. The chapter concludes with simulation results verifying the performance of the overall control system, alongside with the position control from Ardupilot.

The MATLAB/Simulink controller version is used as a basis for the practical implementation in the Ardupilot platform, which is presented in chapter 6. The chapter includes a brief explanation of the platform and its simulation setup, as well as the adaptations that were necessary to be done from MATLAB/Simulink implementation. It concludes with results verifying the performance of the control method.

Finally, chapter 7 presents the conclusions that can be taken from this thesis, alongside some suggestions for future work in the topic.

# Chapter 2

# State of the Art on Multirotor UAV Control

The simplest form of controlling a system is using linear controllers. However, aircraft dynamics are intrinsically nonlinear and therefore this form of control does not meet the design requirements for the total flight envelope, only working around a certain flight condition (i.e., around the trim point used to linearize the system). Thus, two different approaches are usually followed: gain scheduling of linear controllers or nonlinear control design.

This chapter presents the different type of controllers available in the literature, including a brief explanation of their main principles, as well as the advantages and disadvantages present in each control methodology.

## 2.1 Gain Scheduling

In gain scheduling, several linear controllers are designed around different trim points, with the objective of extending the flight envelope. This strategy has been largely used in the aerospace industry, since it provides robust properties with an easy design and implementation [Milhim, 2010]. In addition to the design of different linear controllers, gain scheduling also requires functions that interpolate between the different designed controllers. The selection of operating points and the design of the interpolation scheme remains a time consuming process [Oosterom and Babuška, 2006], that also relies on the experience and skill of the design engineer, which often leads to more operating points than necessary. Another disadvantage is a consequence of the interpolated transitions between operating points, making impossible a formal stability proof of the overall controller.

A fuzzy control approach to design these functions, including pilot in the loop tests in the validation process, is presented in [Oosterom and Babuška, 2006]. The design of an automated gain scheduled controller using a relay feedback test in a wind tunnel is available in [Poksawat et al., 2017].

## 2.2 Nonlinear Control Design

### 2.2.1 Nonlinear Dynamic Inversion and Variants

Nonlinear Dynamic Inversion theoretical development dates back from the seventies with the works of Krener and Brockett [Krener, 1975], [Brockett, 1978]. This methodology uses an exact model of the system to linearize the dynamics of an aircraft, resulting in the same linear system for every aircraft, in theory, allowing the usage of standard linear control techniques (e.g. a PID controller) for stabilization or reference tracking. A major disadvantage relies with the necessity of a complete aerodynamic model of the aircraft, including all system uncertainties. This requirement can result in a loss of performance, or even instability, in the presence of a model mismatch.

In order to increase the robustness to parameter uncertainties, NDI has been used coupled with a structured singular value synthesis [Reiner et al., 1996], and the uncertainties have also been described using a Linear Fractional Representation (LFR). The latter method was applied in a reentry vehicle flying into the atmosphere in [Juliana et al., 2005]. Another strategy to enhance a Nonlinear Dynamic Inversion (NDI) controller, reducing its sensitivity to modeling errors, consists in the use of an online physical model identification technique. This type of technique is used in [Lombaerts et al., 2010] to estimate a new Boeing 747 model during a failure scenario, creating in this manner a fault tolerant controller.

In the late nineties an incremental approach of NDI that does not require the knowledge of state-only dependent dynamics, including kinematics and the gravitational component of the aerodynamic forces, was introduced as Simplified NDI by [Smith, 1998], being currently known in the literature as Incremental Nonlinear Dynamic Inversion (INDI). This variant used instead the feedback of angular accelerations, eliminating sensitivity to model mismatch when compared with conventional nonlinear dynamic inversion [Sieberling et al., 2010]. The angular acceleration feedback approach presents the disadvantage of being sensible to sensor measurement time delays. Several methods have been used for estimating the angular accelerations, including linear accelerometer measurements coupled with differentiated rate signals and least squares estimation [Bacon and Ostroff, 2000]. The robustness of the control system stability was then evaluated using Robustness Analysis for Simulation based Control Law Evaluation (RASCLE) experimental simulation tool, developed at NASA Langley Research Center. Another estimation technique is introduced by [Sieberling et al., 2010], consisting in a linear predictive filter, which aims to correct the sensor time delay problems. Together with INDI, this technique is known as Predictive Incremental Nonlinear Dynamic Inversion (PINDI). A differentiation of the angular rates, followed by a second order low-pass filtering to reduce the amplification of noise occurred during the differentiation, is used in [Smeur et al., 2015]. Stability and robustness of INDI sensor based approach were assessed with wind disturbance and path following loop simulation tests [Azinheira et al., 2015]. A systematic method to design the outer loop PI(D) linear controllers in a nonlinear flight control system via INDI has also been designed [Acquatella et al., 2017].

Even though the model parameters necessary in INDI are greatly reduced if compared with regular NDI, its manual estimation and tuning can represent a challenge, so online adaptive estimation tech-

niques have also been used. The most common technique is the least squares solution [Smeur et al., 2015]. The latter also performed experimental tests in a Parrot Bebop Drone regarding disturbance rejection, performance and parameter adaption.

### 2.2.2 Backstepping and Variants

Backstepping is a nonlinear control design approach that stands on Lyapunov stability theory, contrary to NDI. This represents an advantage in terms of control law certification, since global asymptotic stability can be proved [van Gils et al., 2016]. Backstepping (BKS) presents another advantage relative to NDI, namely, it is flexible with which nonlinearities can be canceled. For instance, if the system presents nonlinearities that have a stabilizing effect then one can choose to keep them, only canceling the remaining nonlinearities.

Its development started in the early nineties with [Kanellakopoulos et al., 1991], however, this initial approach required overparameterization, resulting in systems of higher order that were not practical for the majority of flight control problems, according to [Steinberg and Page, 1998]. Later developments eliminated the overparameterization with the introduction of tuning functions, at the cost of an increase in computational complexity with the order of the system, due to the need of computing the tuning functions derivatives analytically. Command filters have been used to prevent this computation during the design process [Farrell et al., 2009], [Dong et al., 2012], [Gong et al., 2012].

To reduce the dependence from the system model, Incremental Backstepping (IBKS) was introduced by [Acquatella et al., 2013], requiring only the control effectiveness part. This approach also uses acceleration estimates to allow the design of the increments in the control action, similarly to INDI. IBKS robustness to uncertainties in the control effectiveness term is tested in [Lu et al., 2015]. The paper also suggests two methods to increase the robustness of IBKS, validating them with simulation examples. A fault tolerant controller for a Hexarotor system using IBKS was designed and experimentally tested in an outdoor environment by [Falconí et al., 2016].

An adaptive fault tolerant controller using backstepping theory is developed in [Avram et al., 2016], where actuation faults are simulated by losses of effectiveness in a motor of a quadrotor Unmanned Aerial Vehicle (UAV) during experimental tests, confirming the effectiveness of the proposed controller. An adaptive IBKS controller was developed for an F16 model in [van Gils et al., 2016].

### 2.2.3 Sliding Mode Control

Another nonlinear control design technique commonly used in multirotor control problems is Sliding Mode Control (SMC), which uses discontinuous feedback control laws to force the system states to reach a desired sliding surface. Signum functions are used to choose between the different feedback control laws. The advantage of this control technique is its insensitivity to model errors, parametric uncertainties and other disturbances, leading to a robust controller with good performance. However, there are disadvantages related with two interconnected phenomena, namely, chattering and high activity of the control action [Lopez et al., 2016]. In the literature signum functions are commonly replaced by

saturations, in order to reduce the chattering phenomena [Li et al., 2016], [Wang et al., 2016].

Trajectory tracking using a Sliding Mode Control (SMC) algorithm is achieved in [Salazar et al., 2014] for a mini quadrotor UAV system, in a simulation environment including external disturbances in the inputs, to test stability robustness. SMC has also been used for altitude tracking in the presence of uncertainties like a wind gust [Lopez et al., 2016]. Trajectory tracking and flight formation problems, in a leader-follower scheme, are solved using two different SMC controllers in [Mercado et al., 2013]. The latter also includes experimental results validating the approach.

Some literature also compared and combined SMC with other control methods. In [Chen et al., 2016] it is used for attitude tracking, with the outer loop of position tracking being solved with BKS approach. A stability analysis of the overall system, using Lyapunov stability theory, was presented. Simulation results show that robustness and tracking performance were achieved. Altitude controllers using nonlinear SMC and BKS, as well as Linear Quadratic Regulator (LQR), are tested in an experimental environment [Wang et al., 2016]. The results show better control performance for SMC and BKS. Finally, sliding mode has also been compared with NDI in [Li et al., 2016], where it is concluded, with simulation results, that SMC provides better robustness with disturbances whereas NDI has advantage in control parameter selection.

# Chapter 3

# Nonlinear Dynamic Inversion (NDI)

Nonlinear Dynamic Inversion (NDI) is a control method developed in the late 1970's to overcome the limitations of conventional linear techniques. This chapter starts with a description of NDI basics for a Single Input Single Output (SISO) system in section 3.1, followed by the procedure of input-output linearization in 3.2. In 3.3 an extension to Multiple Input Multiple Output (MIMO) systems is presented. The description of the Incremental Nonlinear Dynamic Inversion (INDI) is explained in section 3.4. Finally, a small comparison between them is presented in section 3.5.

The development of this chapter is mainly based on the works of [Simplício, 2011], [Acquatella, 2011], [da Silva, 2015], [Chu, 2016] and [Slotine and Li, 1991].

## 3.1 NDI Basics

The intuitive idea of NDI consists in the cancellation of the nonlinearities in a system, resulting in a linear structure for the closed loop dynamics. Hence, conventional linear control techniques can be designed so that a desired dynamics is achieved for the closed loop system.

This concept can be simply understood when applied to nonlinear SISO systems in the so called companion form or controllability canonical form [Slotine and Li, 1991]. A SISO system is in the companion form if its dynamics can be represented by:

$$x^{(n)} = f(\mathbf{x}) + g(\mathbf{x})u \tag{3.1}$$

where $x \in \mathbb{R}$ represents the scalar output; $u \in \mathbb{R}$ is the scalar control input; $\mathbf{x} = [x,\ \dot{x},\ \ldots,\ x^{(n-1)}]^T$ is the state vector and $f(\mathbf{x})$ and $g(\mathbf{x})$ are nonlinear functions, from $\mathbb{R}^n$ to $\mathbb{R}$. Equation (3.1) can be expressed in a state-space representation, with $x = x_1$, $\dot{x} = x_2$, ..., etc., as in Equation (3.2).

$$\frac{\mathrm{d}}{\mathrm{d}t}\begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} x_2 \\ \vdots \\ x_n \\ f(\mathbf{x}) + g(\mathbf{x})u \end{bmatrix} \tag{3.2}$$

For a system in the companion form (Equation (3.2)), the nonlinearities of the system can be canceled introducing the virtual control input $\nu$, defined as:

$$\nu = \frac{\mathrm{d}x_n}{\mathrm{d}t} = f(\mathbf{x}) + g(\mathbf{x})u \tag{3.3}$$

resulting in a control input $u$ as in Equation (3.4), where it is assumed that $g(\mathbf{x}) \neq 0$.

$$u = g(\mathbf{x})^{-1}\left[\nu - f(\mathbf{x})\right] \tag{3.4}$$

The resulting system is a cascade of $n$ integrators, which can be made exponentially stable (i.e., poles located in the left-half of the complex plane) using linear feedback theory, with the gains chosen so that the closed loop response presents the desired characteristics. For example, using the following control law:

$$\nu = -k_0 x \; - k_1 \dot{x} \; - \; \ldots \; - k_{n-1} x^{(n-1)} \tag{3.5}$$

with $k_0, k_1, ..., k_{n-1}$ representing the proportional gains, leads to an exponentially stable closed loop dynamics composed by multiple integrators, so that $x(t) \to 0$ :

$$x^{(n)} + k_{n-1}x^{(n-1)} + \ldots + k_1\dot{x} + k_0 x = 0 \tag{3.6}$$

A stabilization problem can be described with two control loops: the inner loop performing the dynamic inversion based on Equation (3.4) and the outer control loop based on Equation (3.5).

For tasks involving the tracking of a smooth desired output $x_d(t)$, (i.e., all the derivatives are known and bounded) the control law:

$$\nu = -k_0 e \; - k_1 \dot{e} \; - \; \ldots \; - k_{n-1} e^{(n-1)} \tag{3.7}$$

results in a similar closed loop exponentially stable dynamics for the tracking error $e = x_d(t) - x(t)$:

$$e^{(n)} + k_{n-1}e^{(n-1)} + \ldots + k_1\dot{e} + k_0 e = 0 \tag{3.8}$$

It can be seen in Figure 3.1 that the control system also has a two loop structure: an inner loop performing the dynamic inversion based on Equation (3.4) and the outer control loop based on Equation (3.7). The inner loop is again in a linear multiple integrator form, therefore pole placement, classic control

or other linear control methods can be used to compute the gains that will result in a stable system.

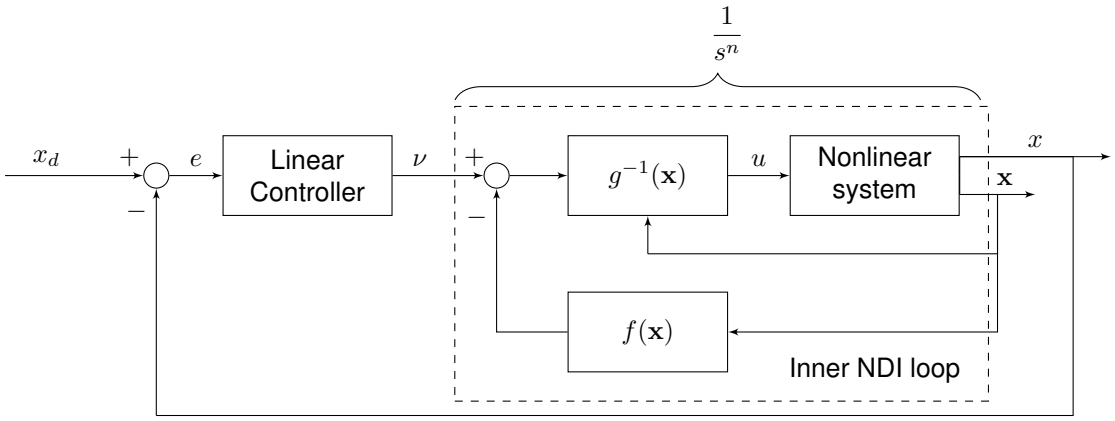



Figure 3.1: Tracking problem using NDI in the inner loop and linear control in the outer loop.

As one can see in Figure 3.1, it is necessary to consider that the model of the system is fully and accurately known, the function $g(\mathbf{x})$ must be invertible and the state vector must also be known, since it will be fed back into $f(\mathbf{x})$ and $g(\mathbf{x})$ functions. This represents a disadvantage in terms of model uncertainties.

The above simple explanation was derived from a SISO system in the so called companion form. The extension to MIMO systems is quite simple and it will be explained in section 3.3. For systems that don't have an explicit relation between the control inputs and the output (i.e., they are not in the companion form), it is necessary to perform first an input-output linearization process [Slotine and Li, 1991].

## 3.2 Input-Output Linearization

The input-output linearization is a transformation used to find an explicit relation between the control inputs and the output, useful in systems that are not in the companion form or in a case where an output function has a nonlinear dependence on the states.

To illustrate this transformation a SISO system with the following state-space representation will be analyzed:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x})u \tag{3.9a}$$

$$y = h(\mathbf{x}) \tag{3.9b}$$

where $\mathbf{x} \in \mathbb{R}^n$ is again the state vector; $u$ and $y$ are the scalar control input and output, respectively; $\mathbf{f}(\mathbf{x})$ and $\mathbf{g}(\mathbf{x})$ are this time nonlinear vector fields from $\mathbb{R}^n$ to $\mathbb{R}$ and $h(\mathbf{x})$ is a nonlinear function from $\mathbb{R}^n$ to $\mathbb{R}$.

This technique is composed by two major steps:

1. Computation of sucessive differentiations to the nonlinear output until an explicit relation is found between the output $y$ and the control input $u$.

2. Define a state local coordinate transformation that converts the system into the companion form, so that a conventional linear controller method can then be designed to achieve the desired tracking performance.

The process will be illustrated by taking the derivative of the output $y$:

$$\dot{y} = \frac{\mathrm{d}y}{\mathrm{d}t} = \frac{\partial h(\mathbf{x})}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \nabla h(\mathbf{x}) \dot{\mathbf{x}} = \nabla h(\mathbf{x}) \left[\mathbf{f}(\mathbf{x}) + \mathbf{g}(\mathbf{x}) u\right] \tag{3.10}$$

where $\nabla h(\mathbf{x})$ denotes the gradient operator of the scalar function $h(\mathbf{x})$ with respect to the state vector $\mathbf{x}$, so it is a row vector of elements $\frac{\partial h(\mathbf{x})}{\partial x_i}$ with $x_i \in \{1,\ 2,\ \ldots,\ n\}$.

Using the notation of Lie derivative from [Slotine and Li, 1991]:

$$L_f h(\mathbf{x}) = \nabla h(\mathbf{x}) \mathbf{f}(\mathbf{x}) \tag{3.11}$$

one can see that it represents the directional derivative of $h(\mathbf{x})$ along the direction of $f(\mathbf{x})$. With this notation it is possible to rewrite Equation (3.10):

$$\dot{y} = L_f h(\mathbf{x}) + L_g h(\mathbf{x}) u \tag{3.12}$$

and, if $L_g h(\mathbf{x}) \neq 0$, one can solve to obtain the control input $u$ of Equation (3.13), if again the derivative of the output is replaced by the virtual control (i.e., $\nu = \dot{y}$).

$$u = L_g h(\mathbf{x})^{-1} \left[\nu - L_f h(\mathbf{x})\right] \tag{3.13}$$

If $L_g h(\mathbf{x}) = 0$ then an explicit relation between $u$ and $y$ cannot be obtained, requiring further differentiation of $y$:

$$\ddot{y} = \frac{\mathrm{d}\dot{y}}{\mathrm{d}t} = \frac{\partial [L_f h(\mathbf{x})]}{\partial \mathbf{x}} \dot{\mathbf{x}} = \nabla \left[L_f h(\mathbf{x})\right] \mathbf{f}(\mathbf{x}) + \nabla \left[L_f h(\mathbf{x})\right] \mathbf{g}(\mathbf{x}) u \tag{3.14}$$

this expression can be simplified if the concept of high order Lie derivatives from [Slotine and Li, 1991] is introduced in a recursive way:

$$L_f^0 h(\mathbf{x}) = h(\mathbf{x}) \tag{3.15a}$$

$$L_f^i h(\mathbf{x}) = L_f \left[L_f^{i-1} h(\mathbf{x})\right] = \nabla \left[L_f^{i-1} h(\mathbf{x})\right] \mathbf{f}(\mathbf{x}), \quad \text{for i} \in \mathbb{N}. \tag{3.15b}$$

Similarly, being $g(\mathbf{x})$ another function:

$$L_g L_f h(\mathbf{x}) = \nabla \left[L_f h(\mathbf{x})\right] \mathbf{g}(\mathbf{x}) \tag{3.16}$$

and Equation (3.14) can be rewritten as Equation (3.17). If $L_g L_f h(\mathbf{x}) \neq 0$ then it is possible to find the control input introducing a virtual control $\nu = \ddot{y}$, as in Equation (3.18).

$$\ddot{y} = L_f^2 h(\mathbf{x}) L_g L_f h(\mathbf{x}) u \tag{3.17}$$

$$u = L_g L_f h(\mathbf{x})^{-1} \left[\nu - L_f^2 h(\mathbf{x})\right] \tag{3.18}$$

Again, if $L_g L_f h(\mathbf{x}) = 0$ it is not possible to obtain an explicit relation between $u$ and $y$ and the differentiation procedure is repeated until an integer number $r$ is found that makes $L_g L_f^{r-1} h(\mathbf{x}) \neq 0$. The generalized differentiated output can be expressed as:

$$y^{(r)} = L_f^r h(\mathbf{x}) + L_g L_f^{r-1} h(\mathbf{x}) u \tag{3.19}$$

so it is straightforward to obtain the new control input as in the previous cases, as one can see in Equation (3.20).

$$u = L_g L_f^{r-1} h(\mathbf{x})^{-1} \left[\nu - L_f^r h(\mathbf{x})\right] \tag{3.20}$$

The previous expression represents the relation between the virtual control and the output of the linearized system, corresponding to a cascade of $r$ integrators (i.e., $\nu = y^{(r)}$). The number $r$ represents the number of differentiations that were necessary until an explicit dependence on the control input appeared, being commonly referred as the relative degree of the system. If this dependence cannot be found the system is not feedback linearizable and a NDI approach cannot be used [Marino, 1986].

### 3.2.1 Internal Dynamics

When an $n^{th}$ order system is feedback linearizable then its relative degree $r \leq n$, existing unobservable (uncontrolled) motions when $r < n$, known as internal dynamics, since they cannot be seen from system input-output relations.

The system can only be effectively controlled if the internal dynamics are stable (bounded) but the stability of internal dynamics of nonlinear systems is difficult to evaluate, because they are in general nonlinear and coupled with external closed loop dynamics [Chu, 2016].

In the case of a linear system the eigenvalues of the internal dynamics correspond to the zeros present in the transfer functions of the control variables, so they are also known as zero dynamics [Enns et al., 1994]. Therefore, the relative degree will correspond to the excess of poles over zeros, being the internal dynamics only stable if the plant zeros are located in the left half-plane (i.e., the plant is a minimum phase system), having reasonable damping ratios, between 0.6 and 1.0, as stated by [Simplício, 2011].

### 3.2.2 State local coordinate transformation

The second step of the input-output linearization is a transformation of the form:

$$\mathbf{z} = \mathbf{\Phi}(\mathbf{x}) = \left[\phi_1(\mathbf{x}), \ldots, \ \phi_r(\mathbf{x}), \ \phi_{r+1}(\mathbf{x}), \ \ldots, \ \phi_n(\mathbf{x})\right]^T \tag{3.21}$$

with the first $r$ components corresponding to the Lie derivatives:

$$\phi_i = L_f^{i-1} h(\mathbf{x}), \quad i = 1, \ldots, r \tag{3.22}$$

The remaining $n - r$ components will represent the internal dynamics and therefore are not affected by the input $u$ explicitly. For the first $r$ components one gets:

$$\dot{z}_i = \frac{\partial \phi_i(\mathbf{x})}{\partial \mathbf{x}} \dot{\mathbf{x}} = \frac{\partial}{\partial \mathbf{x}} \left[ L_f^{i-1} h(\mathbf{x}) \right] \mathbf{f}(\mathbf{x}) + \frac{\partial}{\partial \mathbf{x}} \left[ L_f^{i-1} h(\mathbf{x}) \right] \mathbf{g}(\mathbf{x}) u = L_f^i h(\mathbf{x}) + L_g L_f^{i-1} h(\mathbf{x}) u \tag{3.23}$$

where $L_g L_f^{i-1} h(\mathbf{x}) u \neq 0$ only when $i = r$.

If one sets $\mathbf{x} = \mathbf{\Phi^{-1}}(\mathbf{z})$, $f(\mathbf{z}) = L_f^r h(\mathbf{x})$ and $g(\mathbf{z}) = L_g L_f^{r-1} h(\mathbf{x})$ in Equation (3.23), Equation (3.24) is achieved. For the remaining components (i.e., $i < r$) it is possible to write Equation (3.25).

$$\dot{z}_r = f(\mathbf{z}) + g(\mathbf{z}) u \tag{3.24}$$

$$\dot{z}_i = L_f^i h(\mathbf{x}) = z_{i+1}, \quad i = 1, \ \ldots, \ r - 1 \tag{3.25}$$

The two previous Equations constitute a controllability canonical form. There are still $n - r$ components to be determined, where one can always choose $L_g L_f^{i-1} h(\mathbf{x}) = 0$, obtaining:

$$\dot{z}_i = L_f^i \phi(\mathbf{x}) = q_i(\mathbf{z}), \quad i = r + 1, \ \ldots, \ n \tag{3.26}$$

where $q_i(\mathbf{z})$ does not depend explicitly on the control input, therefore the states $z_{r+1}, \ \ldots, \ z_n$ cannot be controlled. The final state space representation of the system with the coordinate transformation is:

$$\frac{\mathrm{d}}{\mathrm{d}t} \begin{bmatrix} z_1 \\ \vdots \\ z_{r-1} \\ z_r \\ z_{r+1} \\ \vdots \\ z_n \end{bmatrix} = \begin{bmatrix} z_2 \\ \vdots \\ z_r \\ f(\mathbf{z}) + g(\mathbf{z}) u \\ q_{r+1}(\mathbf{z}) \\ \vdots \\ q_n(\mathbf{z}) \end{bmatrix}. \tag{3.27}$$

Similarly to the previous section it is possible to solve for the control input $u$, introducing the virtual control $\nu = z_1^{(r)}$, as in Equation (3.28). The final step is the design of a control law for the linearized system, based on the tracking error $e = y - y_d$, with $y = z_1$. The controller presents the same expression of Equation (3.7). It is important to remind that if there are internal dynamics the controller does not account for them, so it will only be effective if these dynamics are stable.

$$u = g(\mathbf{z})^{-1} \left[ \nu - f(\mathbf{z}) \right] \tag{3.28}$$

## 3.3 Extension to MIMO Systems

The concepts presented in the previous sections for SISO systems, such as input-output linearization, internal dynamics and companion form can be easily adapted to a multivariable NDI scenario. Considering a MIMO system with the number of inputs being equal to the number of outputs (i.e., a square system), one can express an $n^{th}$ order system as:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u} \tag{3.29a}$$

$$\mathbf{y} = \mathbf{h}(\mathbf{x}) \tag{3.29b}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the $n^{th}$ order state vector; $\mathbf{u} \in \mathbb{R}^m$ and $\mathbf{y} \in \mathbb{R}^m$ are the control input and output vectors, respectively; $\mathbf{f}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ are smooth field vectors in $\mathbb{R}^n$ and $\mathbf{G}(\mathbf{x})$ is a square matrix in $\mathbb{R}^n$.

Differentiating a component $y_i$ of the output vector $\mathbf{y}$ one gets:

$$\dot{y_i} = \frac{\partial h_i}{\partial \mathbf{x}} \frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \nabla h_i \left[\mathbf{f}(\mathbf{x}) + \mathbf{G}(\mathbf{x})\mathbf{u}\right] = \nabla h_i \left[\mathbf{f}(\mathbf{x}) + \mathbf{g_1}(\mathbf{x})u_1 + \cdots + \mathbf{g_n}(\mathbf{x})u_n\right] \tag{3.30}$$

where $g_j,\ j = 1,\ \ldots,\ n$ are the smooth field vectors multiplying for the correspondent input $u_j$, hence are given by the columns of $\mathbf{G}(\mathbf{x})$. Using Lie derivative notation:

$$\dot{y_i} = L_f h_i(\mathbf{x}) + \sum_{j=1}^{n} L_{g_j} h_i(\mathbf{x}) u_j \tag{3.31}$$

Similarly to the SISO case an explicit dependence on the input may not appear (i.e., $L_{g_j} h_i(\mathbf{x}) u_j = 0$ for $j = 1,\ \ldots,\ n$). Representing the smallest number of differentiations necessary for an explicit dependence as $r_i$, it is possible to write:

$$y_i^{(r_i)} = L_f^{r_i} h_i(\mathbf{x}) + \sum_{j=1}^{n} L_{g_j} L_f^{r_i - 1} h_i(\mathbf{x}) u_j \tag{3.32}$$

with $r_i$ being the relative degree of the $i^{th}$ output component. The total relative degree of the system will be:

$$r = \sum_{j=1}^{n} r_j \tag{3.33}$$

Once again if the total relative degree is smaller than the total system order, there are internal dynamics that must be bounded to assure an effective control. Collecting all the differentiated outputs $y_i$ one can obtain a similar expression depending on the input $\mathbf{u}$:

$$\begin{bmatrix} y_1^{(r_1)} \\ \vdots \\ y_n^{(r_n)} \end{bmatrix} = \mathbf{a}(\mathbf{x}) + \mathbf{B}(\mathbf{x})\mathbf{u} \tag{3.34}$$

where $\mathbf{a}(\mathbf{x})$ and $\mathbf{B}(\mathbf{x})$ are constituted by the Lie derivatives, represented in Equations (3.36) and (3.37). Replacing the differentiated outputs vector by a virtual control vector $\nu = [\nu_1, \ldots, \nu_n]^T$ one can obtain an expression for the input $\mathbf{u}$:

$$\mathbf{u} = \mathbf{B}(\mathbf{x})^{-\mathbf{1}} \left[\nu - \mathbf{a}(\mathbf{x})\right] \tag{3.35}$$

$$\mathbf{a}(\mathbf{x}) = \begin{bmatrix} L_f^{r_1} h_1(\mathbf{x}) \\ \vdots \\ L_f^{r_n} h_n(\mathbf{x}) \end{bmatrix} \tag{3.36}$$

$$\mathbf{B}(\mathbf{x}) = \begin{bmatrix} L_{g_1} L_f^{r_1-1} h_1(\mathbf{x}) & \ldots & L_{g_n} L_f^{r_1-1} h_1(\mathbf{x}) \\ \vdots & \vdots & \vdots \\ L_{g_1} L_f^{r_n-1} h_n(\mathbf{x}) & \ldots & L_{g_n} L_f^{r_n-1} h_n(\mathbf{x}) \end{bmatrix} \tag{3.37}$$

If the number of inputs is different from the number of outputs, $\mathbf{B}(\mathbf{x})$ is not square and therefore one needs to use a Moore–Penrose pseudoinverse. The closed loop form $y_i^{(r_i)} = \nu_i$ for $i = 1, \ldots, n$ shows that each component is independent and decoupled from the remaining components. Thus, the controllers for each output channel can be designed separately, in order to meet the required tracking performance.

This MIMO extension concludes the derivation of NDI control theory, when one can identify a major drawback, the need for a complete and accurately known model of the system. The next section introduces an incremental variant of the NDI control theory (INDI), where only the dynamics of control derivatives are necessary from the model.

## 3.4 Incremental Nonlinear Dynamic Inversion (INDI)

The concept of INDI was developed to reduce the dependence from an exact model of the system, and consequently improve the robustness to model uncertainties. For this derivation one can consider a general nonlinear MIMO system with the following form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}) \tag{3.38}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the $n^{th}$ order state vector; $\mathbf{u}$ is the control input vector and $\mathbf{f}(\mathbf{x}, \mathbf{u})$ is a nonlinear function. Applying a first order Taylor series expansion, the system can be linearized around the point $(\mathbf{x} = \mathbf{x_0}, \mathbf{u} = \mathbf{u_0})$:

$$\dot{\mathbf{x}} \approx \mathbf{f}(\mathbf{x_0}, \mathbf{u_0}) + \left.\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x_0}, \mathbf{u}=\mathbf{u_0}} (\mathbf{x} - \mathbf{x_0}) + \left.\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{x}=\mathbf{x_0}, \mathbf{u}=\mathbf{u_0}} (\mathbf{u} - \mathbf{u_0}) \tag{3.39}$$

This expression can be further simplified assuming a high sample rate for the system and using the principle that input dynamics are considerably faster than state dynamics, hence $\mathbf{x} - \mathbf{x_0} \approx 0$, obtaining Equation (3.40). Replacing $\mathbf{f}(\mathbf{x_0}, \mathbf{u_0})$ by $\dot{\mathbf{x}}_0$ and $\left.\frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{x}=\mathbf{x_0}, \mathbf{u}=\mathbf{u_0}}$ by $\mathbf{G}(\mathbf{x_0}, \mathbf{u_0})$ in Equation (3.39) one gets:

$$\dot{\mathbf{x}} \approx \dot{\mathbf{x}}_0 + \mathbf{G}(\mathbf{x_0}, \mathbf{u_0}) (\mathbf{u} - \mathbf{u_0}) \tag{3.40}$$

which can be solved for an incremental input form $\Delta\mathbf{u} = \mathbf{u} - \mathbf{u_0}$ if matrix $\mathbf{G}(\mathbf{x_0}, \mathbf{u_0})$ is a square nonsingular matrix (i.e., it can be inverted, and consequently the system has the same number of inputs and states), otherwise control allocation problems may arise and a Moore–Penrose pseudoinverse should be used.

Again if one introduces the virtual control input $\nu = \dot{\mathbf{x}}$, the incremental form is given then by:

$$\Delta\mathbf{u} = \mathbf{G^{-1}}(\mathbf{x_0}, \mathbf{u_0})\,(\nu - \dot{\mathbf{x}}_\mathbf{0}) \tag{3.41}$$

In INDI control theory, $\dot{\mathbf{x}}_\mathbf{0}$ is assumed to be measurable and therefore this method is also denominated sensor based control, in contrast with $\mathbf{f}(\mathbf{x}, \mathbf{u})$ that is an example of model based control. The total control inputs are obtained by adding the previous control input $\mathbf{u_0}$ to $\Delta\mathbf{u}$ (Equation (3.42)). Observing this equation one can see that it represents a discrete integrator, so it will present integrative characteristics. Matrix $\mathbf{G}(\mathbf{x_0}, \mathbf{u_0})$ is referred in the literature ([Chu, 2016]) as control effectiveness matrix, since it includes the relation between changes in the states and control inputs of the system (i.e., control derivatives).

Similarly to NDI the system is now linearized and a linear control law can be designed to meet the tracking or stabilization requirements. A diagram of this approach can be depicted in Figure 3.2.

$$\mathbf{u} = \mathbf{u_0} + \Delta\mathbf{u} \tag{3.42}$$

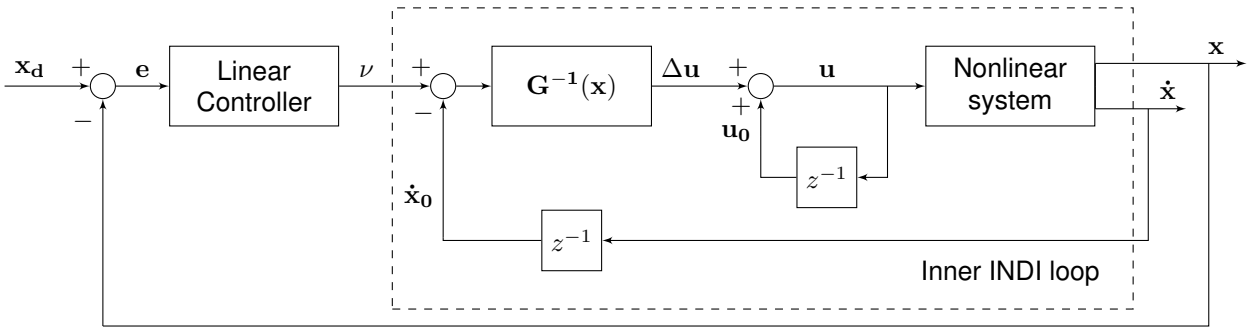



Figure 3.2: Tracking problem using INDI in the inner loop and linear control in the outer loop.

It is worth mentioning that the dependency of plant dynamics is not completely eliminated, since changes in $\mathbf{f}(\mathbf{x}, \mathbf{u})$ are reflected in measurements of $\dot{\mathbf{x}}_0$. The need for measuring $\dot{\mathbf{x}}_\mathbf{0}$ makes this control law more dependent from sensor noise and biases than NDI.

## 3.5 NDI vs INDI Comparison

This chapter presented the derivation of both NDI and INDI control laws, whose control input equations, for a SISO system, are given by Equations (3.43) and (3.44), respectively.

$$u = g(x)^{-1}\left[\nu - f(x)\right] \tag{3.43}$$

$$u = u_0 + g(x)^{-1} [\nu - \dot{x}_0] \tag{3.44}$$

The following conclusions can be drawn from the comparison between NDI and INDI:

- NDI requires the complete knowledge of the system and it is applicable only to systems affine in the control input. Alternatively, INDI only uses the control effectiveness matrix, not requiring a model for $f(x)$. For this reason INDI is expected to be more robust to model uncertainties. Moreover, INDI approach can be used with a general nonlinear system.
- INDI controller needs a high sampling frequency, an actuation dynamics much faster than a system dynamics and the observations of both the state derivatives and the control input signal. These requirements mean that INDI will be more dependent on the accuracy of sensor measurements. Sometimes the state derivatives cannot be measured and therefore they need to be estimated. This represents a disadvantage of this controller, since it will be more dependent from sensor noise, biases and delays. When estimation is needed INDI is also dependent of numerical errors and the noise on the measurements is highly increased.

The following chapter will present the used model for the quadrotor, as well as the implementation of NDI and INDI for controlling the attitude and vertical speed.

# Chapter 4

# Quadrotor Modelling and Controller Implementation

After explaining the theory of Nonlinear Dynamic Inversion (NDI), including its incremental form, this chapter presents the implementation of the controllers in Spyro quadrotor model.

A key step to design an efficient controller is to develop a model of the system to be controlled, therefore section 4.1 details the adopted nonlinear model, including the explanation beyond the selection of this model over others used in literature. Section 4.2 presents the implementation of NDI theory in attitude and vertical velocity tracking controller. The implementation of Incremental Nonlinear Dynamic Inversion (INDI) for the same tracking controller is detailed in section 4.3. Section 4.4 presents the implemented position controller. The chapter concludes with some implementation considerations, including sampling time and state derivatives measurement, among other details.

## 4.1 Quadrotor Model

### 4.1.1 Spyro Quadrotor

The UX-Spyro is the multirotor Unmanned Aerial Vehicle (UAV) produced by UAVision company that is being modeled (Figure 4.1). It runs an adapted version of Ardupilot, an autopilot system (see chapter 6), in a Pixhawk board. Some of its characteristics can be seen in Appendix A.1.

Figure 4.1: Spyro Quadrotor. [source: https://cineaerials.com/]

### 4.1.2 Reference frames

An important aspect in the modelling of the quadrotor is the choice of the body reference frame, since it will influence both the inertia and actuation matrices. In this case a reference frame equal to the one used in Ardupilot was chosen, avoiding compatibility issues during the implementation of the control algorithm in the autopilot system. The arrows around the forces produced in each propeller represent the direction of rotation of that propeller.

For the inertial reference frame a regular North-East-Down (NED) frame was used, since the movement of the earth can be despised in this case. The $x$ and $y$ axes are aligned with the meridian and parallel lines, respectively, and the $z$ axis points to the center of the earth. The reference frames can be depicted in Figure 4.2. It is worth notice that the quadrotor is symmetric relative to the coordinate axes, so the inertial matrix will be a simple diagonal matrix, given that the products of inertia are all zero.

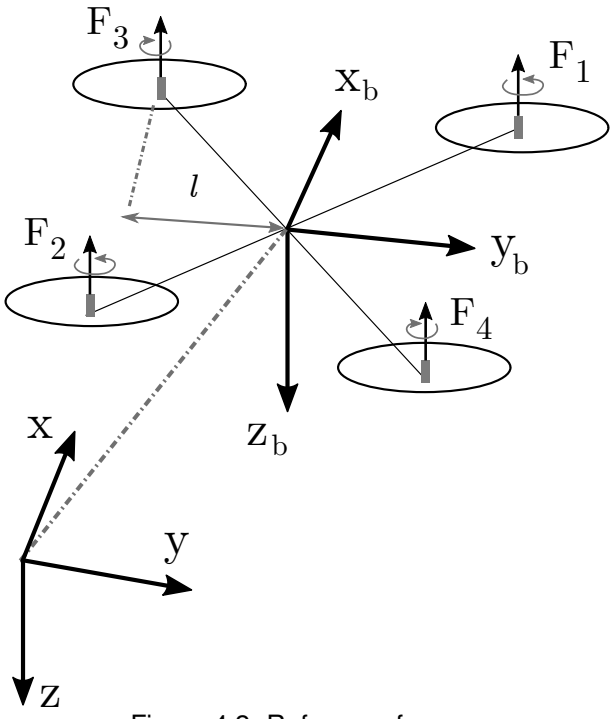



Figure 4.2: Reference frames.

The transformation between both reference frames can be described by $\mathbf{v_b} = \mathbf{R_i^b v_i}$ , with $\mathbf{v_b}$ and $\mathbf{v_i}$ representing a general vector in the body and inertial frames, respectively. The matrix $\mathbf{R_i^b}$ is the Direct Cosine Matrix (DCM), shown in Expression (4.1), where $\phi$, $\theta$ and $\psi$ are the Euler angles roll, pitch and yaw, respectively.

$$\mathbf{R_i^b} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \sin\phi\cos\theta \\ \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \cos\phi\cos\theta \end{bmatrix} \tag{4.1}$$

### 4.1.3 Equations of Motion

Perhaps the most important part of the overall model is the dynamic model, which mostly describes the physical behaviour of the quadrotor. The dynamic model is composed by the Newton-Euler equations of motion, which describe the linear and rotational motions of a rigid body particle. The general form of this set of equations in an inertial frame is represented by Equations (4.2), where $\mathbf{F}$ represents the sum of forces applied to the system, expressed in the inertial frame; $m$ is the mass of the system; $\mathbf{V}$ is the velocity vector in the inertial frame; $\mathbf{M}$ represents the sum of applied moments; $\mathbf{I}$ is the inertia matrix and $\omega$ is the angular velocity vector, also expressed in the inertial frame.

$$\mathbf{F} = \frac{\mathrm{d}}{\mathrm{d}t}(m\mathbf{V}) \tag{4.2a}$$

$$\mathbf{M} = \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{I}\,\omega) \tag{4.2b}$$

In the case of quadrotor control it is more useful to write the moments equation with respect to the body frame, where one must take into account the rotation of the frame in the time derivative, resulting in Equation (4.3) (derivation in Appendix A.2). Subscript $\mathbf{b}$ is included in the variables that are now expressed in the body frame, for the sake of clarity, with math symbol $\times$ representing the cross product. Therefore, the angular velocity vector can be represented by their body components, $\omega_{\mathbf{b}} = (p, q, r)$.

$$\mathbf{M_b} = \mathbf{I_b}\,\dot{\omega}_{\mathbf{b}} + \omega_{\mathbf{b}} \times \mathbf{I_b}\,\omega_{\mathbf{b}} \tag{4.3}$$

### 4.1.4 Modelling of Forces and Moments

To complete the derivation of the equations of motion for the quadrotor is then necessary to model the forces and moments that are acting on the system, including their relation with the inputs of the quadrotor. In this project were considered the two major forces acting on the system, namely, gravity ($F_g = mg$) and the total thrust force applied by the quadrotor propellers ($F_t$). Thrust force always acts on the negative direction of $z_b$ axis and gravity on the positive $z$ direction. The sum of both forces, expressed in the inertial frame, is then given in Equation (4.4). The thrust force is transformed to the inertial frame frame using the inverse of DCM, that is depicted in Equation (4.5).

$$\mathbf{F} = \underbrace{\mathbf{R_b^i}}_{(\mathbf{R_i^b})^{-1}} \begin{bmatrix} 0 \\ 0 \\ -F_t \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{4.4}$$

$$\mathbf{R_b^i} = \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{4.5}$$

The total thrust force is given by the sum of the forces applied in each propeller, so $F_t = \sum_{i=1}^{4} F_i$. Applying this resulting and the transformation it is possible to obtain the force component for each axis, as given in Equation (4.6).

$$\begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = -\left(F_1 + F_2 + F_3 + F_4\right) \begin{bmatrix} \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ \cos\phi\cos\theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{4.6}$$

This model does not include aerodynamic force because its estimation can be a rather complex process that does not compensate in this case, since INDI control approach is a robust method that should overcome its effect. However, if there are problems that require a more complex dynamic model then the inclusion of the aerodynamic effect must be considered.

In the body reference frame (present in Figure 4.2) the moment components are given by Equation (4.7). The $x$ axis component originates a roll moment and the $y$ axis a pitch moment. These two moments are proportional to the thrust forces applied in each propeller, with $l$ being the arm for the moment. For the yaw moment, on the $z$ axis, a relation between the force and the moment must be estimated. For now the proportionality constant will be named $K_m$. To understand the signs in $z$ component one must know that a propeller rotating in a certain direction exerts in the quadrotor a moment in the opposite direction.

$$\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} l\left(-F_1 + F_2 + F_3 - F_4\right) \\ l\left(F_1 - F_2 + F_3 - F_4\right) \\ K_m\left(F_1 + F_2 - F_3 - F_4\right) \end{bmatrix} \tag{4.7}$$

### 4.1.5 Dynamic Model including forces and moments modelling

Applying the force and moment modelling in the equations of motion and replacing both velocity vectors, linear and angular, with their components, as well as including the diagonal inertia matrix components $I_x, I_y$ ans $I_z$ one finally obtains the equations of motion (4.8) and (4.9) for the linear and rotational movements, respectively. This result will then be used to design both NDI and INDI controllers.

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{v}_z \end{bmatrix} = \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \frac{-\left(F_1 + F_2 + F_3 + F_4\right)}{m} \begin{bmatrix} \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ \cos\phi\cos\theta \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{4.8}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}^{-1} \left( \begin{bmatrix} l\left(-F_1 + F_2 + F_3 - F_4\right) \\ l\left(F_1 - F_2 + F_3 - F_4\right) \\ K_m\left(F_1 + F_2 - F_3 - F_4\right) \end{bmatrix} - \begin{bmatrix} qr(I_z - I_y) \\ pr(I_x - I_z) \\ pq(I_y - I_x) \end{bmatrix} \right) \tag{4.9}$$

### 4.1.6 Kinematic model

The kinematic model defines a relation for the rotational movement between the body frame and the inertial frame, in other words it presents a relation (Equation (4.10)) between the body rates $p, q$ and $r$ of the angular velocity vector $\omega_b$ and the Euler rates $\dot{\phi}, \dot{\theta}$ and $\dot{\psi}$.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{4.10}$$

The inverse transformation, from Euler rates to body rates is the following:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \dfrac{\sin\phi}{\cos\theta} & \dfrac{\cos\phi}{\cos\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.11}$$

One can observe that the previous transformation presents a singularity around $\theta = \pm 90^{\text{o}}$. In this case the quadrotor flight capabilities do not require such an aggressive attitude, and therefore this transformation with Euler rates will be used, since it is the most intuitive solution. In flights with aggressive manoeuvres, where pitch angles close to $\pm 90^{\text{o}}$ can be achieved, this transformation will lead to instabilities. To solve this problem one can introduce the transformation to quaternions, instead of Euler rates, losing intuitive results because quaternions do not have a physical meaning.

## 4.2 Implementation of NDI Controller

After presenting the theory behind NDI and the model of the quadrotor it is possible to introduce the implementation of an attitude and vertical velocity controller with the two loop Nonlinear Dynamic Inversion approach. The first step of this implementation consists in the definition of the states and inputs of the inner linearization loop.

### 4.2.1 Inner loop with linearization

Taking into consideration the equations of motion derived in the description of the model, the most appropriate variables for the inner loop state vector derivative($\dot{\mathbf{x}}$) of this particular controller are the body frame angular accelerations ($\dot{p}, \dot{q}$ and $\dot{r}$) to control the attitude and the vertical acceleration in the body frame ($\dot{w}$). The outer loop must then have a linear controller that provides a virtual control $\nu$ for these variables.

For the vertical controller, the acceleration in the inertial frame ($\dot{v}_z$) will be used instead, so that one can develop an altitude controller which corresponds to the second order integral of $\dot{v}_z$, apart from a minus sign because the $z$ axis is pointing down. This reasoning is only similar with $\dot{w}$ if both roll and pitch angles are zero, otherwise altitude will not correspond to the second order integral of body frame vertical acceleration. One could also obtain a virtual control for $\dot{v}_z$ and then transform its value to a reference value for $\dot{w}$ using DCM. This was the first approach used to build both NDI and INDI controllers but when the position controller was included the performance degraded in all control channels, with the existence of oscillations.

The state vector derivative $\dot{\mathbf{x}}$ is shown in expression (4.12), where $T$ is the transpose symbol, stating that column vectors are the ones being used in this thesis. The expression for the state, in this system, can be obtained from the equations of motion (4.8) and (4.9), resulting in a four dimensional expression shown in Equation (4.13).

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{p} & \dot{q} & \dot{r} & \dot{v}_z \end{bmatrix}^T \tag{4.12}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{v}_z \end{bmatrix} = \begin{bmatrix} \frac{l}{I_x}\left(-F_1 + F_2 + F_3 - F_4\right) \\ \frac{l}{I_y}\left(F_1 - F_2 + F_3 - F_4\right) \\ \frac{K_m}{I_z}\left(F_1 + F_2 - F_3 - F_4\right) \\ \frac{-\cos\phi\cos\theta}{m}\left(F_1 + F_2 + F_3 + F_4\right) \end{bmatrix} + \begin{bmatrix} -qr\frac{I_z - I_y}{I_x} \\ -pr\frac{I_x - I_z}{I_y} \\ -pq\frac{I_y - I_x}{I_z} \\ g \end{bmatrix} \tag{4.13}$$

The inputs of the system are a normalized thrust force in each propeller ($T_i$), between 0 and 1, that will be modeled as proportional to the thrust force given by each propeller (Equation (4.14)). Other models in literature, as [da Silva, 2015], consider the thrust force proportional to the square of the angular velocity in each propeller. In this case it did not seem a reasonable approach, since a relation between angular velocity and a normalized thrust force is not known. If a better approximation is needed one can use an identification method, such as a least squares estimation.

$$F_i = K_T T_i, \quad \forall i = 1, \ldots, 4 \tag{4.14}$$

Introducing in (4.13) the new relation with the thrust force and isolating the inputs $T_i$, it is possible to obtain the expressions for $\mathbf{f}(\mathbf{x})$ and $\mathbf{G}(\mathbf{x})$ from Equation (4.15).

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{v}_z \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{-lK_T}{I_x} & \frac{lK_T}{I_x} & \frac{lK_T}{I_x} & \frac{-lK_T}{I_x} \\ \frac{lK_T}{I_y} & -\frac{lK_T}{I_y} & \frac{lK_T}{I_y} & \frac{-lK_T}{I_y} \\ \frac{K_m K_T}{I_z} & \frac{K_m K_T}{I_z} & \frac{-K_m K_T}{I_z} & \frac{-K_m K_T}{I_z} \\ \frac{-\cos\phi\cos\theta}{m} K_T & \frac{-\cos\phi\cos\theta}{m} K_T & \frac{-\cos\phi\cos\theta}{m} K_T & \frac{-\cos\phi\cos\theta}{m} K_T \end{bmatrix}}_{\mathbf{G}(\mathbf{x})} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} + \underbrace{\begin{bmatrix} -qr\frac{I_z - I_y}{I_x} \\ -pr\frac{I_x - I_z}{I_y} \\ -pq\frac{I_y - I_x}{I_z} \\ g \end{bmatrix}}_{\mathbf{f}(\mathbf{x})} \tag{4.15}$$

Both the moments of inertia and mass of the quadrotor can be seen in Appendix A.1. The process of obtaining numerical estimates for both $\mathbf{G}(\mathbf{x})$ and $\mathbf{f}(\mathbf{x})$ will be discussed in chapter 5. After the definition of the states, inputs, $\mathbf{f}(\mathbf{x})$ and $\mathbf{G}(\mathbf{x})$ the step that follows is to define the outputs of the system.

### 4.2.2 Outer loop with Linear Controller

The choice for the inputs of the linear controller coincides with the outputs of the position controller provided in Ardupilot. In this manner one doses not need to transform the output references of the position controller into the input references of the attitude controller. These outputs are the roll and pitch angles, the yaw rate and the vertical velocity in the inertial frame, so that the reference input vector $\mathbf{y} = \left(\phi,\ \theta,\ \dot{\psi},\ v_z\right)^T$.

The last step during the design of the NDI controller consists in the choice for the linear controller. For the roll and pitch angles a second order controller is necessary, given that there are two derivatives to transform an angle into an angular acceleration.

A possible controller may thus be given as in Equations (4.16) for this two channels, where $\dot{p}_d$ and $\dot{q}_d$ are the virtual control inputs for the inner loop; $\phi_d$ and $\theta_d$ are the input references given by the position controller; $\phi, \theta, p$ and $q$ are the roll and pitch angles, as well as the body angular rates in the $x$ and $y$ axes; finally $\alpha$ and $\beta$ are two controller adjustable parameters.

$$\dot{p}_d = \alpha(\phi_d - \phi) + \beta p \tag{4.16a}$$

$$\dot{q}_d = \alpha(\theta_d - \theta) + \beta q \tag{4.16b}$$

Considering small Euler roll and pitch angles, transformation (4.11) allows one to write $\dot{\phi} \approx p$ and $\dot{\theta} \approx q$, leading to $\dot{p} \approx \ddot{\phi}$ and $\dot{q} \approx \ddot{\theta}$. Replacing these approximations in (4.16) results in the following expressions.

$$\ddot{\phi} = \alpha(\phi_d - \phi) + \beta\dot{\phi} \tag{4.17a}$$

$$\ddot{\theta} = \alpha(\theta_d - \theta) + \beta\dot{\theta} \tag{4.17b}$$

Introducing Laplace transform in the previous expressions and rearranging terms one can conclude that these controllers correspond to second order low-pass filters, as the ones presented in (4.19). The

natural frequency of these filters is $\omega_n$ and the damping ratio is represented by $\xi$.

$$\phi s^2 = \alpha(\phi_d - \phi) + \beta\phi s \Leftrightarrow \frac{\phi}{\phi_d} = \frac{\alpha}{s^2 - \beta s + \alpha} \tag{4.18a}$$

$$\theta s^2 = \alpha(\theta_d - \theta) + \beta\theta s \Leftrightarrow \frac{\theta}{\theta_d} = \frac{\alpha}{s^2 - \beta s + \alpha} \tag{4.18b}$$

$$\frac{x}{x_d} = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \Rightarrow \begin{cases} \alpha = \omega_n^2 \\ \beta = -2\xi\omega_n \end{cases} \tag{4.19}$$

In order to understand the influence of $\alpha$ and $\beta$ in the time response of the filter and help in the process of selecting reasonable values for these parameters a MATLAB/Simulink model was developed. The influence of the damping ratio $\xi$ and the settling time $t_s$ (Equation (4.20)) is explained in detail in Appendix A.3. In summary, a higher damping ratio decreases the maximum overshoot and oscillations of the time response, but it also increases the rise time. A smaller settling time is in general preferable, but considerably small settling times may cause an oscillatory response. The values $\xi = 0.9$ and $t_s = 1\ [s]$ were selected as initial estimates.

$$t_{s\ [2\%]} = \frac{4}{\xi\omega_n} \tag{4.20}$$

The selection of both damping ratio and settling time automatically defines the values for the controller parameters $\alpha$ and $\beta$. The obtained values are the following:

$$\begin{cases} \alpha \approx 19.75\ [s^{-2}] \\ \beta = -8\ [s^{-1}] \end{cases} \tag{4.21}$$

The linear controllers for the yaw rate and vertical velocity can be first order systems, since only one derivative exists between inputs given to the controller and the outputs provided by it. Once again one needs to consider small attitude angles, so that $\ddot{\psi} \approx \dot{r}$. The expressions for these controllers can be depicted in the set of Equations (4.22), where $\dot{r}_d$ and $\dot{v}_{z_d}$ are the virtual controls for the inner loop; $\dot{\psi}_d$ and $v_{z_d}$ represent the desired inputs provided by the position controller; $\dot{\psi}$ and $v_z$ are the current yaw rate and vertical velocity in the inertial frame. Finally $K_{\dot{\psi}}$ and $K_{v_z}$ are adjustable gain parameters. Rearranging the terms and introducing Laplace transform these expressions can be identified as first order low-pass filters, which general expression can be depicted in Equation (4.23).

$$\ddot{\psi} \approx \dot{r}_d = K_{\dot{\psi}} \left(\dot{\psi}_d - \dot{\psi}\right) \tag{4.22a}$$

$$\dot{v}_{z_d} = K_{v_z} \left(v_{z_d} - v_z\right) \tag{4.22b}$$

$$\frac{x}{x_d} = \frac{K}{s + K} \tag{4.23}$$

Similarly to the second order systems used to control the roll and pitch angles, a MATLAB/Simulink model of these systems was created, in order to understand the influence of the gains and consequently choose an appropriate value for them. Once again a detailed analysis can be depicted in Appendix A.3. In general the rise time decreases with increasing gain and the oscillations increase together with an increase in gain. Hence, a first choice for the proportional gains is $K_{\dot{\psi}} = K_{v_z} = 2\ [s^{-1}]$. These values can be modified in future simulations and experimental results, if unreasonable performance is achieved with these controllers.

It is possible to group the four linear controllers into a four dimensional linear controller vector with the form of Equation (4.24), where control matrices $\mathbf{K_1}$ and $\mathbf{K_2}$ are introduced.

$$\begin{cases} \dot{p}_d = \alpha\left(\phi_d - \phi\right) + \beta p \\ \dot{q}_d = \alpha\left(\theta_d - \theta\right) + \beta q \\ \dot{r}_d = K_{\dot{\psi}}\left(\dot{\psi}_d - \dot{\psi}\right) \\ \dot{v}_{z_d} = K_{v_z}\left(v_{z_d} - v_z\right) \end{cases} \Rightarrow \begin{bmatrix} \dot{p}_d \\ \dot{q}_d \\ \dot{r}_d \\ \dot{v}_{z_d} \end{bmatrix} = \underbrace{\begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \alpha & 0 & 0 \\ 0 & 0 & K_{\dot{\psi}} & 0 \\ 0 & 0 & 0 & K_{v_z} \end{bmatrix}}_{\mathbf{K_1}} \begin{bmatrix} \phi_d - \phi \\ \theta_d - \theta \\ \dot{\psi}_d - \dot{\psi} \\ v_{z_d} - v_z \end{bmatrix} + \underbrace{\begin{bmatrix} \beta & 0 \\ 0 & \beta \\ 0 & 0 \\ 0 & 0 \end{bmatrix}}_{\mathbf{K_2}} \begin{bmatrix} p \\ q \end{bmatrix} \tag{4.24}$$

The design of the outer loop linear controller vector concludes the implementation of the NDI control strategy in an attitude and vertical velocity controller.

### 4.2.3 NDI controller block diagram

The block diagram of the complete NDI controller can be depicted in Figure 4.3. The diagram includes the results obtained for $\mathbf{f(x)}$, $\mathbf{G(x)}$, $\mathbf{K_1}$ and $\mathbf{K_2}$ in previous sections.

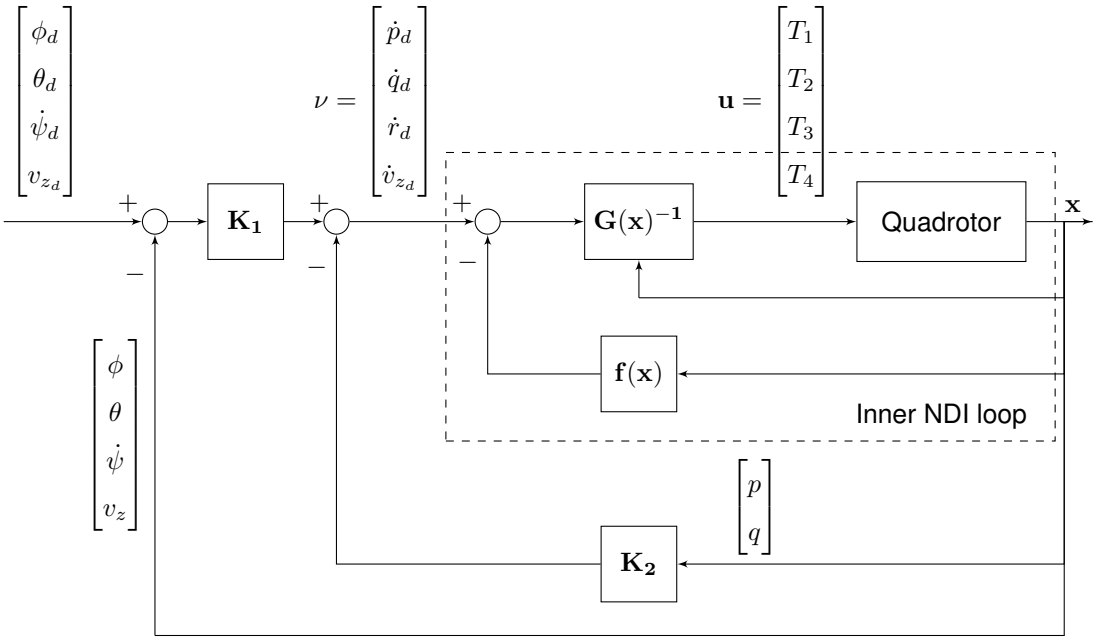


Figure 4.3: NDI controller block diagram.

An interesting detail is related with the inversion of matrix $\mathbf{G}(\mathbf{x})$, given that it is a $4 \times 4$ square matrix including the vertical velocity controller, allowing a standard matrix inversion. In other NDI applications to quadrotor control, [da Silva, 2015] and [Coelho, 2017], only the attitude is included in the NDI controller, thus resulting in a $3 \times 4$ $\mathbf{G}(\mathbf{x})$ matrix that requires a Moore-Penrose pseudo-inverse. This algorithm needs a higher computational power, and therefore it takes more time to be executed, which can represent a major disadvantage when it is used in a real-time application. In this case matrix $\mathbf{G}(\mathbf{x})$ from Equation (4.15) can be approximated as a constant matrix, if small roll ($\phi$) and pitch ($\theta$) angles are considered, allowing one to make an *a prori* offline inversion and store the results for a later use.

The following section describes the implementation of the INDI approach for the same attitude and vertical velocity controller.

## 4.3 Implementation of INDI ontroller

One difference between INDI and NDI controllers in the inner loop consists in a replacement of $\mathbf{f}(\mathbf{x})$ for an estimate $\dot{\mathbf{x}}_\mathbf{0}$ of the NDI state derivative (Equation (4.12)). A brief description of the several processes that have been used in the literature, as well as the adopted estimation method, will be discussed in section 4.5. The input vector with the desired $T_i$ values is now an increment ($\mathbf{\Delta u}$) relative to the previous control action (represented by $\mathbf{u_0}$), therefore, to obtain the complete input ($\mathbf{u}$) one needs to add that previous control values to the increments provided by the INDI controller.

$$\mathbf{u} = \mathbf{u_0} + \mathbf{\Delta u} \tag{4.25}$$

These two modifications completed the adaptation of the NDI controller into its incremental variant. The block diagram for this new controller is depicted in Figure 4.4. Comparing the diagrams of the two NDI controllers it is possible to observe one advantage of the incremental design, namely, it does not require a model for $\mathbf{f}(\mathbf{x})$, even though an estimate for the state vector is now necessary.

## 4.4 Position Controller

The previous sections described the implementation of two Nonlinear Dynamic Inversion variants to solve the problem of attitude and vertical velocity tracking, with both expecting a reference input vector $\mathbf{y_d} = \left(\phi_d,\ \theta_d,\ \dot{\psi}_d,\ v_{z_d}\right)^T$. The position controller receives a desired 3D position expressed in the inertial NED reference frame and transforms it into the desired roll and pitch angles, as well as the vertical velocity represented in the inertial frame, for the NDI/INDI controllers. The yaw rate is provided directly by the pilot, being zero in autonomous flights.

The controller replicates the position control existing in the code of Ardupilot (a detailed explanation of Ardupilot code and structure is provided in chapter 6). The position control is separated in two different loops, one that transforms the horizontal position ($x$ and $y$ axes components) into the desired roll and pitch angles. The other loop transforms the $z$ axis component into a desired vertical velocity.

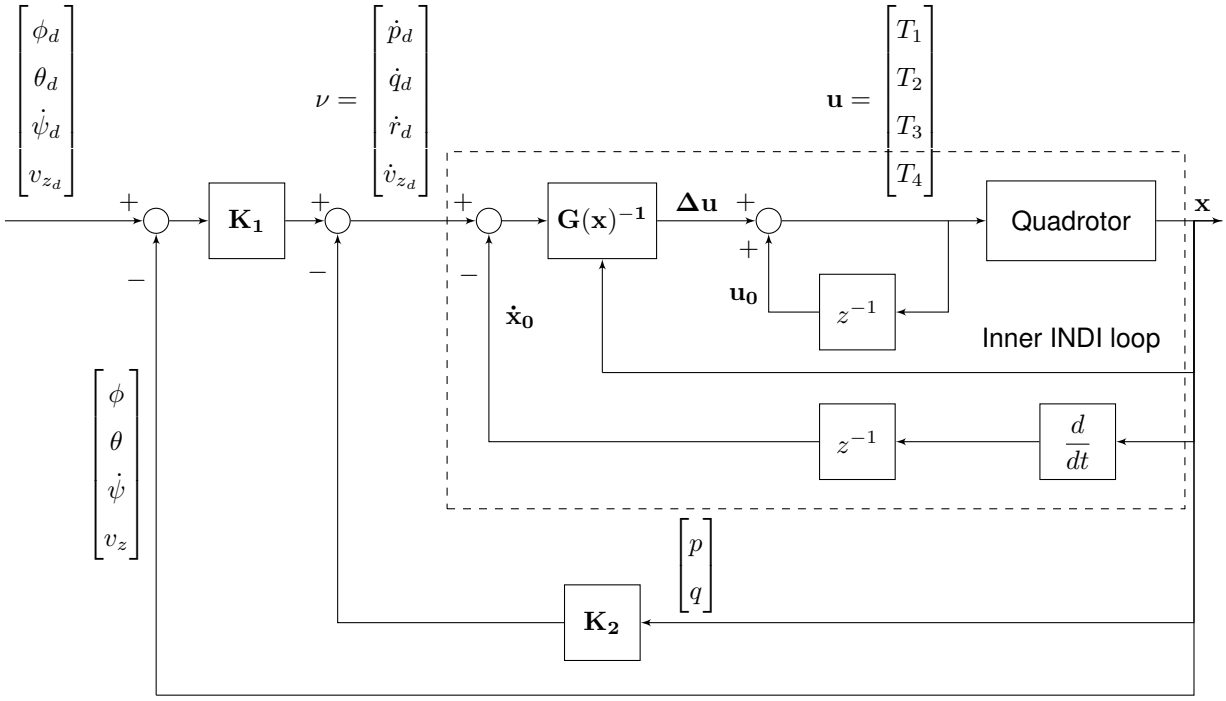



Figure 4.4: INDI controller block diagram.

### 4.4.1 Horizontal position controller

Starting by the horizontal position controller one can divide it in the following stages:

1. The desired position is constrained in a radius of $1\ [m]$ from the current position, a square root controller obtains a desired velocity for the $x$ and $y$ components, which are then constrained for values higher than $12\ [m/s]$.

2. A PI controller with proportional gain $k_{p_v} = 1\ [s^{-1}]$ and integral gain $k_{i_v} = 0.5\ [s^{-2}]$ transforms the velocity references into linear acceleration references.

3. The desired accelerations are constrained to the interval $[-9.81\ m/s^2\ ;\ 9.81\ m/s^2]$, passed by a low-pass filter, converted into a right and forward acceleration components and transformed into the desired roll and pitch angles.

Some details require further explanation, namely, the implementation of the square root controller and the digital first order low-pass filter, the conversion between acceleration components and the transformation into roll and pitch angles.

**Square Root Controller**

The square root controller of the first stage is mathematically represented by the set of Equations (4.26). The $v_{sqrt}$ value is computed using Equation (4.27); $d$ is the horizontal distance between the current and desired positions; $x_d$, $x$, $y_d$ and $y$ are the desired and current position components; $k_p$ represents an

adjustable proportional gain and $a$ is a limiting acceleration value.

$$\begin{cases} v_{x_d} & = v_{sqrt} \times \dfrac{x_d - x}{d} \\ v_{y_d} & = v_{sqrt} \times \dfrac{y_d - y}{d} \end{cases} \tag{4.26}$$

$$v_{sqrt} = \sqrt{2a \times \left( d - \frac{a}{2k_p} \right)} \tag{4.27}$$

**Low-pass Filter**

After the saturation to interval $[-9.81\ m/s^2\ ;\ 9.81\ m/s^2]$ the accelerations pass through a low-pass filter, in order to attenuate the high frequency noise from the measurements of position and velocity. The cutoff frequency ($f_c$) presents a value of $5\ [Hz]$. The digital implementation of this filter can be described by Equation (4.28), where $\Delta t$ represents the sampling time and $in$ and $out$ are the input and output values of the filter, respectively.

$$out = in \times \frac{\Delta t}{\Delta t + \dfrac{1}{2\pi f_c}} \tag{4.28}$$

**Conversion of acceleration to right forward components**

To convert the accelerations one must only rotate the components using the current yaw angle, as described by the following expressions, with $a_x$ and $a_y$ representing the acceleration components expressed in the inertial frame; $a_{forward}$ and $a_{right}$ are the acceleration forward and right components.

$$\begin{cases} a_{forward} & = a_x \cos\psi + a_y \sin\psi \\ a_{right} & = -a_x \sin\psi + a_y \cos\psi \end{cases} \tag{4.29}$$

**Transformation into desired roll and pitch angles**

The transformation from forward and right linear acceleration components to roll and pitch angles follows Equations (4.30). The factor $\frac{180}{\pi}$ converts the angles from radians to degrees and $atan$ represents the arctangent mathematical function. Both desired angle values are saturated within the interval $[-30^{\text{o}}\ ;\ 30^{\text{o}}]$ before being sent to the attitude NDI/INDI controllers.

$$\begin{cases} \theta_d & = atan\left( \dfrac{-a_{forward}}{g} \right) \times \dfrac{180}{\pi} \\ \phi_d & = atan\left( \dfrac{a_{right} \times \cos\theta_d}{g} \right) \times \dfrac{180}{\pi} \end{cases} \tag{4.30}$$

**Block Diagram**

After the explanation present in each stage of the horizontal position control implemented in Ardupilot, a block diagram overview can be depicted in the following Figures, namely, the first stage is presented in Figure 4.5, second stage in Figure 4.6 and the final third stage in Figure 4.7.

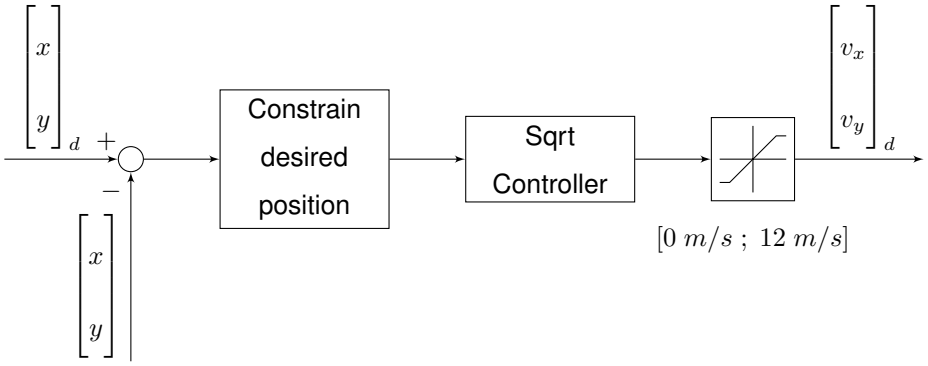



Figure 4.5: First stage of horizontal position controller block diagram.

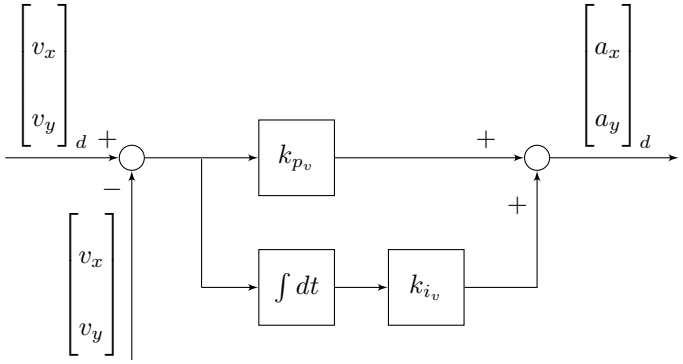



Figure 4.6: Second stage of horizontal position controller block diagram.

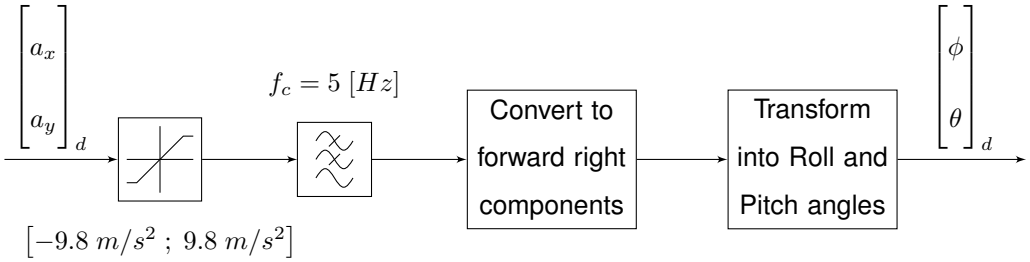



Figure 4.7: Third stage of horizontal position controller block diagram.

### 4.4.2 Vertical position controller

The vertical position controller is simpler than the horizontal position controller, since it only needs to transform a desired position into a desired velocity. A square root controller, similar to the one presented in Equations (4.26) but applied to the $z$ axis, is used for that purpose. Before and after the square root controller there are saturations in position and velocity, respectively. The saturation in position restrains the position error to the interval $[-1\ m\ ;\ 1\ m]$ and the velocity saturation restrains the desired velocity to the interval $[-2.5\ m/s\ ;\ 1.5\ m/s]$, where it is worth mentioning that the higher absolute value ($-2.5\ [m]$) corresponds to an up velocity, since the $z$ axis is pointing down. A block diagram of the vertical position controller is depicted in Figure 4.8.

The block diagram of the vertical position controller completes the description of the overall position controller, whose block diagram can be seen in Figure 4.9. The final section of this chapter will analyze

some details that are raised during an implementation, both in simulation and experimental scenarios.

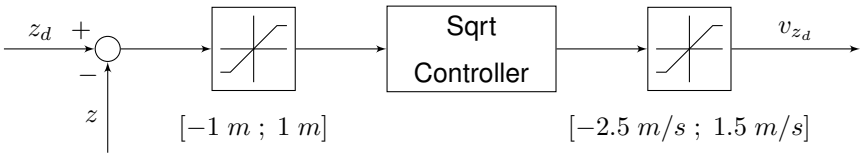


Figure 4.8: Vertical position controller block diagram.

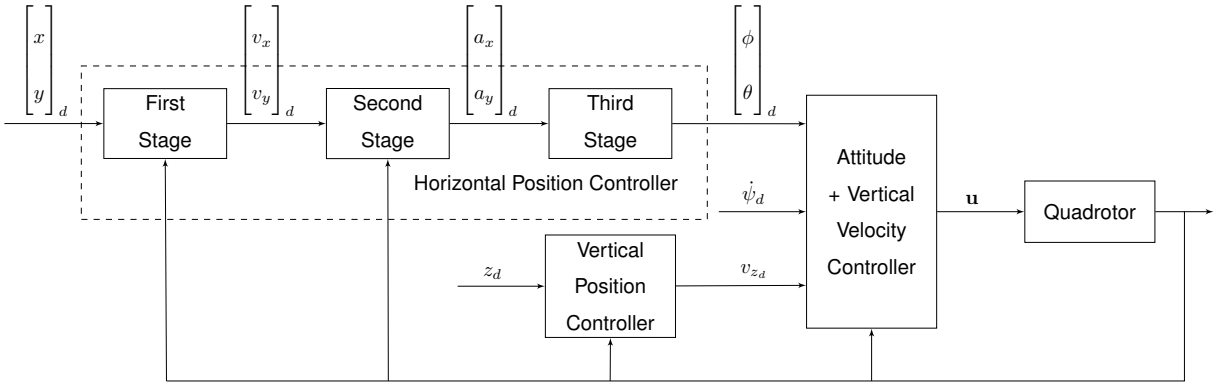


Figure 4.9: Position controller block diagram.

## 4.5 Implementation Details

As it was seen before, the major advantage of incremental NDI over its nonincremental variant relies in the fact that it does not need part of the model of the system, namely, the $\mathbf{f}(\mathbf{x})$ function. However, the output of this function is replaced by an estimate of the state vector ($\dot{\mathbf{x}}_\mathbf{0}$). A seen in chapter 2 several estimation processes have been used in the literature. For this thesis the angular and linear accelerations of the state vector are estimated using a differentiation followed by a low-pass filtering, as used in [Smeur et al., 2015] and in [da Silva, 2015].

This differentiation plus filtering approach was selected due to its low complexity and independence from a system model, allied with satisfactory results. A compromise must be found for the order of the filter, since high frequency noise attenuation increases with filter order, but the delay of the estimated accelerations is also increased. The low-pass filtering must be applied to all the necessary variables for the INDI controllers and not only to the state vector $\mathbf{x_0}$, in order to achieve time synchronization between them and therefore avoid possible syncing problems. Further development on this topic can be seen in chapter 5, where the model order will be chosen using MATLAB/Simulink simulations.

Another possible solution to reduce disturbances in INDI controller is the introduction of an adjustable input scaling gain $\eta$ that varies between $0$ and $1$, in the incremental control action $\mathbf{\Delta u}$. This parameter scales the new incremental control action in the overall input $\mathbf{u}$, with $\eta = 1$ corresponding to a complete action and $\eta = 0$ no action at all, as shown in Equation (4.31a). Replacing the expression for $\mathbf{\Delta u}$ one

can observe that the new parameter multiplies the control effectiveness matrix $\mathbf{G}(\mathbf{x})$.

$$\mathbf{u} = \mathbf{u_0} + \eta \mathbf{\Delta u} \tag{4.31a}$$

$$\mathbf{u} = \mathbf{u_0} + \eta \mathbf{G}(\mathbf{x})^{-\mathbf{1}} \left[\nu - \dot{\mathbf{x}}_\mathbf{0}\right] \tag{4.31b}$$

A more refined adjustment can be performed if a separate parameter is introduced for each variable of the state vector, being $\eta$ a diagonal matrix.

A low value for these parameters reduce the effect of disturbances because they minimize control action to sudden changes in the acceleration estimates. At the same time control action saturations are also minimized. However, the tracking performance degrades for lower input scaling gain values, given that a perfect dynamic inversion is no longer achieved. The value $\eta = 0.4$ will be used as an initial estimate and a parametric analysis to select the better value will be performed in chapter 5.

Another detail that must be analyzed is the sampling time of digital implementations, where a compromise must also be achieved. On one side INDI derivation required a small sampling time, in order to apply some simplifications, but on the other side a smaller sampling time amplifies the noise obtained from the differentiation process, leading to a degradation of performance. At the lack of a better reference value, the Ardupilot controller algorithm sampling time will be used as a starting value. The code is run at $400\ [Hz]$, which corresponds to a sampling time $t_s = 0.0025\ [s]$. A validation of this value, or a new choice for it, will be also tested with MATLAB/Simulink in the following chapter.

# Chapter 5

# Implementation and Results in MATLAB/Simulink

This chapter presents the implementation of the controllers from the previous chapter in a MATLAB/Simulink environment, including the adaptations and extra details that were added to obtain simulations with a better approximation to real flight scenarios. It also includes simulation results validating the performance of NDI and INDI control designs.

The first section describes the additional details added to the quadrotor model from the previous chapter. Section 5.2 presents the modelling of the various sensors included in the Spyro quadrotor. In section 5.3 the implementation and validation of the state vector derivative ($\dot{\mathbf{x}}_\mathbf{0}$) for INDI controller is shown. The simulation results and validation of the attitude and vertical velocity NDI and INDI controllers can be seen in sections 5.4 and 5.5, respectively. The selection of input scaling gain ($\eta$) and sampling time ($T_s$) using parametric analysis is presented in section 5.6. Robustness tests to model uncertainties, wind velocity and loss of performance in an actuator are presented in section 5.7. Finally, the validation of the INDI controller combined with the position controller from Ardupilot is depicted in section 5.8.

## 5.1 Quadrotor Model

The model of the Quadrotor that was used for all the simulations performed in MATLAB/Simulink simulation environment was developed by the supervisor of this thesis as an interpreted MATLAB function, that follows the equations of motion described by Equations (4.8) and (4.9). It also includes the following details, in order to represent a more complex model that provides a better approximation to the real Spyro quadrotor:

- Battery discharge model
- Aerodynamic drag
- Wind velocity input
- Motor and propeller models

- Magnetic field information

The battery model is an important addition, given that all the energy that is consumed in a quadrotor is provided by the battery and consequently its capacity and efficiency represent relevant limitations in terms of flight time. Having a model for the battery discharge allows one to compare the efficiency of the applied controllers in terms of used energy, which can be used as a criterion to select the better controller.

Aerodynamic drag and wind velocity are the most important dynamic effects that were neglected in the modelling of the quadrotor utilized to build the NDI/INDI controllers, in the previous chapter. The first is a force that is always present during any quadrotor flight, even though it is considerably smaller at low speed flights. The latter can be considered the major disturbance for the greater part of the flights. The wind velocity ($V_w$) is added to the quadrotor velocity relative to the ground ($V$), in order to give the quadrotor velocity relative to the air ($V_a$). The aerodynamic drag is described by Equation (5.1), where $C_{D_0}$ is a nonisotropic drag coefficient provided by UAVision; $\|V_a\|$ is the 3D euclidean norm of the air velocity vector.

$$D = -C_{D_0}\|Va\|Va \tag{5.1}$$

The models for the motor and propellers simulate the loss of efficiency with battery discharge in terms of providing Thrust. In other words, with a lower battery energy level it is necessary to provide more power to the motors, in order to obtain the same Thrust force from them. The models also provide information for the speed of each propeller and the current of each motor as a function of time.

This model also has information about the magnetic field, so that a magnetometer can be modeled. This sensor modelling, among several others will be described in section 5.2.

### 5.1.1 Numerical derivation of control effectiveness matrix $\mathbf{G}(\mathbf{x})$

As it was seen in the previous chapter, $\mathbf{G}(\mathbf{x})$ can be approximated as a constant matrix that does not require a real-time estimation method. This matrix can be obtained using a linearization of the system, given that a linear system can be written as Equation (5.2), and comparing with Equation (3.29a) one can conclude that $\mathbf{B}(\mathbf{x}) \equiv \mathbf{G}(\mathbf{x})$.

$$\dot{\mathbf{x}} = \mathbf{A}(\mathbf{x})\mathbf{x} + \mathbf{B}(\mathbf{x})\mathbf{u} \tag{5.2}$$

Using MATLAB/Simulink the system was numerically linearized, obtaining the values of Equation (5.3) for $\mathbf{G}(\mathbf{x})$ matrix. For $\mathbf{f}(\mathbf{x})$ one has to compute its value in real-time, given that it is dependent from the body angular velocity components.

$$\mathbf{G}(\mathbf{x}) = \begin{bmatrix} -87.4992 & 87.4992 & 87.4992 & -87.4992 \\ 53.7083 & -53.7083 & 53.7083 & -53.7083 \\ 6.5935 & 6.5935 & -6.5935 & -6.5935 \\ -12.0382 & -12.0382 & -12.0382 & -12.0382 \end{bmatrix} \tag{5.3}$$

## 5.2 Sensors Modelling

The controllers designed in the previous chapter require the feedback of the following list of variables:

- Position vector expressed in inertial frame: $\mathbf{P} = (x,\ y,\ z)^T$
- Velocity vector expressed in inertial frame: $\mathbf{V} = (v_x,\ v_y,\ v_z)^T$
- Acceleration vector expressed in inertial frame: $\mathbf{A} = (a_x,\ a_y,\ a_z)^T$
- Euler angles: $\mathbf{\Theta} = (\phi,\ \theta,\ \psi)^T$
- Angular velocity vector expressed in body frame: $\omega_{\mathbf{b}} = (p,\ q,\ r)^T$

These variables were estimated used the following set of sensors, which largely corresponds to the usual sensors available in a quadrotor:

- Gyroscope, to measure the angular velocity vector $\omega_b$
- Accelerometer, to measure the body frame acceleration vector
- Magnetometer, measuring the magnetic field
- Barometer, to measure the altitude $h$
- GPS sensor, measuring both the velocity and position in the inertial frame
- Attitude sensor, to measure the Euler angles

All sensors models are performed by introducing Band-Limited White Noise around the ideal values given by the quadrotor model, also including a sampling time that models the sampling frequency limitation for each specific sensor. A zero-order order hold is also introduced, in order to model the natural delay existing in a sensor measurement. The gyroscope, accelerometer, magnetometer and barometer raw sensor values provided by UAVision can be depicted in Table 5.1. The Table includes values for noise standard deviations and sampling times. It is worth mentioning that the accelerometer values are written in $[g]$ units, where $1g \approx 9.80665\ [m/s^2]$. The magnetometer standard deviation is given as a value relative to a 3D location around Lisbon, thus it does not have units.

The objective of the simulations is to replicate the values provided by the sensors and estimators in Ardupilot. This platform uses an Extended Kalman Filter (EKF) to reduce the noise of sensor measurements, at the same time that it uses sensor fusion to obtain values for variables that are not measured

Table 5.1: Raw sensor noise modulation values.

| Sensor | Noise Standard Deviation | Sampling Time |
|---|---|---|
| Gyroscope | $0.076\ [rad/s]$ | 0.02 $[s]$ |
| Accelerometer | $0.017\ [g]$ | 0.02 $[s]$ |
| Magnetometer | $0.07\ [-]$ | 0.02 $[s]$ |
| Barometer | $0.36\ [mbar]$ | 0.10 $[s]$ |

by the existing sensors. Therefore, by using raw sensor information one is largely overestimating the noise present in the values provided to the controllers in Ardupilot. Common noise power values after using the EKF are not available for Ardupilot, but exist for Pixhawk, which is an autopilot platform based on Ardupilot that possesses the same EKF estimation system. The values were taken from [PX4, 2017] and can be depicted in Table 5.2. One can observe that the EKF estimation process reduces the noise standard deviations in, at least, one order of magnitude, thus it can be considered an effective method to obtain more accurate estimates.

Table 5.2: Pixhawk sensor noise modelling values.

| Sensor | Noise Standard Deviation | Sampling Time |
|---|---|---|
| Gyroscope | $8.727 \times 10^{-4}\ [rad/s]$ | 0.02 $[s]$ |
| Accelerometer | $0.0011\ [g]$ | 0.02 $[s]$ |
| Magnetometer | $0.005\ [-]$ | 0.02 $[s]$ |
| Barometer | $0.012\ [mbar]$ | 0.10 $[s]$ |

For the GPS and Attitude sensors a model from Xsens company was used, namely the MTI 100-series. The noise standard deviations were taken from the datasheet in Appendix D.1. These values are shown in Table 5.3.

Table 5.3: Xsens sensor noise modelling values.

| Sensor | Noise Standard Deviation | Sampling Time |
|---|---|---|
| GPS velocity | $0.1\ [m/s]$ | 0.25 $[s]$ |
| GPS horizontal position | $1\ [m]$ | 0.25 $[s]$ |
| Attitude (Roll and Pitch) | $0.0052\ [rad/s]$ | 0.01 $[s]$ |
| Attitude (Yaw) | $0.0175\ [rad/s]$ | 0.01 $[s]$ |

The GPS vertical position estimation is not used since it gives a worse estimate than the barometric altitude. The GPS measures have an insufficient sampling frequency and are considerably noisy, so they are commonly introduced in the EKF estimation. Position and velocity estimates are then available at higher sampling rates. To simulate that effect the sampling rates were increased to $50\ [Hz]$ and $25\ [Hz]$, or sampling times of $T_s = 0.02\ [s]$ and of $T_s = 0.04\ [s]$, for GPS velocity and position, respectively. The frequency in the position estimates is lower due to the slower dynamics of this control loop.

The majority of quadrotors do not have an attitude sensor, estimating the Euler angles with sensor fusion between the gyroscope, accelerometer and magnetometer measures. In this case Ardupilot

estimates the attitude in quaternions and then transforms them to Euler angles. This approach was not followed due to its complexity and given that Xsens sensor model to measure Euler angles produce acceptable results for simulation purposes.

## 5.3 Estimation of INDI State Vector Derivative

It was stated in chapter 4 that the state derivative for the INDI controller will be estimated using a differentiation followed by a low-pass filtering. This technique can be implemented using a first order high-pass filter, described by Equation (5.4) in the Laplace domain. The differentiation is given by the $s$ variable in the numerator, with the low-pass filtering being given by the denominator expression. It is worth mentioning that the static gain of this filter is not equal to $1$.

The implementation can also be made with a second order bandpass filter (Equation (5.5)).

$$\text{First order high-pass filter} = \frac{s}{1+Ts} \tag{5.4}$$

$$\text{Second order bandpass filter} = \frac{\omega_n^2 s}{s^2 + 2\xi\omega_n s + \omega_n^2} \tag{5.5}$$

With both filters calibrated to obtain good performance in the estimation, it is possible to compare them and conclude that, on one side, the second order filter adds a higher time delay, which is a disadvantage if the system has fast dynamics that need to be controlled. On the other side, it has the advantage of reducing the high frequency noise, as one can see in Figure 5.1, where the bode magnitude plots of the first order filter is shown for unit time constant $T$. The second order natural frequency $\omega_n$ was selected as the equivalent one that results in the same $T = 1\ [s]$, being $\omega_n = 2\pi\ [rad/s]$. For the damping ratio the value $\xi = 0.8$ was used. The first order filter has a constant magnitude in high frequencies, thus it does not reduce the high frequency noise as the second order filter.

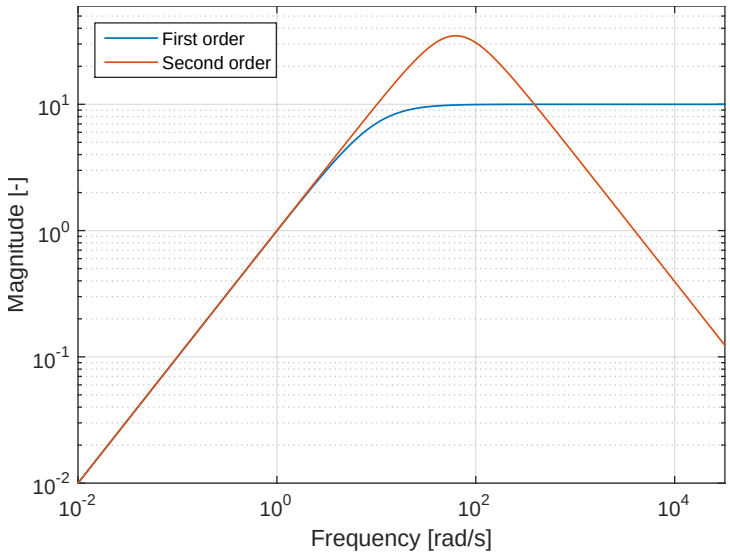


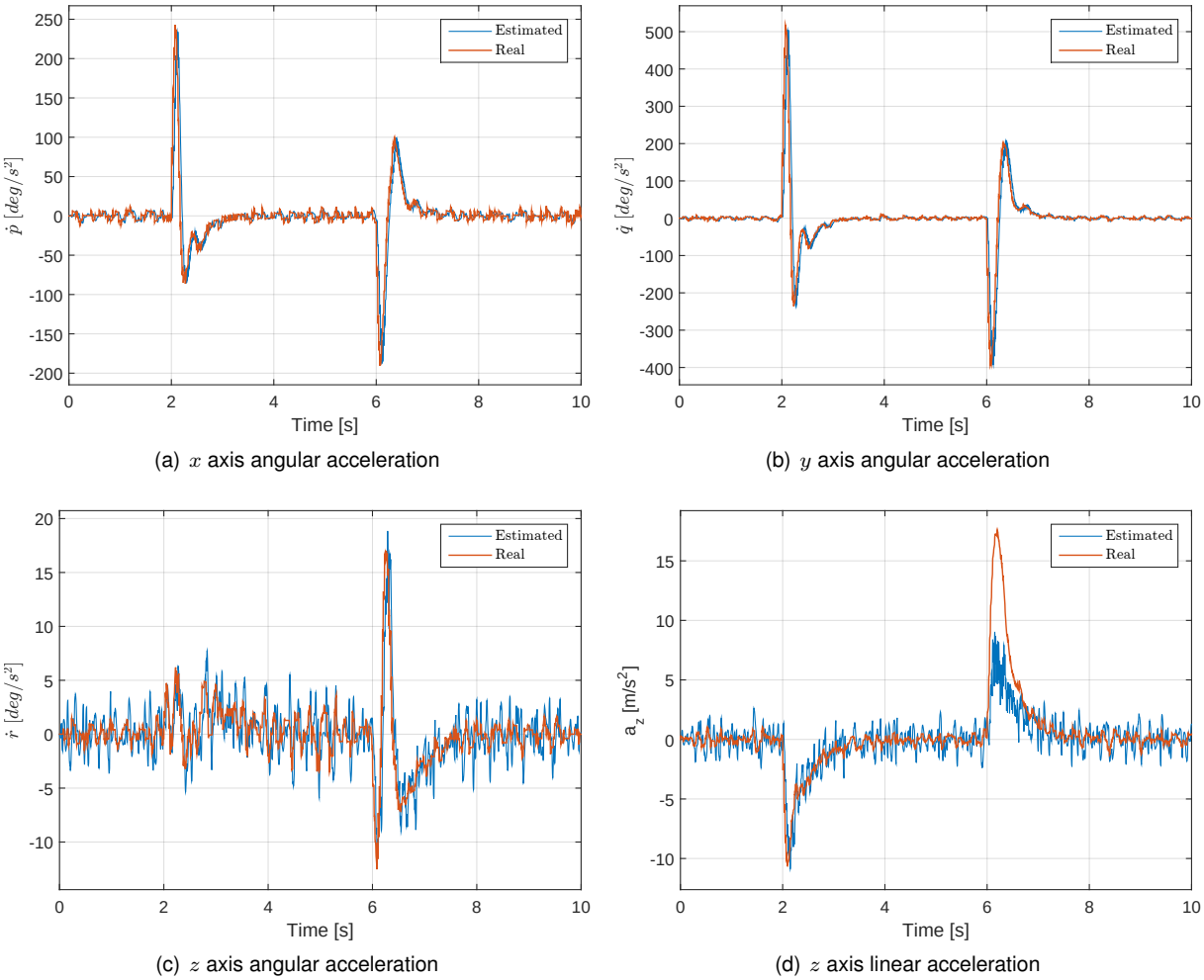
Figure 5.1: Bode Magnitude plot of state derivative estimator filters.

The best results for the tracking problem using the INDI controller were obtained using a second

order filter with a natural frequency $\omega_n = 200\pi\ [rad/s]$ and a damping ratio $\xi = 0.8$, thus these values will be used in the following simulations.

As explained in chapter 4, a Second-Order low-pass filter (Equation (5.6)) with the same natural frequency and damping ratio will be used for the remaining variables that are necessary for the INDI controller, to achieve synchronization between all the variables. Desynchronized variables can interfere with the dynamic of the closed loop system, and consequently provoke unintended disturbances. For instance, if the low-pass filter is not introduced, a virtual input $\nu$ is given to the system, which results in an incremental input without the corresponding derivative estimate, that is not yet calculated. When the estimate is computed a new virtual input was already given to the system, resulting in a different incremental input. An estimate of the inner state vector (Equation (4.12)) from a simulation with INDI controller can be seen in Figure 5.2.

$$\text{Second order low-pass filter} = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \tag{5.6}$$




(a) $x$ axis angular acceleration

(b) $y$ axis angular acceleration

(c) $z$ axis angular acceleration

(d) $z$ axis linear acceleration

Figure 5.2: INDI inner state estimation with second order bandpass filter.

Analyzing the results, it is possible to say that $x$ and $y$ axis angular acceleration estimates present less noise than the $z$ axis angular acceleration. This result is a consequence of the higher noise in yaw sensor measurement than the roll and pitch measurements (sensor modelling explained in section 5.2).

## 5.4 NDI Controller Simulation Results

### 5.4.1 NDI simulation results

Two types of simulations will be presented for each controller, one with sequential reference inputs and another with simultaneous inputs. The reference inputs for both simulation types are described in the following Table, where the vertical velocity reference is negative in order to be an upward velocity, given that the $z$ axis of the inertial frame is pointing down.

Table 5.4: Reference inputs for simulations.

| | Variable | Value | Time [$s$] |
|---|---|---|---|
| Type 1 | $\phi_d$ | 10 [$^\circ$] | 1-6 |
| | $\theta_d$ | 20 [$^\circ$] | 8-13 |
| 30 [$s$] sim | $\dot{\psi}_d$ | 3 [$^\circ/s$] | 15-20 |
| | $v_{z_d}$ | -4 [$m/s$] | 22-27 |
| Type 2 | $\phi_d$ | 10 [$^\circ$] | 2-6 |
| | $\theta_d$ | 20 [$^\circ$] | 2-6 |
| 10 [$s$] sim | $\dot{\psi}_d$ | 3 [$^\circ/s$] | 2-6 |
| | $v_{z_d}$ | -4 [$m/s$] | 2-6 |

When the type 1 simulation was run with the NDI controller (MATLAB/Simulink block diagram presented in Appendix B) the vertical velocity control channel does not reach the reference of $-4$ $[m/s]$, staying at $-2$ $[m/s]$, as one can see in Figure 5.3. Furthermore, there are also small couplings between the altitude channel and the remaining channels, resulting in small oscillations of the $v_z$ curve when its reference is zero. Therefore, it is possible to conclude that a successful altitude tracking was not reached. For the remaining channels reference tracking is successfully achieved with good performance.

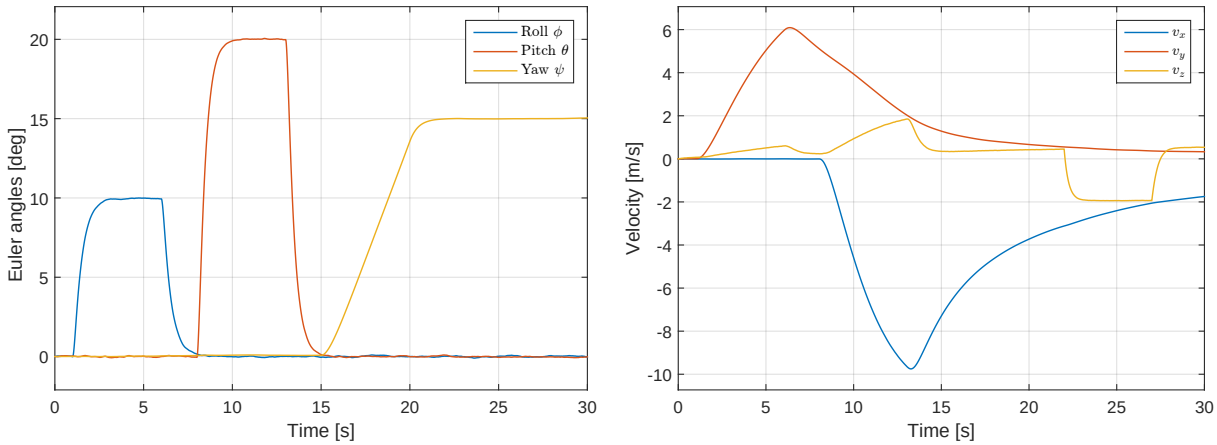
The static error can be canceled if the first order proportional control law, used for the vertical velocity in the outer loop of the NDI controller, is replaced by a proportional integrative control law. In other words the static error can be canceled if an integrative term is introduced in the control law of the vertical velocity. The new controller presents the form of Equation (A.7). Once again, the selection of the integrative gain was performed using a parametric simulation, shown in Appendix A.3. The chosen value was $K_i = 0.2$ $[s^-2]$.

The type 1 simulation was again performed, this time with the new PI controller, producing the results of Figure 5.4.

Analyzing the simulation with the new PI control law one can conclude the following points:

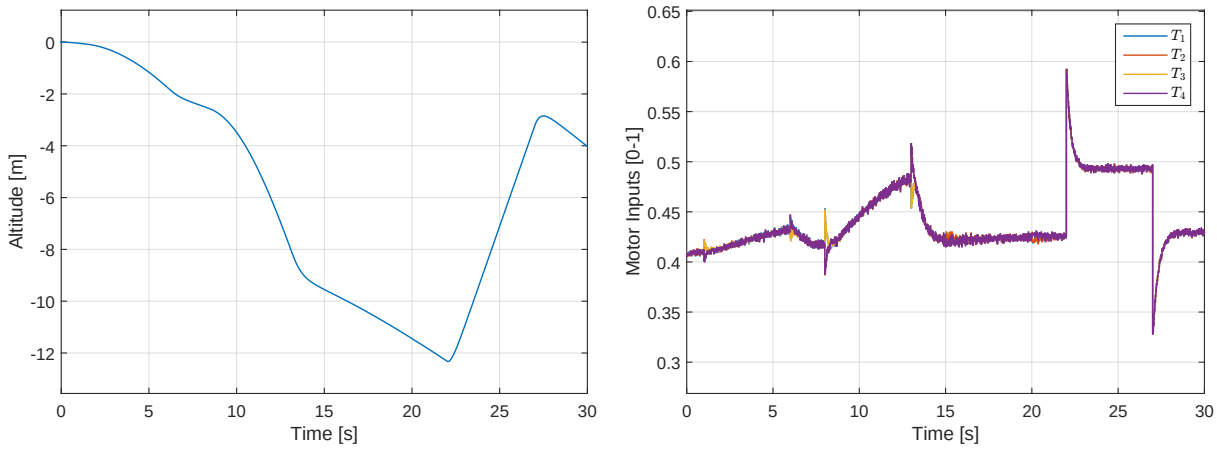
1. Vertical velocity control channel still presents some minor couplings, resulting in the oscillations of the $v_z$ curve.
2. The static error in altitude was eliminated, since the reference $v_z$ is followed.

To verify this control design a type 2 simulation was also performed, originating the results of Figure




(a) Euler Angles

(b) Inertial frame velocity




(c) Altitude

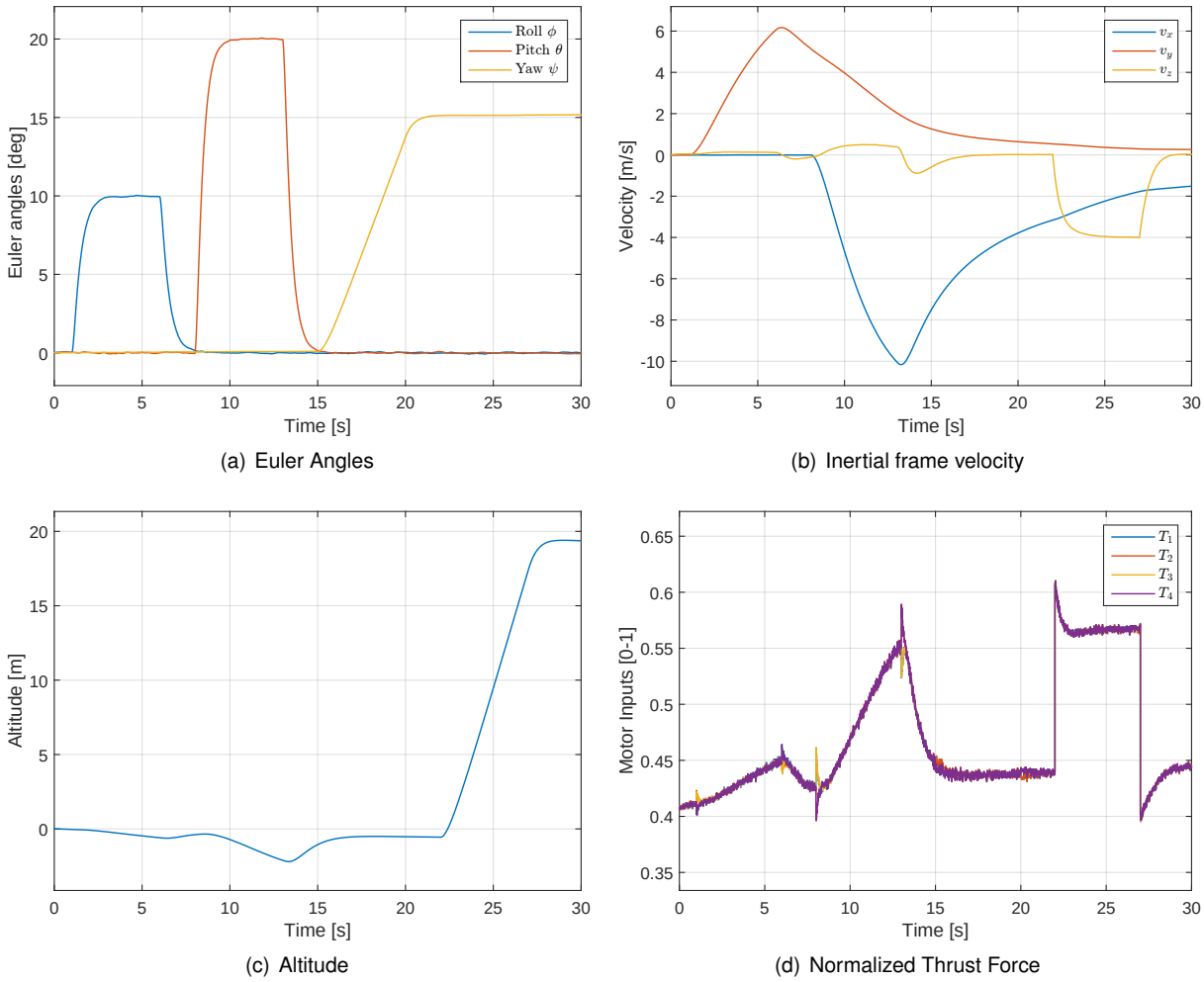
(d) Normalized Thrust Force

Figure 5.3: Type 1 Simulation for NDI controller with P control law in vertical velocity.

5.5. This type 2 simulation is a more demanding simulation, since it asks for simultaneous inputs, which will result in a larger control action. Furthermore, the existing couplings should be more evident.

With simultaneous references the NDI controller still presents good performance for the attitude channels, given that all the desired references are achieved. The roll and pitch angle tracking is directly seen from the Euler angles graph, but the yaw rate tracking can also be seen in the same graph, even though it is in an indirect manner. A final yaw angle around $12.5\ [^\circ]$ is reached in a four second reference duration, thus, the yaw rate is approximately the $3\ [^\circ/s]$. The vertical velocity channel this time does not reach the asked $-4\ [m/s]$ and when the reference returns to zero the vertical velocity still takes almost four seconds to achieve the same value. The previous behaviour occurs due to the integrative term because the integral must also converge, which happens in this case when the altitude reaches $\int_2^6 4\ [m/s]dt = 16\ [m]$. The reference tracking for this channel presents some degradation due to an imperfect dynamic inversion of the quadrotor model, which is a consequence of the disturbances and the noise modeled in the sensors, as well as the simplifications made to the model.

The results of the simulations with the NDI controller show that the application of this control design for the proposed reference tracking problem was almost globally verified, existing only minor tracking issues with the altitude channel. In order to validate these results a comparison with the original attitude




(a) Euler Angles

(b) Inertial frame velocity

(c) Altitude

(d) Normalized Thrust Force

Figure 5.4: Type 1 Simulation for NDI controller with PI control law in vertical velocity.

and vertical velocity controller, used in Ardupilot, must be performed. The following two sections present the design and simulation results for the original controller.

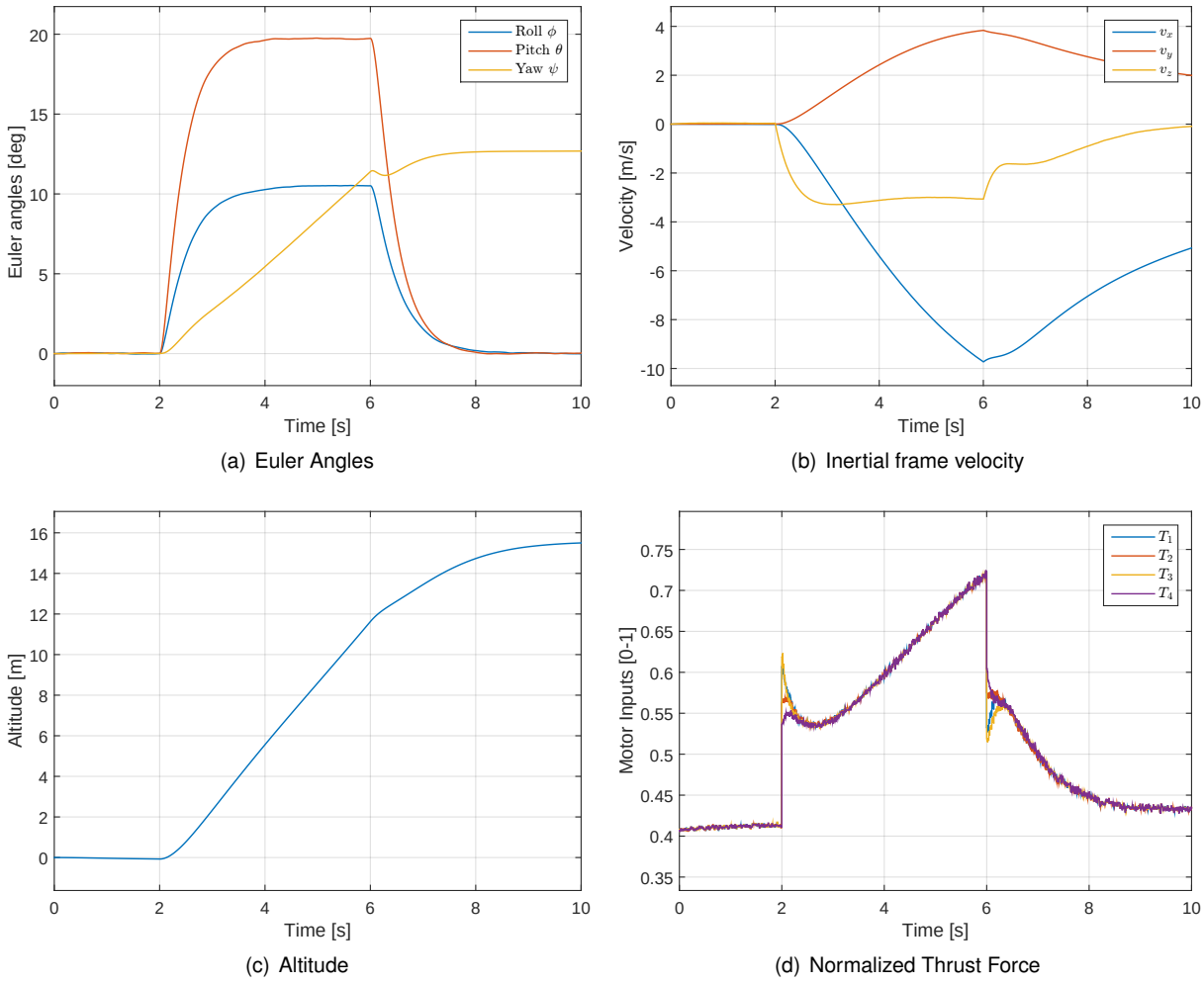
### 5.4.2 Ardupilot attitude + vertical velocity control design

Similarly to the position control design of section 4.4, the controller can be divided in two parallel parts, namely, an attitude part that receives the Euler angles and rate inputs, as well as a vertical control part that receives the vertical velocity input.

#### Attitude control design

The attitude control design is composed by a first stage that transforms the desired roll and pitch angles, as well as the desired yaw rate, into reference body rates ($p, q$ and $r$). It also has a second stage that transforms the body rates into control inputs in range $[0 - 1]$ for each channel. In order to obtain the inputs it is only necessary to multiply the control inputs by the actuation matrix of the quadrotor.

The first stage is composed by an acceleration limiter and a square root controller with a feedforward term and the second stage includes a PID controller. The block diagram of the attitude control design is presented in Figure 5.6.




(a) Euler Angles (b) Inertial frame velocity

(c) Altitude (d) Normalized Thrust Force

Figure 5.5: Type 2 Simulation for NDI controller with PI control law in vertical velocity.

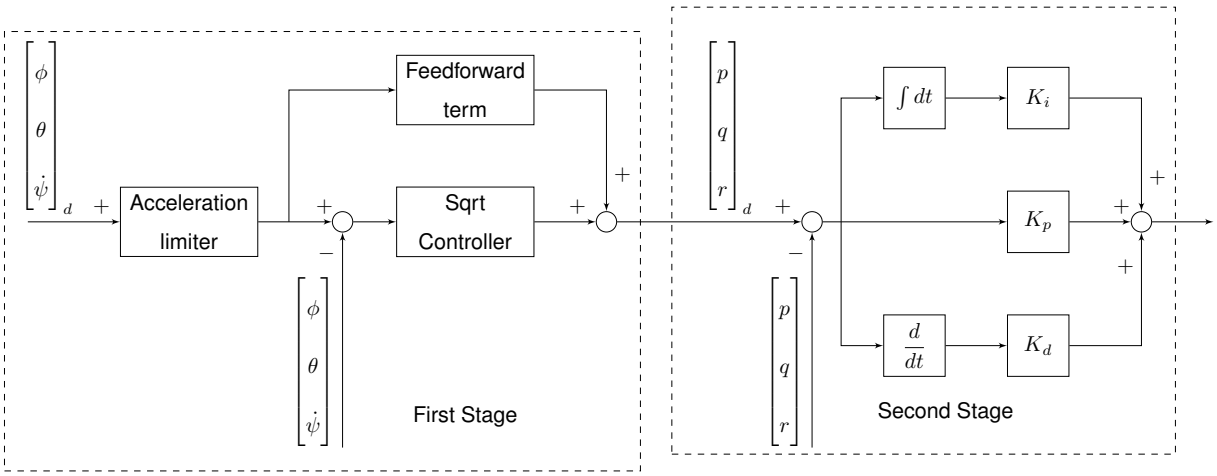



Figure 5.6: Block diagram of original Ardupilot attitude controller.

### Vertical control design

The vertical control design receives a vertical velocity reference and can be decomposed in the following four stages: the velocity reference passes through a limiter and an integrator, obtaining a reference altitude; the desired altitude error is saturated and then transformed into a desired vertical velocity, using a square root controller with feedforward component. The velocity reference is again

saturated; the vertical velocity error passes through a low-pass filter and a proportional controller with feedforward component, resulting in a acceleration reference; the acceleration error also passes through a low-pass filter, before a PID controller is applied, which provides a desired throttle value relative to the hover condition. The first and second stages of the controller can be seen in Figure 5.7. The third and fourth stages are depicted in Figure 5.8.

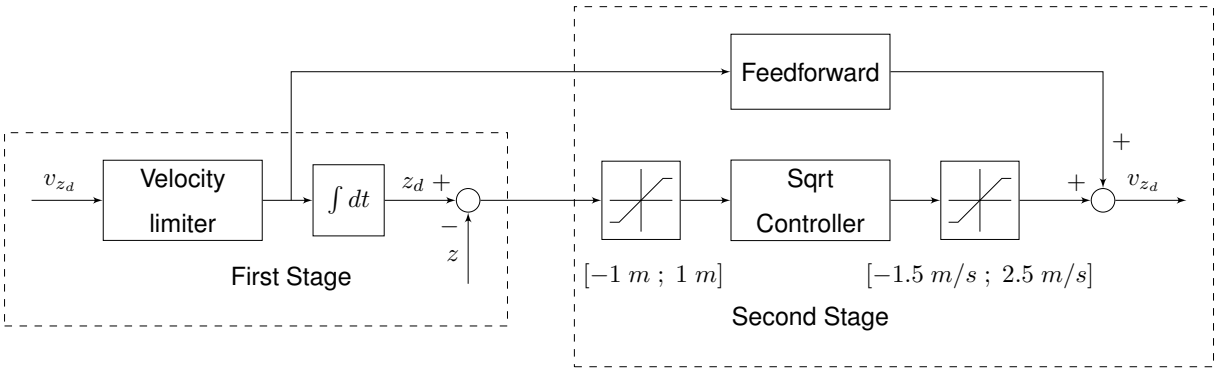



Figure 5.7: Block diagram for the first two stages of original Ardupilot vertical velocity controller.

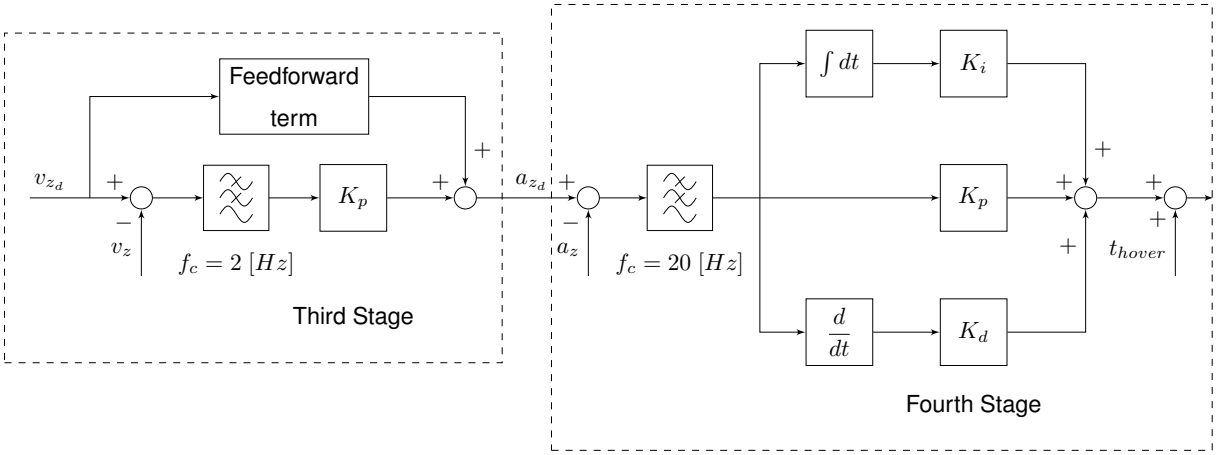



Figure 5.8: Block diagram for the last two stages of original Ardupilot vertical velocity controller.

The complete original control design, including the attitude and vertical velocity controllers, as well as the actuation matrix that transforms the inputs of each channel into inputs are shown in Figure 5.9. The results of this controller from both simulation types are presented in the following section.

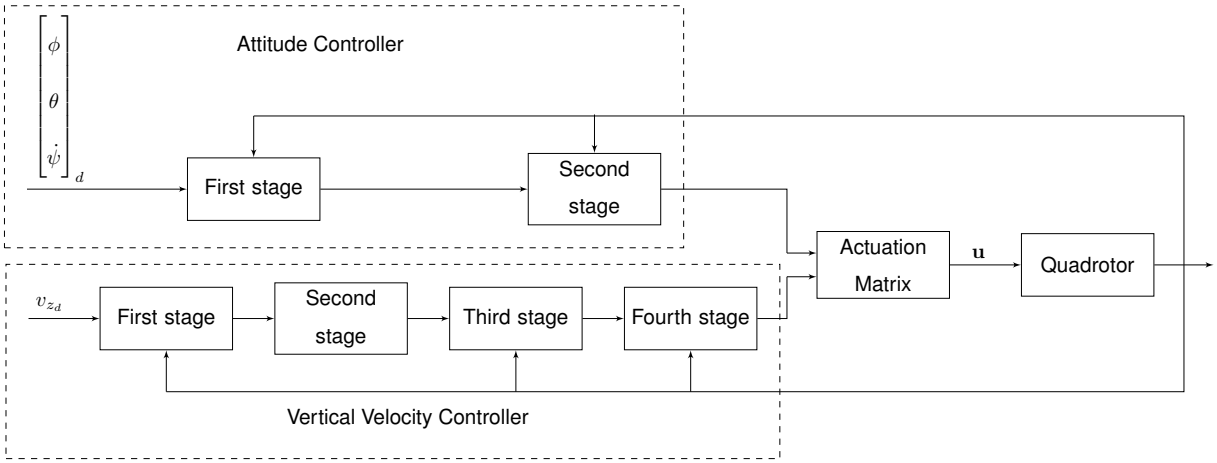
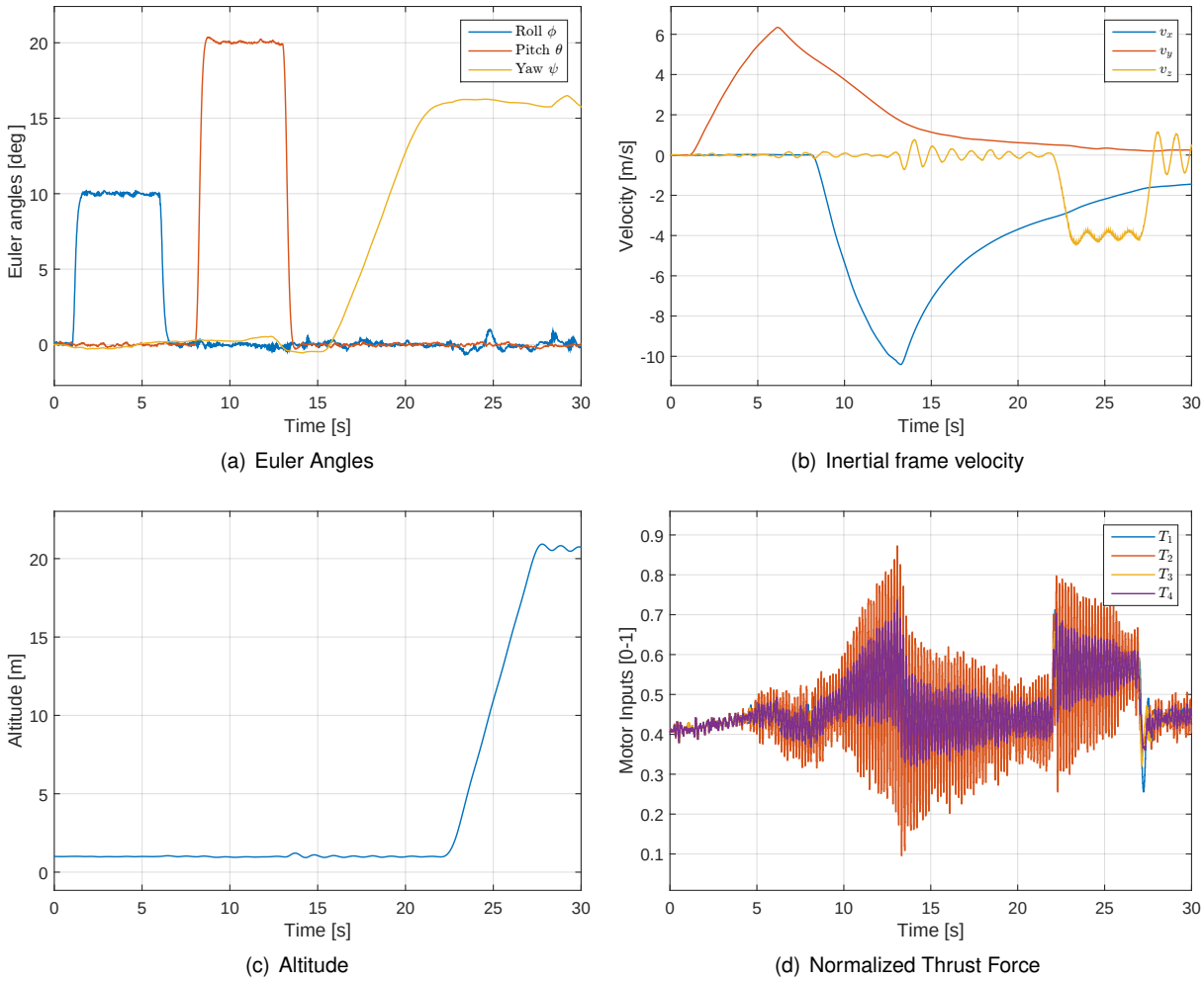



Figure 5.9: Complete block diagram of original Ardupilot attitude and vertical velocity controller.

### 5.4.3 Ardupilot control design simulation results

The tuning process of the several gains and parameters from the Ardupilot controller was a rather difficult and time consuming task, due to the influences that exist between the several parameters. In other words when an adjustment was done in a parameter several other parameters also needed to be changed, in order to achieve good results. The NDI/INDI controllers are advantageous in this aspect, given that they present a considerably smaller number of adjustable parameters, only 5 instead of the 42 from the original controller, and consequently one needs less time to the design these controllers. The results of both simulations (type 1 and type 2 simulations) for the original Ardupilot controller are presented in Figures 5.10 and 5.11.




(a) Euler Angles

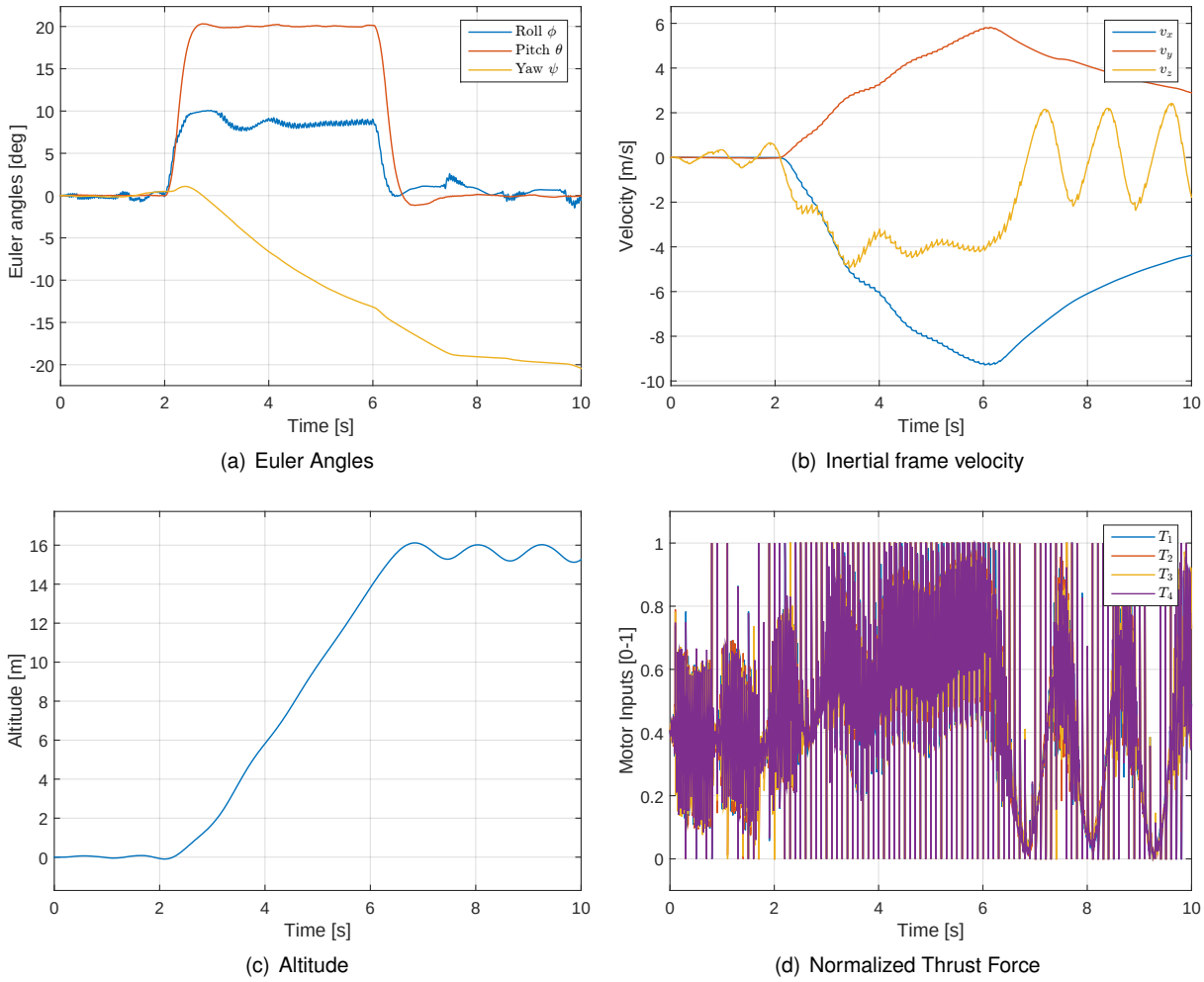
(b) Inertial frame velocity

(c) Altitude

(d) Normalized Thrust Force

Figure 5.10: Type 1 simulation results for Ardupilot original controller.

Analyzing the type 1 simulation results it is possible to say that tracking is successfully achieved. All the references are reached, existing only a minor static error in the final yaw angle and altitude values. The final value for the desired yaw angle is $3\ [^\circ/s] \times 5\ [s] = 15\ [^\circ]$ and in the simulation the value is closer to $16\ [^\circ]$. The final desired altitude is $4\ [m/s] \times 5\ [s] = 20\ [m]$ and the simulation result is also a bit higher than $20$. These errors are related with the lack of robustness to the noise of the modeled sensors, resulting in an oscillatory behaviour for the vertical inertial velocity $v_z$, and consequently for the final altitude value. The Euler angles curves also present small oscillations. The effect of noise is




(a) Euler Angles

(b) Inertial frame velocity

(c) Altitude

(d) Normalized Thrust Force

Figure 5.11: Type 2 simulation results for Ardupilot original controller.

also present in the inputs, producing an oscillatory response with considerable control action, which will result in a higher battery consumption.

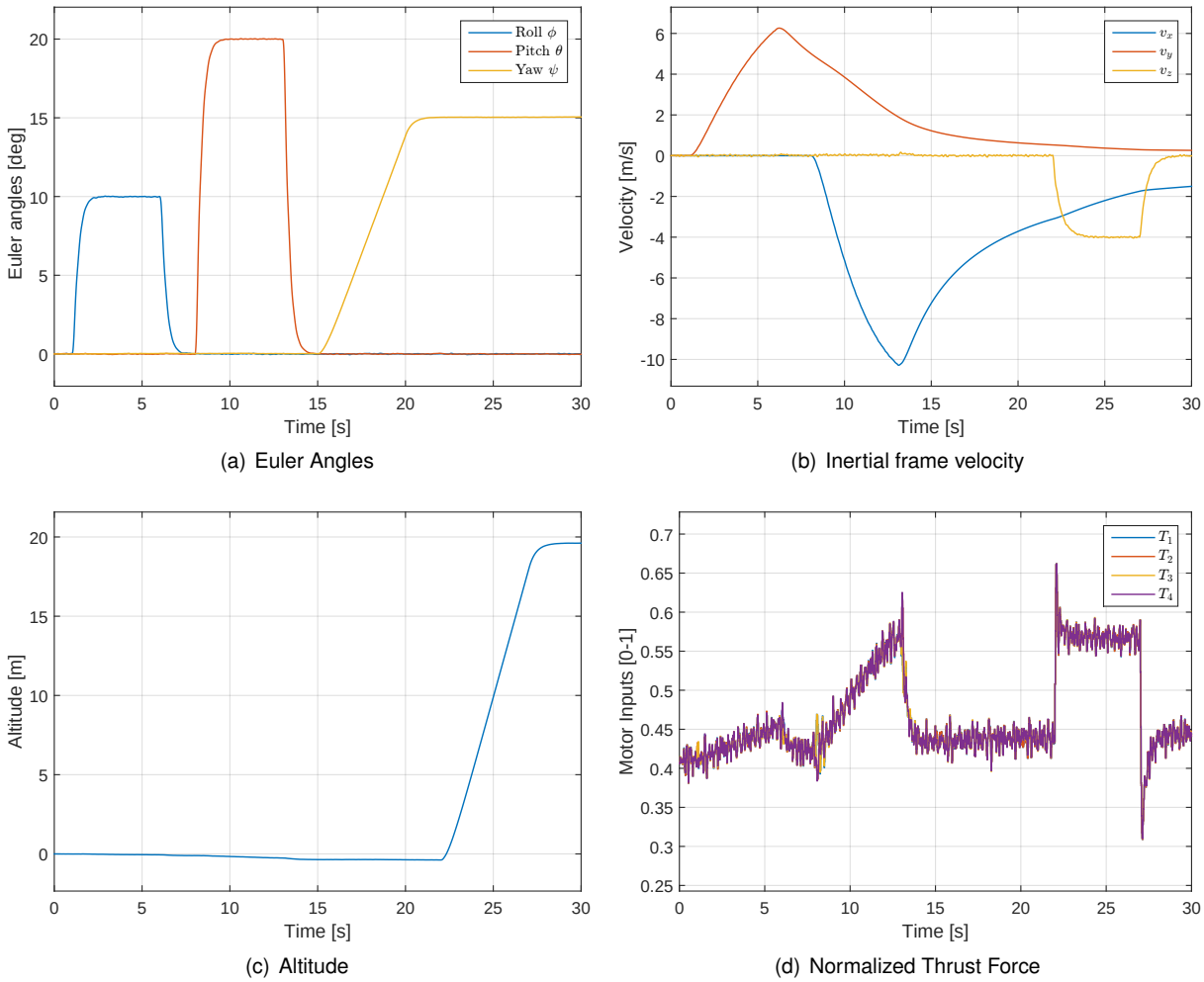
For the type 2 simulation the tracking was no longer achieved for all the references, given that the yaw rate is no longer controlled. The roll angle reached the desired reference of $10\ [^\circ]$, but the effect of noise is clear, given that the curve has high frequency oscillations throughout the simulation. It also possesses small disturbances that deviate the curve from its reference value several times during the simulation. The pitch angle is the channel with better tracking, presenting only minor overshoots during the reference steps. Finally, the oscillations in the vertical velocity curve are obvious, being more pronounced after the reference steps (after the $6\ [s]$), which result in some oscillations also in the altitude of the quadrotor, even though the desired final value of $16\ [m]$ is achieved.

Comparing the simulations of both the original and NDI controllers one can observe that the latter is more robust to noise, given that it produces simulation results with less oscillations. In type 1 simulations the two controllers achieved the desired tracking for all the channels. For type 2 simulations only the NDI controller guarantees tracking, with the original controller failing to reach the desired yaw rate. The explanation behind this fact lies in the saturation of the control action as one can see in the inputs graph, as well as in the couplings existing between the different control channels. The higher, and more

oscillatory, control action in the original controller is also related with the lack of robustness to noise. Consequently, the battery consumption will be higher for the original controller, which results in a lower flight time. Not only was the NDI controller validated, since it can reach the same results as the original controller, but it also has a better tracking and an expected lower battery consumption. In the following section the results for the INDI controller will be presented.

## 5.5 INDI Controller Simulation Results

The results of type 1 simulation with an INDI control design can be depicted in Figure 5.12. The block diagram that was used in MATLAB/Simulink environment is shown in Appendix B. The sampling frequency of the simulations was reduced from $400\ [Hz]$ to $50\ [Hz]$ because at the higher frequency the controller was not stable, due to the existence of a larger quantity of high frequency noise. A selection of the final sampling time, using comparisons between different values, will be addressed in section 5.6.2.




(a) Euler Angles

(b) Inertial frame velocity

(c) Altitude

(d) Normalized Thrust Force

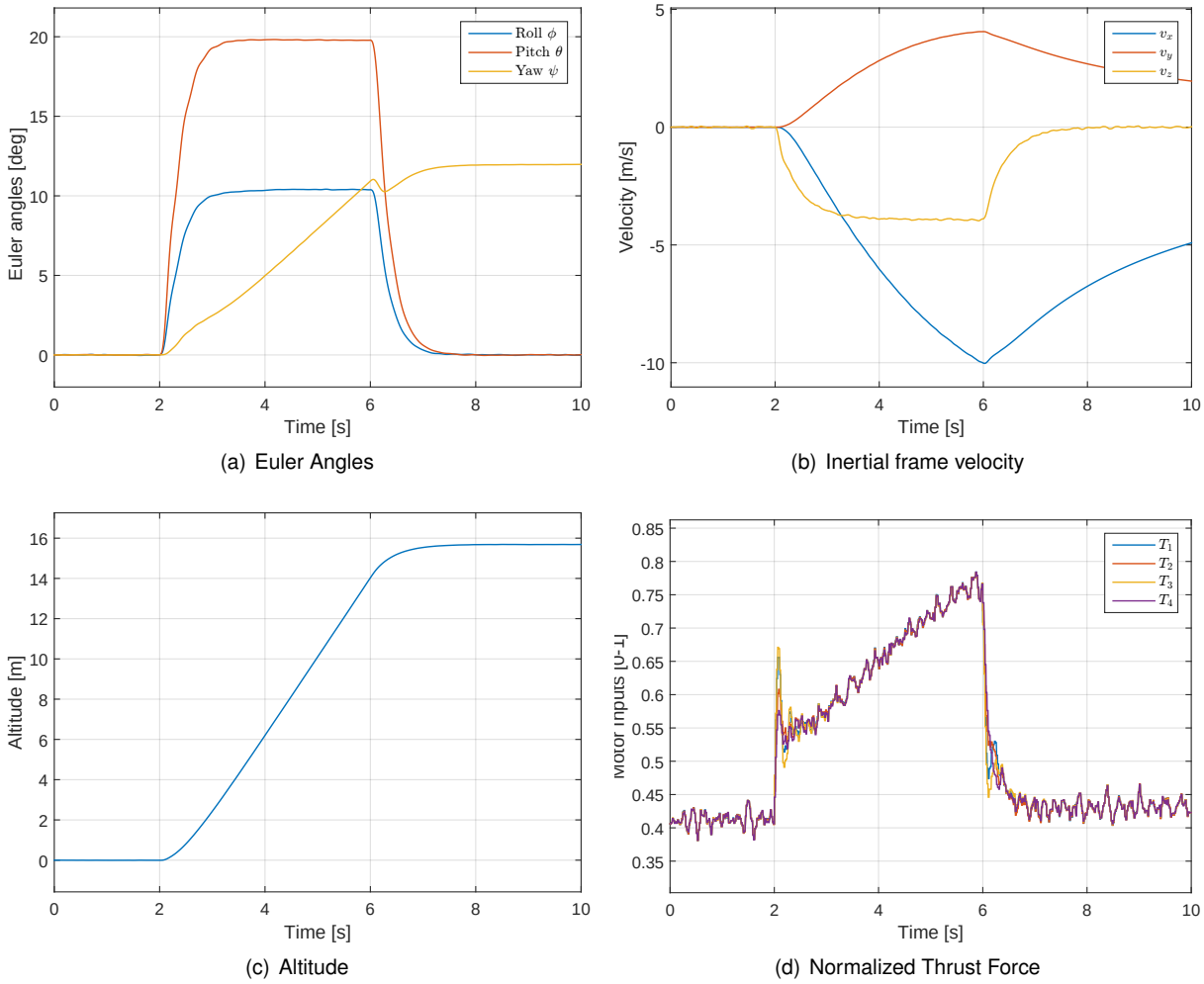
Figure 5.12: Type 1 simulation results for INDI controller.

All the references are followed with good performance in type 1 simulation, without the disturbances that were seen in the vertical velocity curve for the NDI controller, present in Figure 5.3. The disturbances were happening due to couplings created by inaccuracies in the model, which do not provided a

complete dynamic inversion. The INDI controller solves this problem, since it is less model dependent.

The control inputs are very similar in both controllers, with INDI having only a slightly more oscillatory input than NDI. These oscillations are related with the noise that exists in the measurement of the state derivatives. Even though, avoiding the dependence of a part of the model is more important, therefore, INDI results to type 1 simulation are considered preferable over the NDI results.

For the type 2 simulation INDI controller produced the graphs of Figure 5.13.




(a) Euler Angles

(b) Inertial frame velocity

(c) Altitude

(d) Normalized Thrust Force

Figure 5.13: Type 2 simulation results for INDI controller.

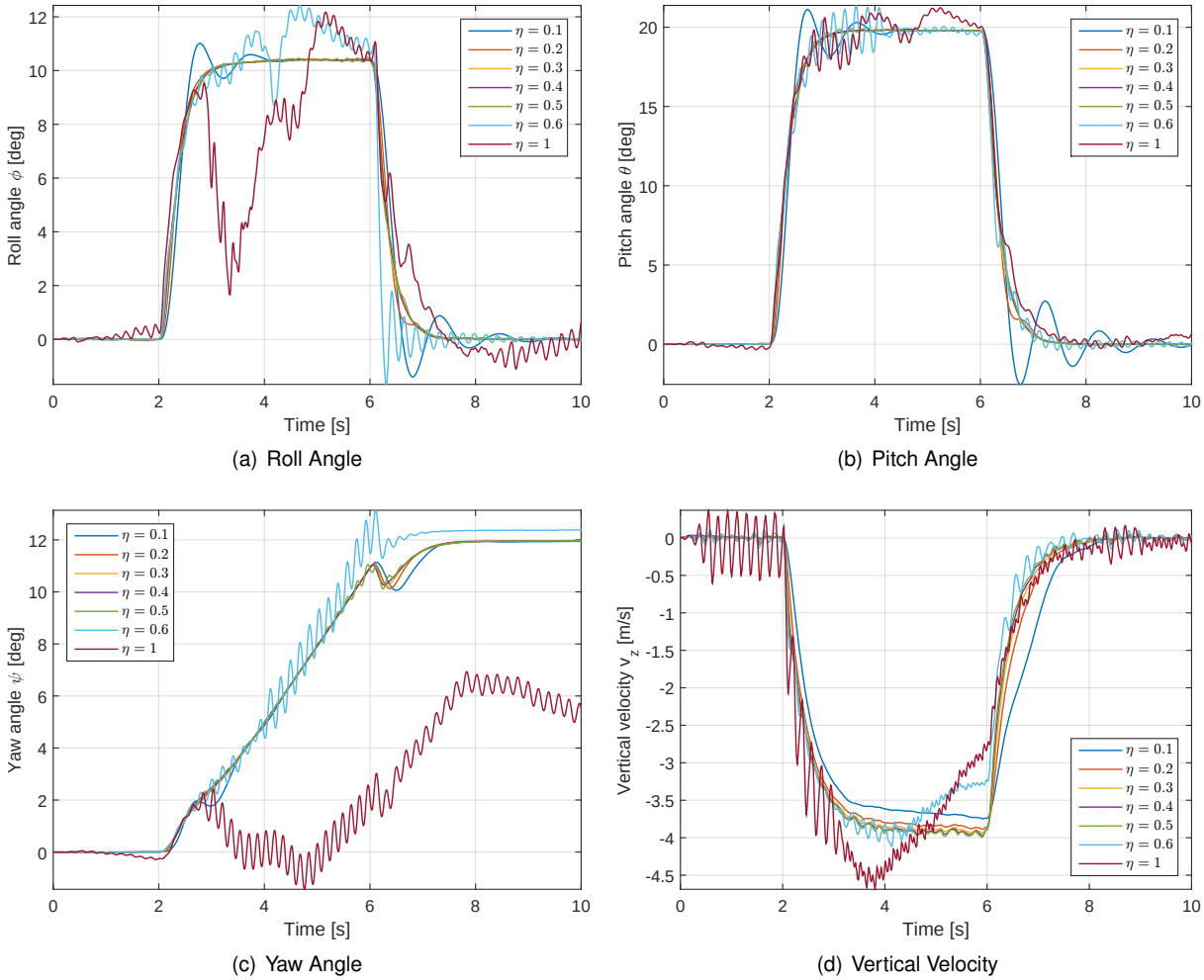
Similarly to type 1 simulation all the references are followed with good performance, in contrast to the NDI controller, where the desired vertical velocity reference was not followed. The control action is again very similar, being a bit smaller for INDI between 6 and 8 seconds of simulation, which in principle will lead to a smaller battery consumption. Therefore, once again INDI controller seems the best approach to solve the tracking problem of the Spyro quadrotor.

The next section will address the problem of choosing the best values for the input scaling gain ($\eta$) that scales the incremental input and for the sampling time.

## 5.6 Selection of Parameters

### 5.6.1 Input scaling gain

The importance of this parameter was explained in section 4.5. This section will present simulation results in MATLAB/Simulink for several different $\eta$ values, since $\eta = 0.1$ until $\eta = 1$. The results correspond to type 2 simulation with simultaneous inputs, since it is the most demanding simulation type. The graphs with comparisons for Euler angles and vertical velocity are shown in Figure 5.14. The complete simulations, including the altitude and the inputs, are depicted in Appendix C.1.




(a) Roll Angle (b) Pitch Angle

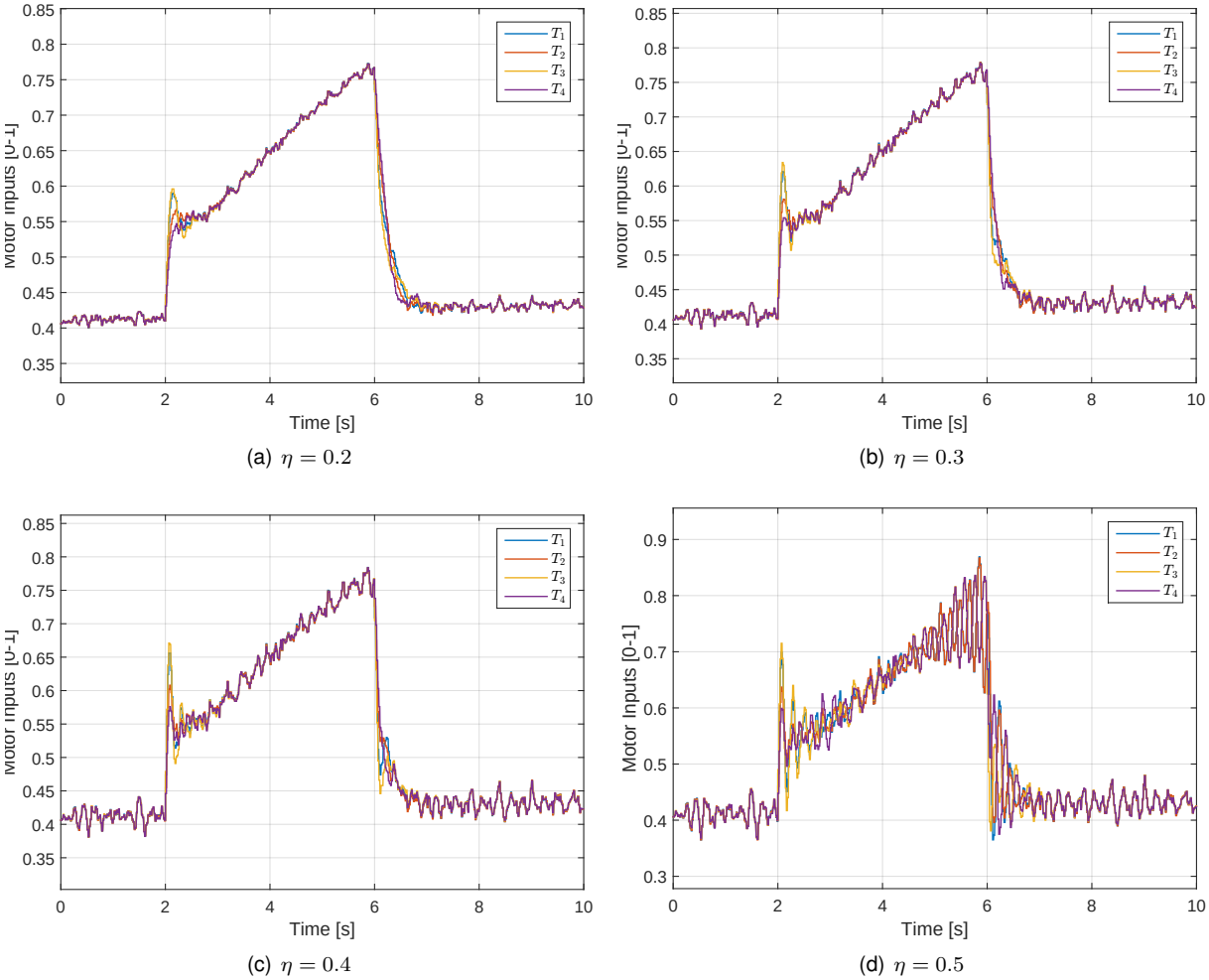
(c) Yaw Angle (d) Vertical Velocity

Figure 5.14: Parametric analysis for $\eta$ with type 2 simulation.

Analyzing the Figure 5.14 it is possible to conclude that the best value for $\eta$ is located between 0.2 and 0.5. For $\eta = 0.6$ the incremental input $\mathbf{\Delta u}$ is not sufficiently attenuated and the response starts to get oscillatory, which emphasizes the disturbances, provoked by possible model errors in matrix $\mathbf{G}(\mathbf{x})$ or by the presence of noise in the estimation of state derivative $\dot{\mathbf{x}}_\mathbf{0}$. When there is no attenuation in the incremental input ($\eta = 1$) the response is even more oscillatory and the tracking is no longer guaranteed for all the variables. Finally, for $\eta = 0.1$ the control has too much attenuation and tracking errors, as well as little overshoots, start to appear.

To choose the best value it is important to compare the inputs for the best values (from $\eta = 0.2$ until

$\eta = 0.5$), in order to predict which value will result in a smaller battery consumption. The graphs for the inputs are depicted in Figure 5.15.




(a) $\eta = 0.2$ (b) $\eta = 0.3$

(c) $\eta = 0.4$ (d) $\eta = 0.5$

Figure 5.15: Normalized Thrust Force for $\eta = 0.2$ until $\eta = 0.5$.

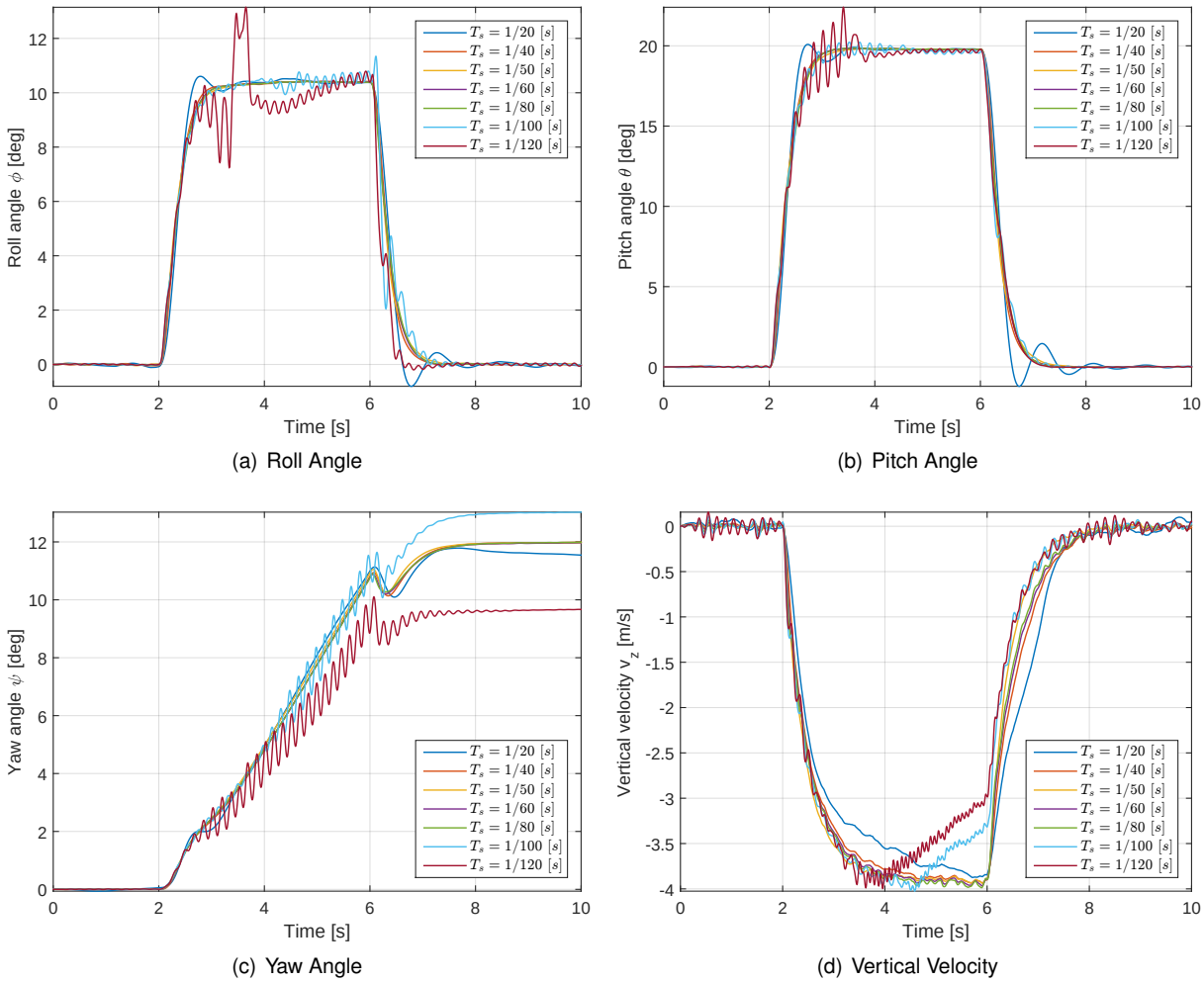
From the graphs with the inputs, one can conclude that $\eta = 0.2$ and $\eta = 0.3$ are the best values for this parameter, since they are the ones that present less oscillatory behaviour in the control inputs. Between these two values the higher one will be chosen, since it is the one that in theory produces smaller static errors in the tracking. Even though the errors are not obvious in this specific simulation they may appear in scenarios with different tracking requirements.

### 5.6.2 Sampling Time

The selection of sampling time follows the same strategy used to choose the best value for $\eta$, namely, a parametric analysis with different sampling times will be performed for a type 2 simulation in MATLAB/Simulink. The tested values vary between sampling frequencies of $120\ [Hz]$ and $20\ [Hz]$, which corresponds to sampling times since $T_s = 0.0083\ [s]$ until $T_s = 0.05\ [s]$.

As explained in section 4.5 a compromise is needed for the sampling time, since higher values result in state derivative estimations with more noise, but a high sampling time is required for an INDI application, so that part of the model can be disregarded. The comparisons for Euler angles and vertical

velocity can be seen in Figure 5.16, with the complete simulation results present in Appendix C.2.




(a) Roll Angle (b) Pitch Angle

(c) Yaw Angle (d) Vertical Velocity

Figure 5.16: Parametric analysis for $T_s$ with type 2 simulation.

Observing the comparisons between the different sampling times one can conclude that a better response is reached from $T_s = 1/40$ $[s]$ until $T_s = 1/80$ $[s]$. For lower sampling times the filter that obtains the derivative estimates ($\dot{\mathbf{x}}_\mathbf{0}$) gives a noisy response and consequently the dynamics of the closed loop system with the controller is affected, resulting in oscillatory simulations. With $T_s = 1/20$ $[s]$ the controller is too slow and does not produce a response that is fast enough to control the dynamics of the system.

Between the best four values, the chosen one will be $T_s = 1/50$ $[s]$ since it is low enough to obtain good estimates from the filters, not getting noisy outputs. It can be also better than $T_s = 1/40$ $[s]$ in scenarios where a faster control is needed. For instance, if with another tracking requirements the system exhibits a faster dynamics, then a faster control represents an advantage.

## 5.7 Robustness Tests

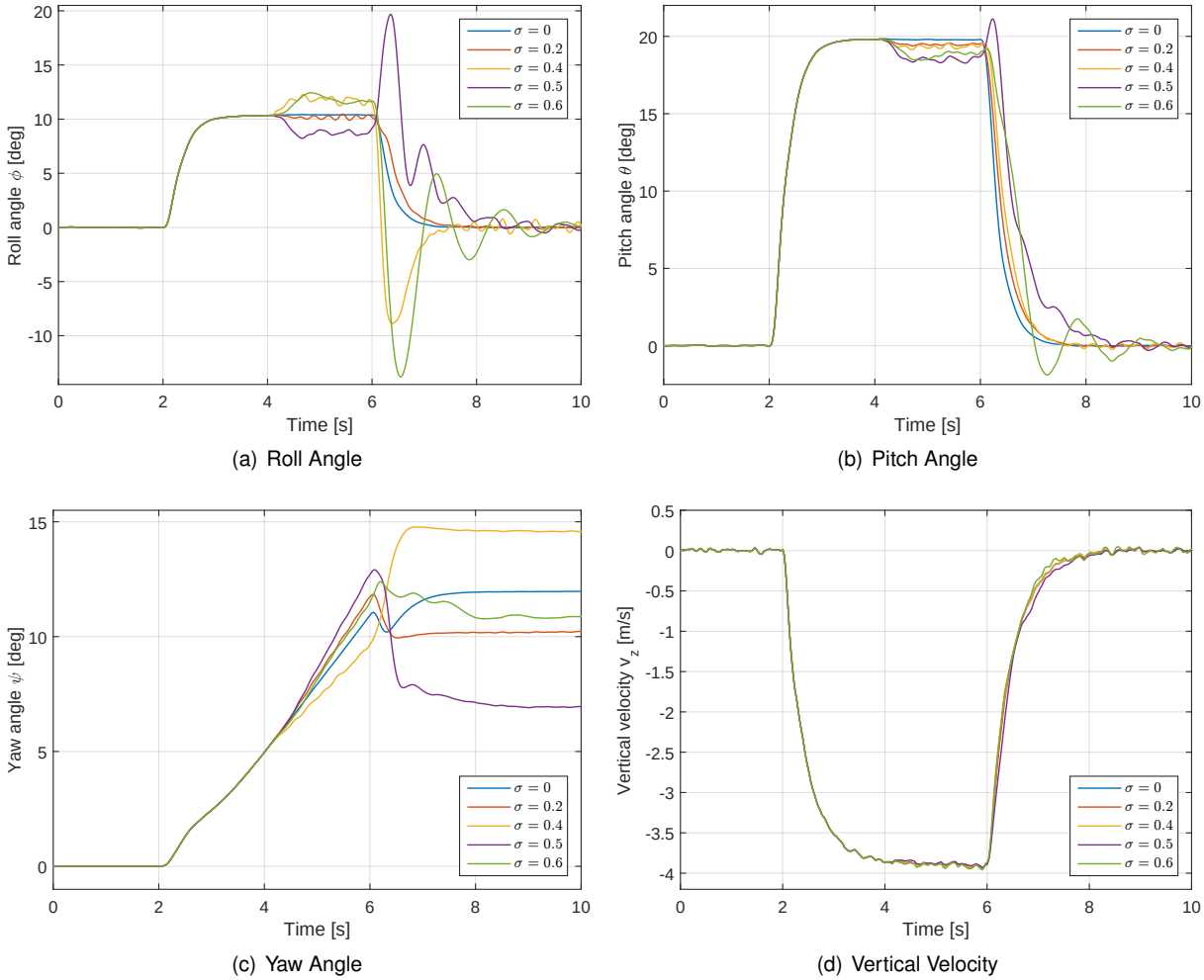
This section presents the results of robustness tests to the most common disturbances in a quadrotor flight, namely, to model uncertainties, wind velocity and loss of performance in an motor.

### 5.7.1 Model uncertainties

In the majority of the situations it is rather difficult to estimate a complete dynamic model of a quadrotor, due to its complexity. Therefore, great part of the models include several simplifications, resulting in an estimated model that presents small differences relative to the real quadrotor model. Hence, the controller must present robustness to these model uncertainties to achieve a good flight performance. Robustness to model uncertainties is also relevant to keep a stable and efficient flight when there are variations in mass, moments of inertia or other geometric parameters, allowing one to use the same controller with different quadrotor variants, without adjusting any parameter.

To simulate the model uncertainties a variation in matrix $\mathbf{G}(\mathbf{x})$ was performed by including a $4 \times 4$ normal distribution $\mathbf{\Delta G} \sim N(\mu, \sigma^2)$ with mean $\mu = 0$ and varying variance $\sigma^2$. With this variation model the resultant matrix $\mathbf{G}(\mathbf{x})_\mathbf{v}$ is related with the original control effectiveness matrix by Equation (5.7). The results for a type 2 simulation in the presence of model uncertainties with several different standard deviations, from $\sigma = 0$ (i.e., the nominal case) until $\sigma = 0.6$, are depicted in Figure 5.17. The uncertain model is activated at $4$ $[s]$ of simulation.

$$\mathbf{G}(\mathbf{x})_\mathbf{v} = \mathbf{G}(\mathbf{x})\,[1 + \mathbf{\Delta G}] \tag{5.7}$$




(a) Roll Angle

(b) Pitch Angle

(c) Yaw Angle

(d) Vertical Velocity

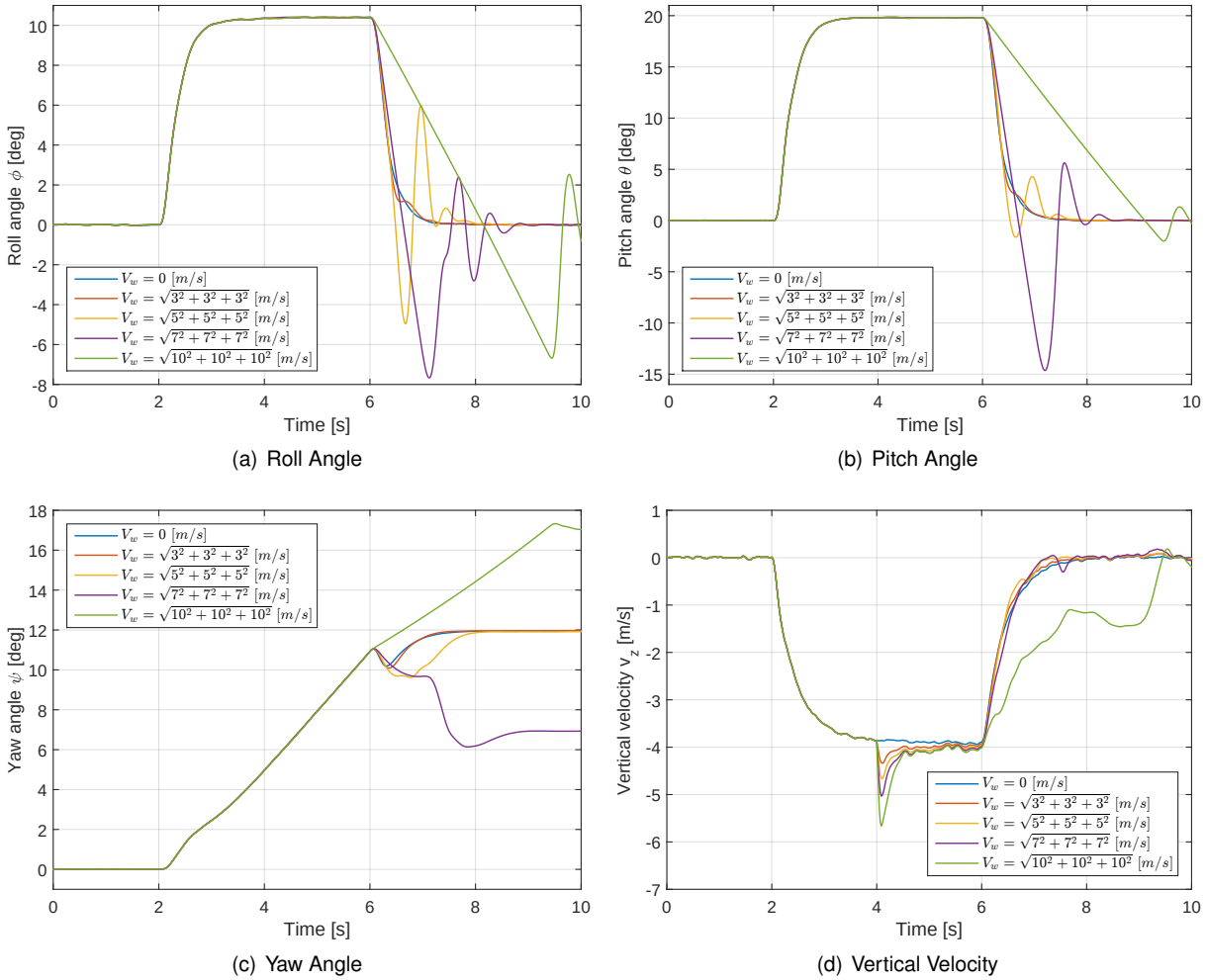
Figure 5.17: Robustness test to model uncertainties with type 2 simulation.

Analyzing Figure 5.17 one can conclude that pitch angle and vertical velocity are the two channels that are more robust to model uncertainties, with the first presenting small oscillations after the second step (i.e., $6\ [s]$), for $\sigma > 0.5$. The latter does not present effects from the model uncertainties, even at $\sigma = 0.6$. The roll angle presents considerable oscillations for $\sigma \geq 0.4$. In the presence of uncertainties the final value for the yaw angle deviates from the nominal situation.

### 5.7.2 Wind Velocity

Wind is usually present in outdoor flights, being a common source of disturbances. Thus, it is interesting to test the robustness of the control system to this type of disturbances. The wind is simulated using a specific input of the MATLAB/Simulink model, described in section 5.1. The wind velocity input is expressed in the inertial NED frame, with the same magnitude for all the axes, therefore, it can be expressed by Equation (5.8). Different values of $v_c$ were tested, from $v_c = 0\ [m/s]$ (i.e., with no wind), until $v_c = 10\ [m/s]$, which gives a total wind speed of $V_w \approx 17.32\ [m/s]$. The results for a type 2 simulation can be seen in Figure 5.18, where the wind input is activated at $4\ [s]$ of simulation.

$$V_w = \sqrt{v_c^2 + v_c^2 + v_c^2} \tag{5.8}$$




(a) Roll Angle

(b) Pitch Angle

(c) Yaw Angle

(d) Vertical Velocity

Figure 5.18: Robustness test to wind velocity input with type 2 simulation.

Observing Figure 5.18 it is interesting to notice that only the vertical velocity presents a small disturbance, proportional to $v_c$ value, when the wind input is activated. The remaining variables only present noticeable perturbations after the second step at $6$ $[s]$. The Euler angles degrade their performance, presenting oscillatory behaviour, for $v_c \geq 5$ $[m/s]$. The vertical velocity starts to present oscillations for $v_c = 10$ $[m/s]$. It is worth mentioning that for the higher wind velocity input (i.e., $v_c = 10$ $[m/s]$) the rate of change of the Euler angles during the second step decreases. If one analysis the inputs for this simulation (Figure 5.19) it is possible to conclude that the effect of the wind is so strong that even with the input at $0\%$ it is not possible to decrease the Euler angles at the same rate as the one used in the remaining simulations.

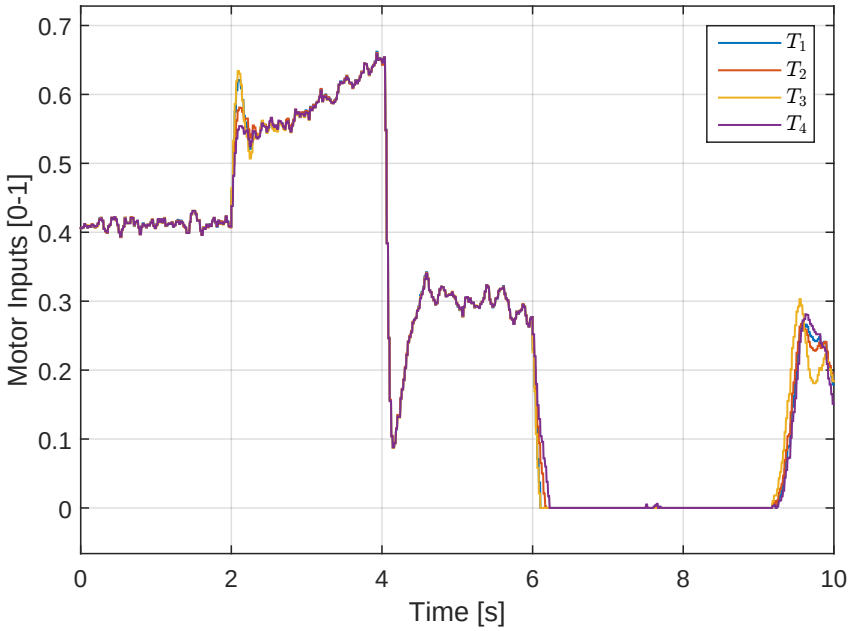


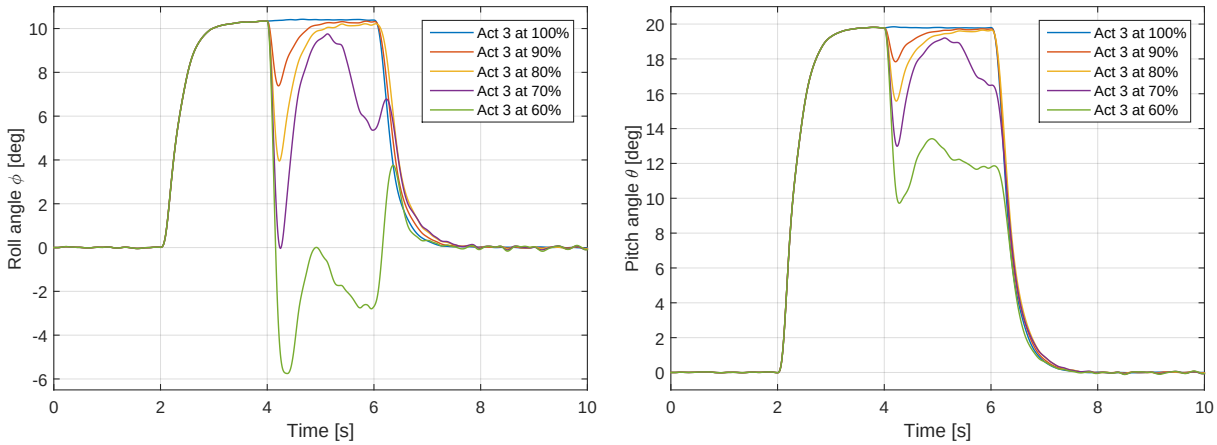
Figure 5.19: Normalized Thrust Force for wind velocity robustness test with $v_c = 10$ $[m/s]$.

### 5.7.3 Loss of Actuator Effectiveness

Besides model uncertainties and wind velocity, disturbances can also appear due to the loss of of effectiveness in the actuators. To simulate this effect the power provided to motor number three is decreased in a varying percentage, from $0\%$ until $40\%$, relative to the nominal value provided by the INDI control algorithm. Once again, the loss of effectiveness in the motor is activated at $4$ $[s]$ of simulation. The results for a type 2 simulation are depicted in Figure 5.20.

The roll and pitch angles present a small negative transient when the loss of effectiveness is activated, recovering its value for an effectiveness of $80\%$ or higher. The yaw angle always presents a static error proportional to the loss of efficiency since the controlled variable is the yaw rate, and even though its value return to the desired one, its integral (the yaw angle) will not reach the initial the same value. This problem can be fixed if a PI controller is introduced for the yaw rate in the linear part of INDI control algorithm. The vertical velocity presents also a transient after the activation of the disturbance, recovering its desired value for an effectiveness higher than $60\%$.

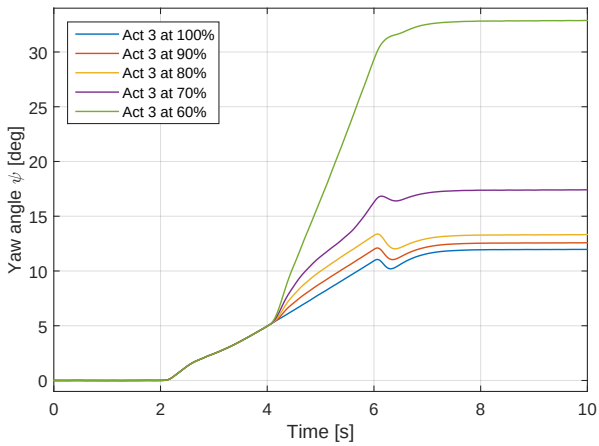
With this results, one can conclude that the INDI control algorithm presents some degree of fault



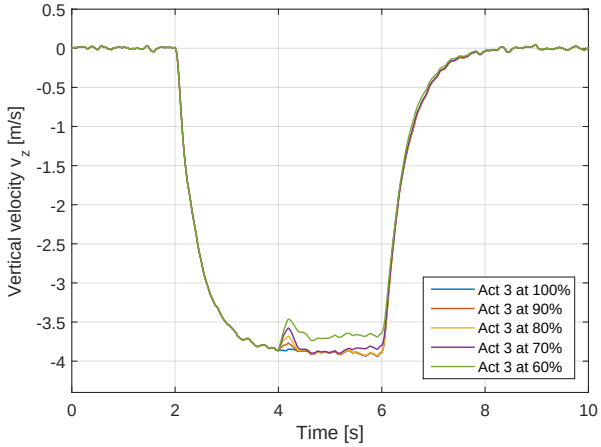

(a) Roll Angle




(b) Pitch Angle




(c) Yaw Angle




(d) Vertical Velocity

Figure 5.20: Robustness test to loss of actuator effectiveness with type 2 simulation.

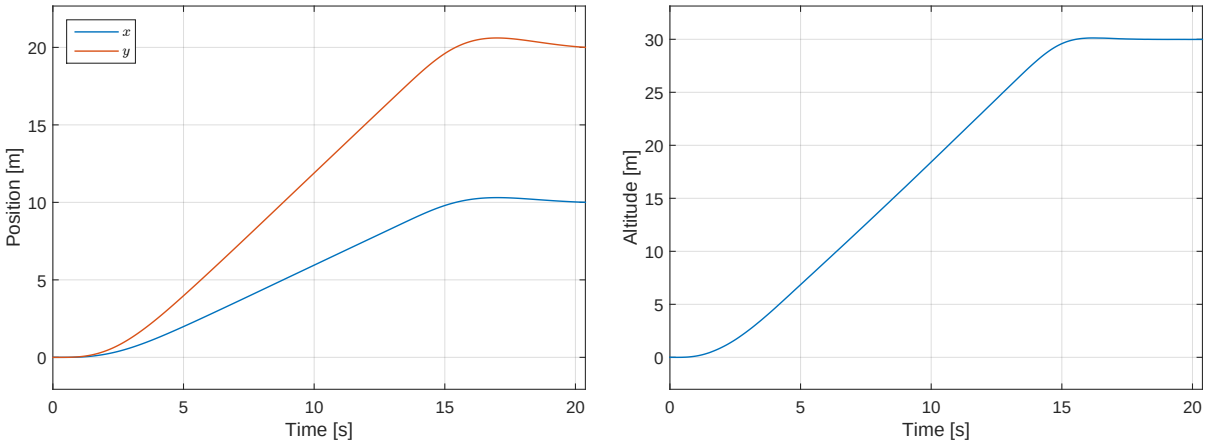
tolerance to loss of effectiveness in its motors. To have a better understanding of the limitations of this fault tolerance capability one can perform other simulations scenarios, with different manoeuvres/flight conditions and a different number of motors loosing its performance.

## 5.8 Position and INDI Controllers Simulation Results

After adjusting $\eta$ and sampling time parameters, the INDI controller was combined with the position control described in section 4.4. The resulting block diagram used in MATLAB/Simulink can be seen in Appendix B. To test this controller two different references were used, namely: a reference point located at $x = 10\ [m]$, $y = 20\ [m]$ and at an altitude $h = 30\ [m]$; an eight shape formed by to regular octagons also located at an altitude $h = 30\ [m]$. The controller sees the eight shape as a sequence of intermediate reference waypoints in the vertices of the octagons, passing to the next waypoint when the quadrotor reaches a distance smaller than $2\ [m]$ from the current reference. The quadrotor returns to home position (i.e. $x = 0\ [m], y = 0\ [m]$ and $z = 0\ [m]$) after all the waypoints are successfully reached.

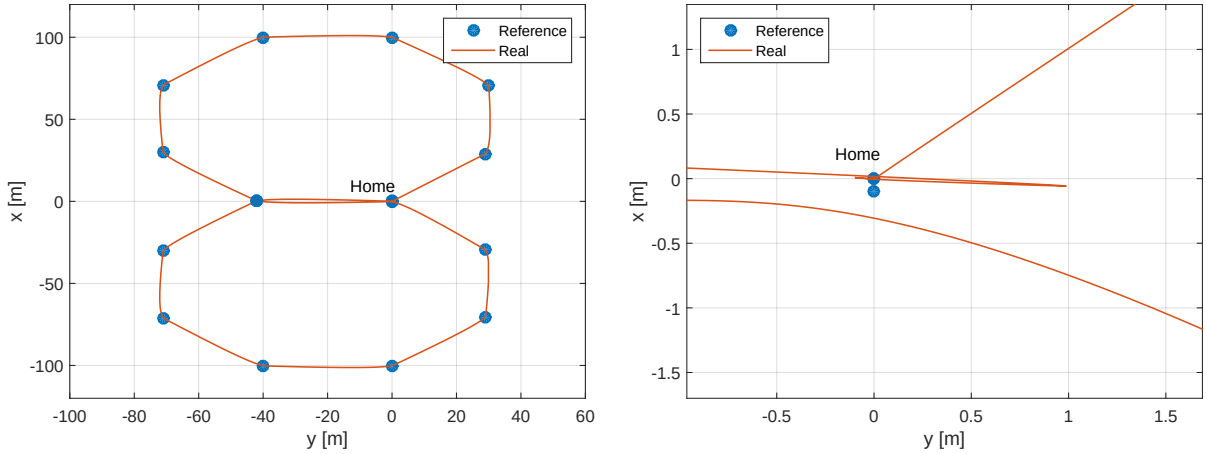
The simulation results with position information for the single waypoint reference are depicted in Figure 5.21 and the results for the tracking of the eight shape, including a 2D horizontal path and a

zoom around home position, are shown in Figure 5.22.




(a) Horizontal position (b) Altitude

Figure 5.21: Simulation results for waypoint tracking.




(a) Horizontal position (b) Zoom around home position

Figure 5.22: Simulation results for eight shape tracking.

Position tracking with the designed control law is achieved with good performance, as one can observe in the previous Figures. The tracking of a waypoint only presented a minor overshoot in $x$ and $y$ positions (Figure 5.21). In the eight shape tracking the path is successfully followed, with the zoomed path around home position allowing one to observe that during the final deceleration to the position above home the quadrotor presents some overshoot, which results in traveling further than the home position and then returning to it. Two dots are shown in the zoomed path since one indicates the initial home position and the other indicates the waypoint near home position, used as a reference in the middle of the simulation.

After a successful implementation of the control strategy in a MATLAB/Simulink environment the following chapter presents the implementation in Ardupilot autopilot system.

# Chapter 6

# Implementation and Results in Ardupilot

Accordingly to its development website ArduPilot (also known as APM) is an open source autopilot system supporting multi-copters, traditional helicopters, fixed wing aircraft and rovers [Ardupilot, 2017]. This chapter aims to present the structure and principles behind the operation of Ardupilot, thus a general description of the autopilot platform is given in the first section. The following two sections have the objective of presenting some adaptations that were necessary, both in Ardupilot operation and in the designed INDI control algorithm, in order to include the latter in the autopilot system. In the final section the results for the eight shape tracking, using the implemented controller, are shown and compared with the original control algorithm given by Ardupilot.

## 6.1 Presentation of Ardupilot platform

The code that constitutes the autopilot platform is written in C++ programming language and can be split into three different main categories: rover, copter and plane. The control structure that will be described corresponds to the copter category, since one wants to control a quadrotor. The control algorithm includes several flight modes, as an automatic flight mode that follows a sequence of predefined coordinates in Latitude-Longitude-Altitude (LLA) reference frame. It also has other flight modes such as altitude hold, where the roll and pitch angles are given by the pilot and the altitude is kept constant, as well as a complete manual mode where roll and pitch angles, yaw rate and throttle input are all provided by the pilot. Depending on the flight mode that is selected the control structure varies. Besides the control it also includes an Extended Kalman Filter (EKF) that filters the noise from some measurements and estimates new information using sensor fusion. Before a simulation or a flight starts the user can define several adjustable parameters used for different purposes, including: tune the autopilot controllers to the specified system, choose flight modes, enable certain features, among other purposes.

The most used flight mode is the automatic, therefore the new control algorithm will be implemented within it.

The main code loop for copter runs at $400\ [Hz]$ and performs the following actions:

- It reads information from sensors
- It runs a new iteration of the EKF
- It runs all controllers included in the specific flight mode
- It sends the new input action to the motors

The control structure is based on the position control described in section 4.4 and on the attitude and vertical velocity control, whose block diagram is depicted in Figure 5.9. It also includes an algorithm that transforms the latitude, longitude and altitude coordinates into $x$, $y$ and $z$ coordinates from the inertial frame. A complete block diagram of the control structure is in Figure 6.1.

Besides the control structure other tasks are running in separate threads, such as monitoring functions guaranteeing that information is being provided by the sensors, logging functions that save flight informations for posterior analysis, communication functions establishing a connection between the quadrotor and a ground control station, among other tasks. It is worth mentioning that Ardupilot possesses verification tests in almost every function, preventing the transmission of errors throughout the code. For instance, every time that a division is made the denominator is tested, in order to check if its value is close to zero, and the function follows an alternative to that division if indeed the denominator may lead to a result near infinity. In order to increase the accuracy of the calculations, the variables are multiplied by 100, resulting in higher numbers that suffer less from numerical errors. E.g. velocities are given in $[cm/s]$, positions in $[cm]$, angular rates in $[centidegrees/s]$ and angles in $[centidegrees]$.

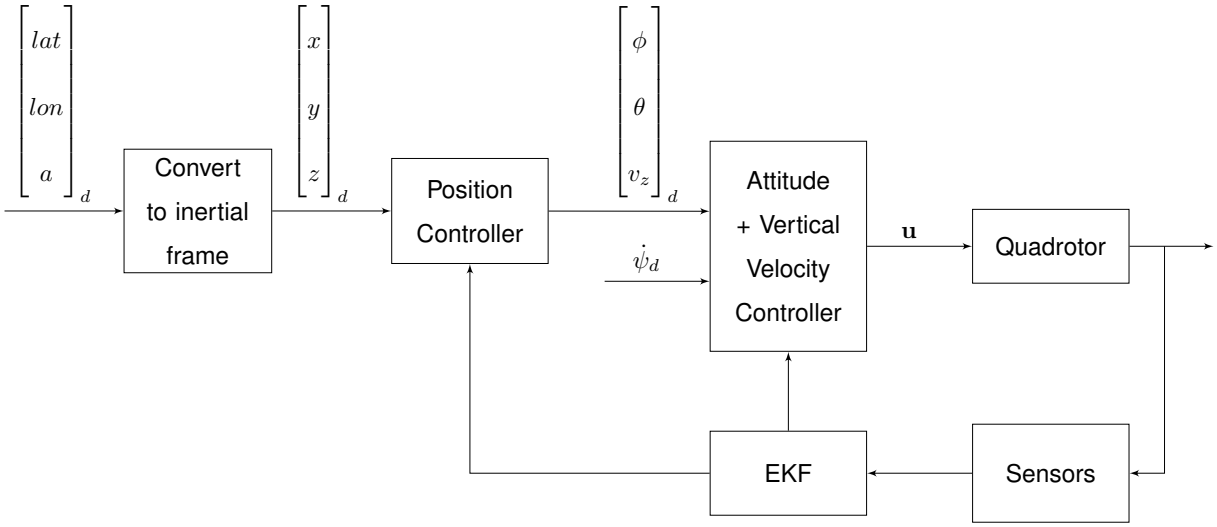


Figure 6.1: Block diagram of automatic mode control structure.

### 6.1.1 Simulation platform

It is also important to describe the simulation platform that will be used to verify the effectiveness of the new INDI control algorithm. Ardupilot has a software simulator, Software in the Loop (SITL), that can be run using a Linux operating system. This simulator was installed in Ubuntu 16.04 alongside

with QGroundControl, which is a software emulator for a Ground Control Station (GCS), allowing one to send commands to the quadrotor, as well as receive telemetry data from the simulated flights. In QGroundControl, it is also possible to change the value of the adjustable parameters from Ardupilot. The dynamics of the copter that is simulated is a general dynamics that cannot be adjusted, not being equal to the Spyro quadrotor. Therefore, if the control algorithm works correctly in this environment, it proves its robustness against model disturbances, given that the dynamic may be considerably different. The information saved in the flight logs will be sent to MATLAB, in order to produce graphical results similar to the ones shown in the previous chapter.

The following sections will introduce the implementation of INDI control algorithm into the automatic flight mode, including all the adaptations that were necessary, both in the INDI control algorithm and in the structure of the Ardupilot code.

## 6.2 Implementation of INDI control algorithm in Ardupilot

Since the INDI controller runs at a lower frequency than Ardupilot main code ($50\ [Hz]$ instead of $400\ [Hz]$) the new control algorithm was implemented in a new task, running on a separate thread at $50\ [Hz]$. The attitude and vertical velocity original control loops from Ardupilot were deactivated, keeping only safety checks that were still necessary. The position control algorithm did not suffer any modification, apart from sending their output values to the new INDI controller.

The main loop of the new controller task executes the following steps:

- It receives the position control references and divides them by 100, in order to transform the values to SI units
- It gets flight information from EKF and builds the state vectors that will be used as inputs for the low-pass and bandpass filters
- It updates both low-pass and bandpass filter estimations using a discrete version of the filters, implemented with finite differences (explained in detail in the following section)
- It computes outer INDI loop with linear controllers, obtaining a virtual input $\nu$ (Equation (4.24))
- It calculates incremental input values ($\mathbf{\Delta u}$)
- It adds previous time step inputs ($\mathbf{u_0}$), in order to get the new input values

The values of control effectiveness matrix $\mathbf{G(x)}$, as well as the INDI gains and $\eta$ were defined as new adjustable parameters, allowing its adjustment in a GCS during flight, and therefore providing an additional degree of flexibility to the implementation. The mathematical library of Ardupilot only has built in functions for vectors with length three. Given that INDI control is made using four dimensional vectors all the mathematical operations that were necessary, namely, addition of two vectors, multiplication between vectors and matrices, cross product of two vectors, multiplication of a vector by a scalar and inversion of a four by four matrix had to be built.

This new control algorithm is not used during takeoff and landing, given that the original controllers for these two flight phases are completely different from the controllers of the remaining flight phases, thus it is not possible to include the designed INDI control law. For this reason the original controllers for takeoff and landing will be used instead, with the quadrotor changing to INDI when it reaches the first waypoint, moving back to original controller before it starts the descending path to land. In order to ensure a safe transition between controllers the current values of the motors, computed with the original controller, are taken as the first input $\mathbf{u_0}$. The bandpass filter estimating the derivative of INDI state and the low-pass filter used to synchronized the time delay between samples start their execution some iterations before the transition, having in this manner better first estimates that already passed the transient output that exists during the first iterations of the filter. The implementation of these filters is described in the following section.

It is worth mentioning that this limitation is related with the control architecture used in Ardupilot and not with the application of INDI control solution.

## 6.3 Discrete Filter Implementation using Finite Differences

To apply the low-pass (LP) and bandpass (BP) filters in Ardupilot one needs to start by converting them from the Laplace domain to the time domain, which originates the set of Equations (6.1), where subscript $k$ represents the current time step. Rearranging the terms one can write the equations with respect to the output of the filters $y_k$, as seen in equations (6.2).

$$BP : H(s) = \frac{\omega_n^2 s}{s^2 + 2\xi\omega_n s + \omega_n^2} \Leftrightarrow y_k'' + 2\xi\omega_n y_k' + \omega_n^2 y_k = \omega_n^2 x_k' \tag{6.1a}$$

$$LP : H(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} \Leftrightarrow y_k'' + 2\xi\omega_n y_k' + \omega_n^2 y_k = \omega_n^2 x_k \tag{6.1b}$$

$$BP : y_k = x_k' - \frac{2\xi}{\omega_n} y_k' - \frac{1}{\omega_n^2} y_k'' \tag{6.2a}$$

$$LP : y_k = x_k - \frac{2\xi}{\omega_n} y_k' - \frac{1}{\omega_n^2} y_k'' \tag{6.2b}$$

The second step is to estimate the time derivatives of $x_k$ and $y_k$. For that purpose second order regressive finite differences, adapted from [Folgado, 2013], will be utilized to estimate both the first and second order time derivative. The second order finite difference is used to estimate the first order time derivative instead of a first order finite difference, given that it provides a more accurate estimate. The expressions for both first and second order time derivative estimates using finite differences are presented in expressions (6.3), where $h_1$ and $h_2$ are the last two time step intervals, with the latter representing the older time step. Since Ardupilot measures the exact time step, including some time fluctuations, $h_1$ and $h_2$ do not have necessarily the same value.

$$p'_k = \frac{p_{k-1} - p_{k-2}}{h_1} + \frac{\frac{p_k - p_{k-1}}{h_2} - \frac{p_{k-1} - p_{k-2}}{h_1}}{h_1 + h_2}(2h_2 + h_1) \tag{6.3a}$$

$$p''_k = 2\frac{\frac{p_k - p_{k-1}}{h_2} - \frac{p_{k-1} - p_{k-2}}{h_1}}{h_1 + h_2} \tag{6.3b}$$

Introducing the finite differences in Equations (6.2) and rearranging terms it is possible to reach an expression with respect to the output of the bandpass and low-pass filters, given in Equations (6.4). The symbol $T_h$ represents the sum of the last two step times, i.e. $T_h = h_1 + h_2$. Other simplifications were also made, namely: $x_0 = x_k - x_{k-1}$, $x_1 = x_{k-1} - x_{k-2}$, $y_0 = y_k - y_{k-1}$, $y_1 = y_{k-1} - y_{k-2}$.

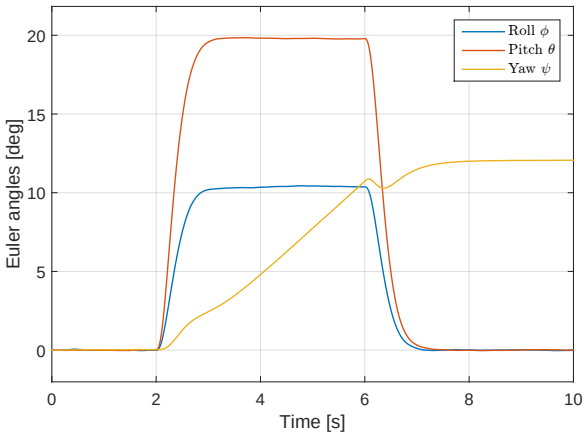
$$BP : y_k = \frac{\frac{x_1}{h_1} - \frac{2\xi y_1}{\omega_n h_1} + \frac{(T_h + h_2)}{\omega_n^2 T}\left(\frac{x_0}{h_2} - \frac{x_1}{h_1}\right) + \frac{2}{T_h \omega_n}\left(\frac{y_{k-1}}{h_2} - \frac{y_1}{h_1}\right)\left((T_h + h_2)\xi + \frac{1}{\omega_n}\right)}{1 + \frac{2}{h_2 T_h \omega_n}\left((T_h + h_2)\xi + \frac{1}{\omega_n}\right)} \tag{6.4a}$$

$$LP : y_k = \frac{x_k - \frac{2\xi y_1}{\omega_n h_1} + \frac{2}{T_h \omega_n}\left(\frac{y_{k-1}}{h_2} - \frac{y_1}{h_1}\right)\left((T_h + h_2)\xi + \frac{1}{\omega_n}\right)}{1 + \frac{2}{h_2 T_h \omega_n}\left((T_h + h_2)\xi + \frac{1}{\omega_n}\right)} \tag{6.4b}$$

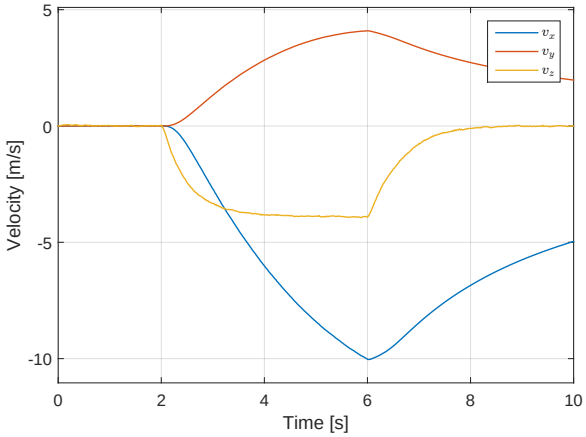
To prove that correct filter estimates are given by these formulas one implemented them in MATLAB/Simulink, replacing the second order filter blocks that were executing the same function. The results of a type 2 simulation (from Table 5.4) for the INDI controller is presented in Figure 6.2.

The simulation results do not have considerable differences, when one compares them with the ones obtained using the second order filter MATLAB/Simulink blocks (presented in Figure 5.13). All references are still successfully followed with good performance, existing slightly more oscillations in the control input. The oscillations are the result of more oscillatory bandpass filter derivative estimations, which are a natural consequence of the implementation, namely: finite differences are only a discrete approximation of the derivative, thus it is natural that they possess some estimation error. With these results one can consider that this discrete implementation of the filters using finite differences is suitable to be used in Ardupilot code.

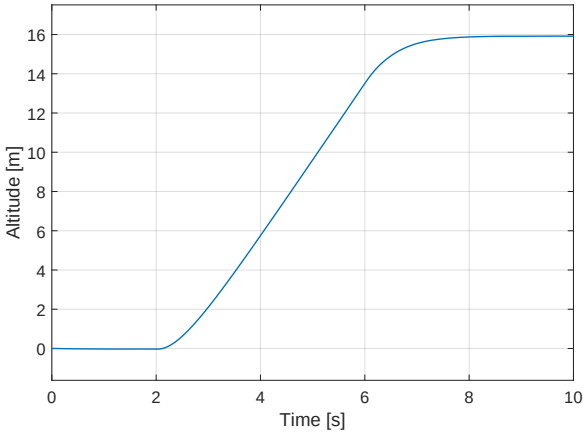
The previous formulas for the filters completed the description of the implementation that was performed in Ardupilot code. The following section presents results obtained in Ardupilot simulation platform for both the INDI and original control laws.



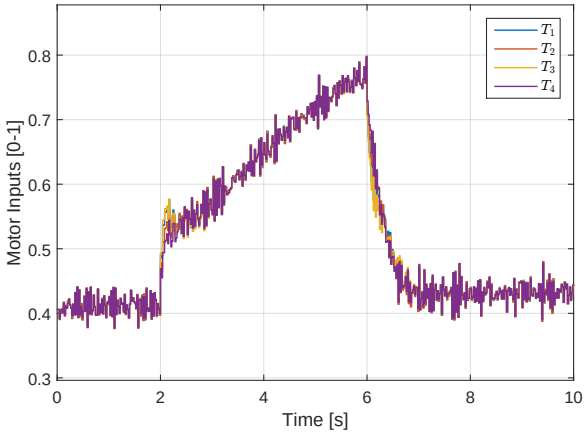

(a) Euler Angles




(b) Inertial frame velocity




(c) Altitude

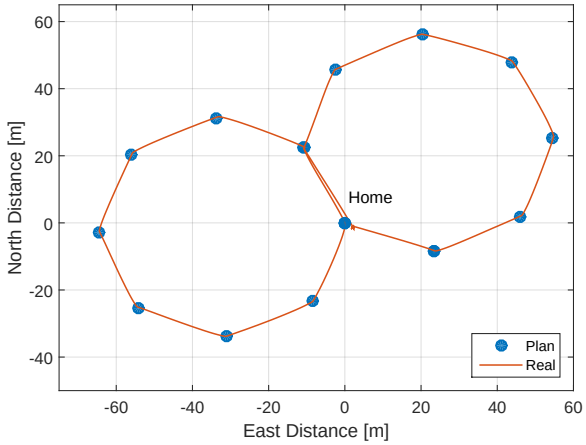



(d) Normalized Thrust Force

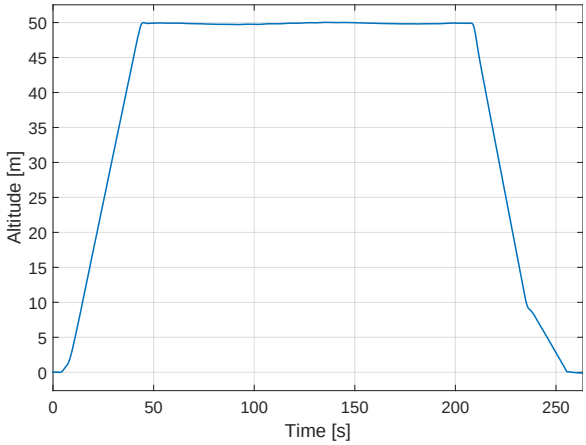
Figure 6.2: Type 2 Simulation for INDI controller with discrete derivative filter.

## 6.4 Ardupilot Simulation Results

A similar eight shape tracking was performed for INDI and original Ardupilot controllers using the simulation platform described in section 6.1.1. Complete simulation results, including 2D horizontal path, altitude, euler angles and inertial frame vertical velocity can be depicted in Figures 6.3 and 6.5 for INDI controller. The same information is shown in Figures 6.4 and 6.6 for the original controller .

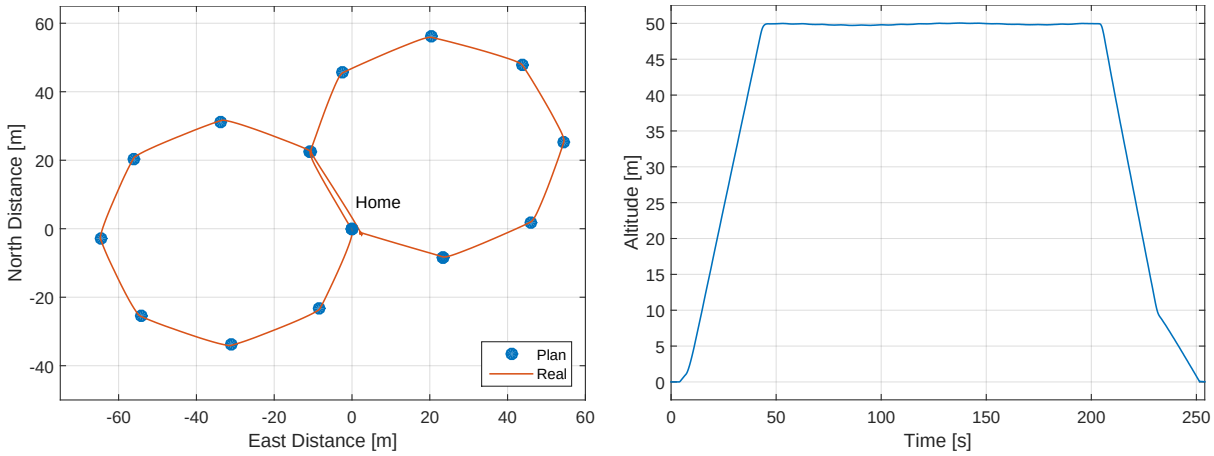



(a) 2D Horizontal path



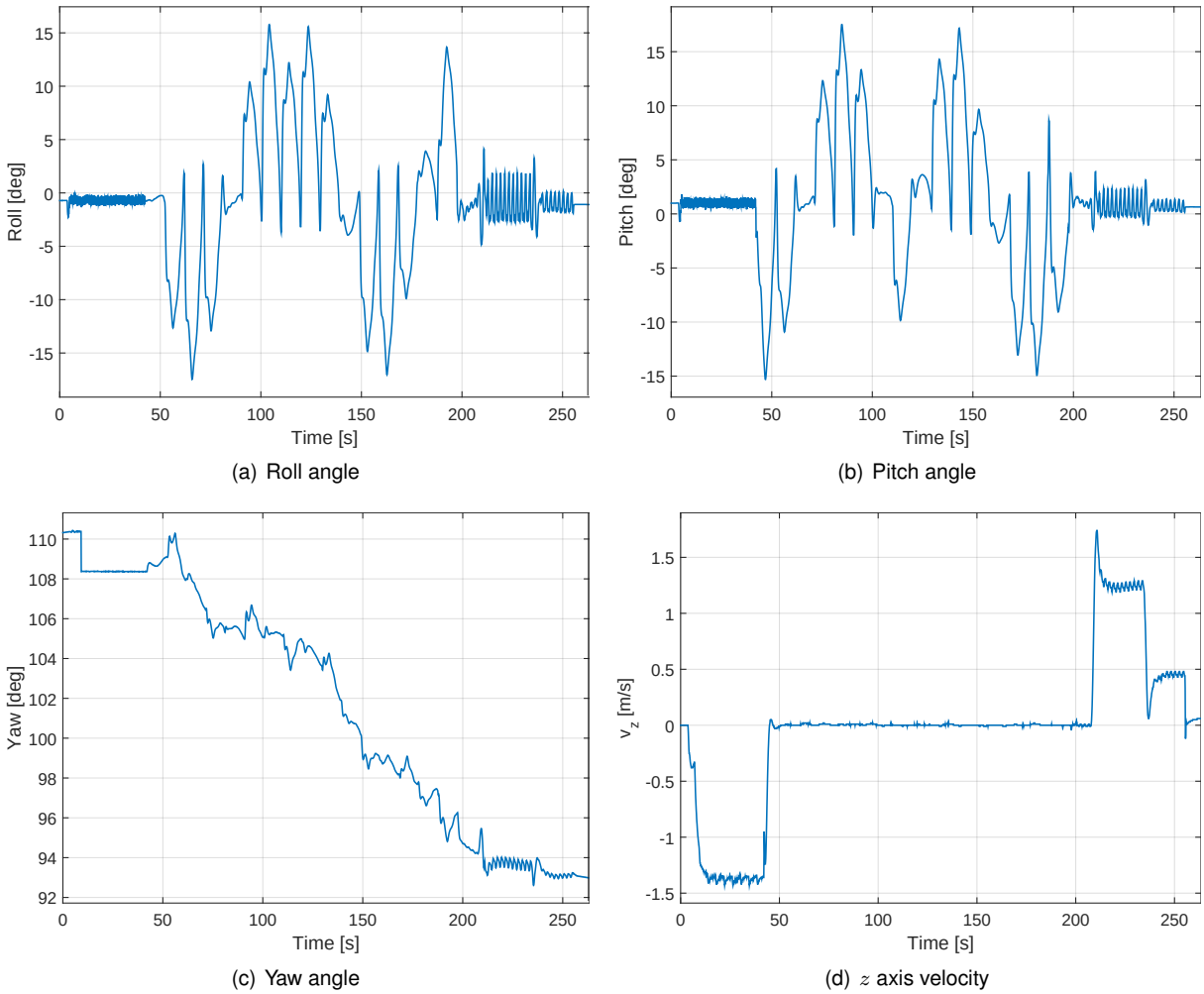

(b) Altitude

Figure 6.3: Position graphs of eight shape tracking with INDI controller.




(a) 2D Horizontal path (b) Altitude

Figure 6.4: Position graphs of eight shape tracking with original controller.



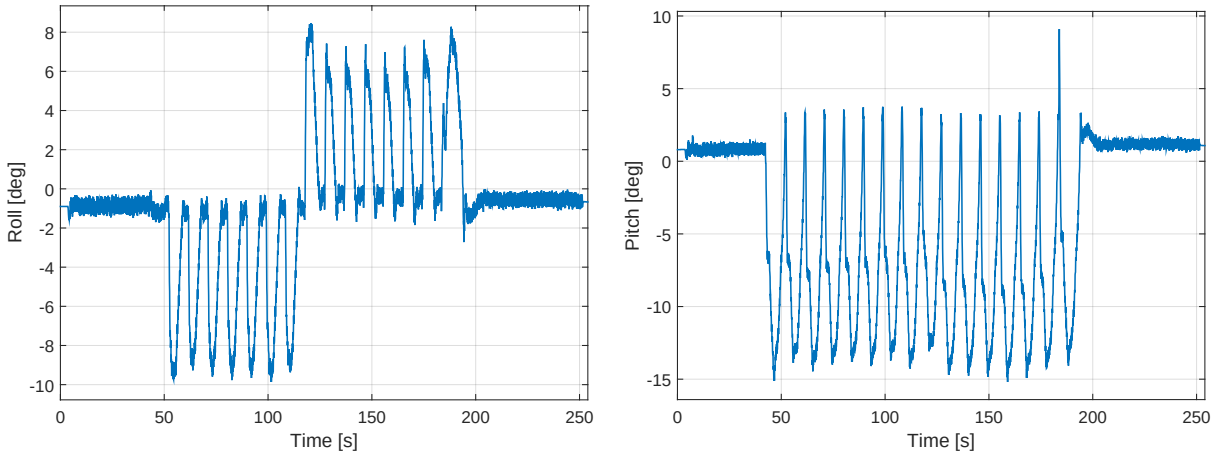

(a) Roll angle (b) Pitch angle

(c) Yaw angle (d) $z$ axis velocity

Figure 6.5: Euler angles and velocity graphs of eight shape tracking with INDI controller.

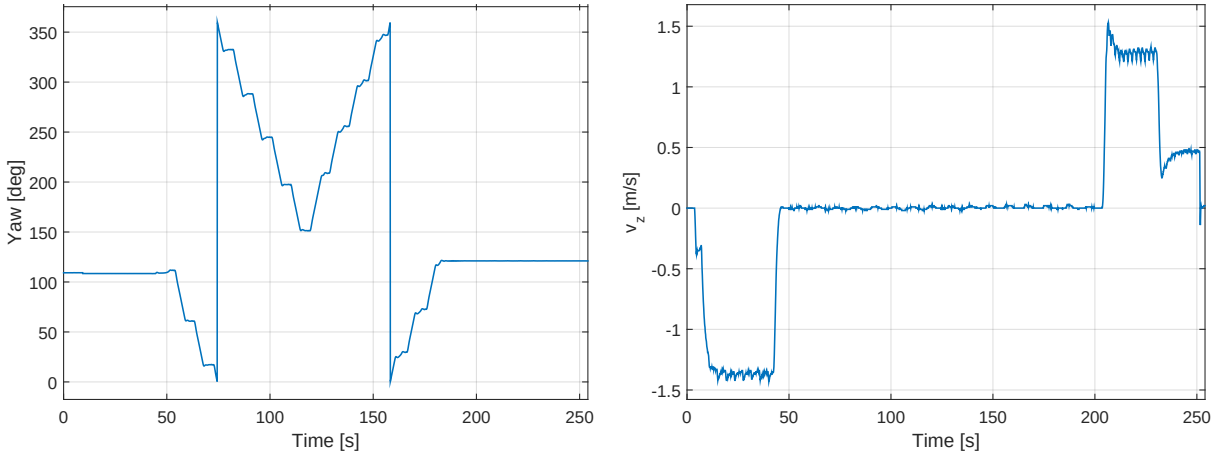
Analyzing the graphical representations of the 2D horizontal paths one can say that the tracking of the waypoints is successfully achieved with both controllers. The eight shapes are not vertical due to the orientation of the $x$ and $y$ axes of the inertial frame, which point to north and east, respectively, not matching the orientation of the eight shape.

To compare and analyze the altitude profiles the graphs presented in Figure 6.7 are more useful,




(a) Roll angle

(b) Pitch angle



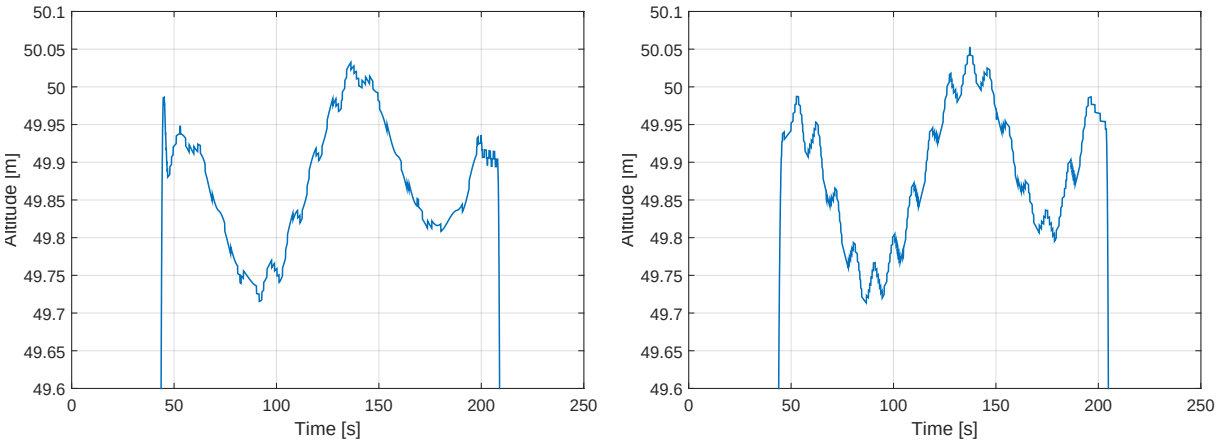

(c) Yaw angle

(d) $z$ axis velocity

Figure 6.6: Euler angles and velocity graphs of eight shape tracking with original controller.

since they only show the altitude profile in the region near $h = 50\ [m]$. By the analysis of these graphs it is possible to say that both controllers present similar disturbances in terms of altitude, therefore they should be related with the vertical position controller, given that it is the same for both control designs. The INDI controller presents an advantage, namely, the altitude profile presents smoother transitions, i.e. the peaks in waypoint transitions are smaller, or in other words the little peaks that are observable in both graphs are smaller for the INDI controller. The more pronounced peaks in altitude profile of the original controller will lead to a higher battery consumption.

Roll and pitch angles of both controllers do not match due to a different control approach in the tracking of a waypoint. The original controller steers the quadrotor front side to the waypoint, thus the initial situation in any waypoint is the same, which leads to the same peaks of roll and pitch angles for all waypoints, apart from an inversion in roll peaks at the middle of the simulation. This is caused by the inversion of turning side in the second octagon. When following waypoints in the first octagon the quadrotor is turning left and in the second octagon is turning right. INDI controller does not steer the quadrotor towards the waypoint, thus the roll and pitch peaks are different for all waypoints. In terms of amplitude the roll peaks are higher with INDI controller ($15\ [^\circ]$ against $10\ [^\circ]$), being both around $15\ [^\circ]$ for the pitch angle. With regard to the existence of high frequency oscillations they are similar in the




(a) INDI controller (b) Original controller

Figure 6.7: Altitude zoomed in graphs of eight shape tracking.

climb phase (i.e. in the first $45\ [s]$ when both angles are close to zero) because the controller is the same, only changing to INDI version when it reaches the cruise altitude of $50\ [m]$. During cruise altitude flight the original controller presents more oscillations than the new INDI controller. In the descent phase INDI controller presents oscillations with more amplitude, probably generated by some transient response after the second exchange of controller, but in original Ardupilot controller the frequency of the oscillations seems higher.

Due to the different approach on following a waypoint the yaw angles are also different. In the original controller its value is controlled, in order to steer towards the waypoint, decreasing when the quadrotor is turning left and increasing in the second octagon, when the quadrotor turns to the right side. The two leaps in the yaw angle, one close to $75\ [s]$ and the other close to $155\ [s]$ happen so that yaw angle is always given in interval $[0^\circ\ ,\ 360^\circ\ ]$. With INDI the yaw angle is not controlled, being free to follow a yaw rate request by the pilot. The roll and pitch angles that are necessary to follow the waypoints originate small changes in yaw during the simulation.

Finally, the vertical velocity curves are very similar, not presenting noticeable differences. It is worth notice that the velocity during the final descent phase decreases to a value close to zero, presenting then a small increase before it reaches zero. This behaviour is explained by a different controller that ardupilot uses in this flight phase. When the quadrotor is reaching the place where it has to land the velocity decreases, with the quadrotor almost hovering over the land place, having then a smooth landing with a smaller velocity. This controller guarantees that the impact on the ground is smaller, avoiding any damage to the quadrotor.

In conclusion, both controllers present very similar results, with INDI presenting smaller oscillations in some variables. The original controller also has the disadvantage of requiring more time to tune the large quantity of gains that exist in the several PID loops. Recalling that the simulation model presents a different dynamics than the Spyro quadrotor that was used to design the INDI controller, including the tuning of the gains, it proves that this control methodology possesses robustness to model changes. For the original controller one used the default gains of the Ardupilot simulator, in order to provide the best simulation for this controller, since these gains are adjusted for the simulation model. To test the

robustness to different disturbances (e.g. wind, sensor biases, loss of efficiency in the motors) one can upload the Ardupilot code to an autopilot board and perform experimental tests with a quadrotor, or include a model of these disturbances into the simulation platform.

# Chapter 7

# Conclusions

This final chapter presents the conclusions that were drawn throughout the thesis, as well as some relevant future work that can be further developed on the topic.

## 7.1 Achievements

In chapter 2 a review of relevant literature on the topic was performed and from it one can conclude that gain scheduling provides a rather simple control solution for the total flight envelope, but at the cost of a time consuming design process, and therefore a nonlinear control method seems a more interesting approach. Between the three main nonlinear techniques it was seen that Sliding Mode Control (SMC) provided an overall robust controller with good performance, though it requires a high control action, and consequently a larger battery consumption; Backstepping (BKS) has the advantage of being based in Lyapunov theory, thus it is possible to prove the stability of the implemented controller. However, it possesses a complex design process. Nonlinear Dynamic Inversion (NDI) was then the selected theory for the controller, due to its rather simple design process allied with interesting robust capabilities, specially in its incremental variant (INDI).

The background theory beyond NDI and INDI was described in chapter 3, from where it was possible to conclude that NDI requires a complete model of the system, whereas INDI only needs the control effectiveness matrix, providing in theory a better robustness to model uncertainties. However, for a successful implementation of INDI one needs an estimation of the state derivative and a high sampling frequency.

A description of the quadrotor model, as well as a practical implementation of NDI and INDI methods was presented in chapter 4.

Both methodologies, i.e. NDI and INDI, were successfully implemented in MATLAB/Simulink in chapter 5, proving its validity as solutions for an attitude and vertical velocity controller for a quadrotor UAV. These new methods could be run at a smaller frequency than the original PID control design from Ardupilot, thus requiring a smaller computational power. An efficient estimation for the state derivative in INDI controller was implemented using a second order bandpass filter, which not only estimates the

derivative but also filters some high frequency noise from sensor measurements. Including the INDI variant in the positional control law from Ardupilot also resulted in a controller with good performance.

The latter was also implemented with success in Ardupilot platform in chapter 6, including a new digital variant of the derivative estimation recurring to second order regressive finite differences that has never been seen in literature related with this topic. The chapter concludes with a comparison between the new control technique and the original controller from Ardupilot, showing that the INDI controller can match the performance of the original control design, which proves the feasibility of this new method in quadrotor control design, with the advantage of consuming less battery. In theory INDI requires a higher sampling rate to be applied, but in simulations it was possible to use this control solution with a lower sampling rate than the one used with the original controller, thus this limitation does not seem to exist in practical implementations. The model of the quadrotor that is present in Ardupilot simulation platform is different from the Spyro model for which the controllers were developed, therefore a good performance of the controller proves the robustness to model uncertainties.

In summary, this document achieved a successful implementation of INDI control technique for quadrotor attitude and vertical velocity control, using an input scaling gain ($\eta$) and a second order band-pass filter to estimate the state vector derivative. The implementation was also successfully incorporated in a professional grade autopilot system (Ardupilot), where it was capable of tracking a desired trajectory.

The developed code in Ardupilot is ready to be used in experimental tests, one only needs to upload the code to an autopilot board. An experimental validation using a real Spyro quadrotor was scheduled with UAVision, which could not only prove that the technique is a valid solution to solve the problem of quadrotor control, but could also test the robustness of this control method to a variety of disturbances that are invariably present in these kind of tests, e.g. wind velocity and actuator loss of performance with battery consumption. Unfortunately UAVision was not available to perform the experimental tests before the delivery deadline of this document.

## 7.2 Future Work

In this project a controller for attitude and vertical velocity was successfully implemented and tested in MATLAB/Simulink using NDI and INDI theory, with the latter being also implemented in Ardupilot platform. During the design process and the conclusions derived from the simulations one can propose some improvements or alternatives to be addressed in future research on this topic, namely:

- Include an adaptive algorithm that would estimate the control effectiveness matrix values in real time, which could account for some stationary disturbances affecting the quadrotor. With this algorithm the values for $\mathbf{G}(\mathbf{x})$ are no longer necessary to be estimated, turning the INDI control version completely model free, so that it could be, in theory, applied to any quadrotor.

- The implementation in Ardupilot was not introduced during takeoff and landing flight, since these two phases are using different control methods, that are not possible to be merged with the INDI controller. These controllers could be replaced by using one of two solutions, namely, develop

an INDI controller specially for these flight phases or design a positional control law using also a nonlinear controller. This new positional controller should have the possibilities of being merged with the existing controller and being applicable to all flight phases.

- Even though MATLAB/Simulink models a great part of the physical dynamics of the quadrotor, as well as the noise obtained from sensor measurements, a real time application with experimental tests would have provided results that could verify the level of robustness of the chosen control method to a variety of disturbances that were not modeled, e.g. loss of performance with battery consumption.

# Bibliography


P. Acquatella. *Robust Nonlinear Spacecraft Attitude Control*. Master's thesis, Delft University of Technology, Faculty of Aerospace Engineering, November 2011.

P. Acquatella, E. van Kampen, Q. P. Chu, et al. Incremental backstepping for robust nonlinear flight control. *Proceedings of the EuroGNC 2013*, 2013.

P. Acquatella, W. van Ekeren, and Q. P. Chu. Pi (d) tuning for flight control systems via incremental nonlinear dynamic inversion. *arXiv preprint arXiv:1701.08981*, 2017.

Ardupilot. Ardupilot development site. `http://ardupilot.org/dev/index.html`, 2017. Accessed: 2017-09-25.

R. C. Avram, X. Zhang, J. Muse, and M. Clark. Nonlinear adaptive control design and controller integrity monitoring for quadrotor uavs. In *Unmanned Aircraft Systems (ICUAS), 2016 International Conference on*, pages 301–309. IEEE, 2016.

J. Azinheira, A. Moutinho, and J. Carvalho. Lateral control of airship with uncertain dynamics using incremental nonlinear dynamics inversion. *IFAC-PapersOnLine*, 48(19):69–74, 2015.

B. J. Bacon and A. J. Ostroff. Reconfigurable flight control using nonlinear dynamic inversion with a special accelerometer implementation. In *AIAA Guidance, Navigation, and Control Conference and Exhibit*, page 4565, 2000.

J. Bowers. Search with aerial rc multirotor (s.w.a.r.m.). `http://sardrones.org/`, 2017. Accessed: 2017-09-30.

R. W. Brockett. Feedback invariants for nonlinear systems. *IFAC Proceedings Volumes*, 11(1):1115–1120, 1978.

F. Chen, R. Jiang, K. Zhang, B. Jiang, and G. Tao. Robust backstepping sliding-mode control and observer-based fault estimation for a quadrotor uav. *IEEE Transactions on Industrial Electronics*, 63 (8):5044–5056, 2016.

Q. P. Chu. Advanced flight control AE4311. Delft University of Technology, Faculty of Aerospace Engineering, 2016.

R. R. G. Coelho. *Quadrotor Control using Incremental Nonlinear Dynamics Inversion*. Master's thesis, Instituto Superior Técnico - Universidade de Lisboa, May 2017.

E. L. S. da Silva. *Incremental Nonlinear Dynamic Inversion for Quadrotor Control*. Master's thesis, Instituto Superior Técnico - Universidade de Lisboa, November 2015.

W. Dong, J. A. Farrell, M. M. Polycarpou, V. Djapic, and M. Sharma. Command filtered adaptive backstepping. *IEEE Transactions on Control Systems Technology*, 20(3):566–580, 2012.

D. Enns, D. Bugajski, R. Hendrick, and G. Stein. Dynamic inversion: an evolving methodology for flight control design. *International Journal of control*, 59(1):71–91, 1994.

G. P. Falconí, V. A. Marvakov, and F. Holzapfel. Fault tolerant control for a hexarotor system using incremental backstepping. In *Control Applications (CCA), 2016 IEEE Conference on*, pages 237–242. IEEE, 2016.

J. A. Farrell, M. Polycarpou, M. Sharma, and W. Dong. Command filtered backstepping. *IEEE Transactions on Automatic Control*, 54(6):1391–1395, 2009.

J. Folgado. Computational mathematics lecture notes. Instituto Superior Técnico - Universidade de Lisboa, 2013.

X. Gong, Y. Bai, C. Peng, C. Zhao, and Y. Tian. Trajectory tracking control of a quad-rotor uav based on command filtered backstepping. In *Intelligent Control and Information Processing (ICICIP), 2012 Third International Conference on*, pages 179–184. IEEE, 2012.

E. R. Johnston, F. Beer, and E. Eisenberg. *Vector Mechanics for Engineers: Statics and Dynamics*. McGraw-Hill, 9th edition, 2010.

S. Juliana, Q. Chu, J. Mulder, and T. Van Baten. The analytical derivation of nonlinear dynamic inversion control for parametric uncertain system. In *AIAA guidance, navigation, and control conference and exhibit*, pages 15–18, 2005.

I. Kanellakopoulos, P. V. Kokotovic, and A. S. Morse. Systematic design of adaptive controllers for feedback linearizable systems. In *American Control Conference, 1991*, pages 649–654. IEEE, 1991.

A. J. Krener. A decomposition theory for differentiable systems. *IFAC Proceedings Volumes*, 8(1):280–289, 1975.

N. Li, S. Yu, and Z. Xi. Nonlinear control design for a quad rotor unmanned aerial vehicle. In *Control Conference (CCC), 2016 35th Chinese*, pages 469–474. IEEE, 2016.

T. Lombaerts, P. Chu, and J. A. B. Mulder. Flight control reconfiguration based on online physical model identification and nonlinear dynamic inversion. In *Fault tolerant flight control*, pages 363–397. Springer, 2010.

R. Lopez, S. Salazar, A. Martinez-Vasquez, I. Gonzalez-Hernandez, and R. Lozano. Altitude control of a quad-rotor using adaptive sliding mode. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1111–1116, June 2016. doi: 10.1109/ICUAS.2016.7502654.

P. Lu, E. Van Kampen, and Q. P. Chu. Robustness and tuning of incremental backstepping approach. In *AIAA Guidance, Navigation, and Control Conference*, page 1762, 2015.

R. Marino. On the largest feedback linearizable subsystem. *Systems & Control Letters*, 6(5):345–351, 1986.

M. Mazur. Six ways drones are revolutionizing agriculture. `https://www.technologyreview.com/s/601935/six-ways-drones-are-revolutionizing-agriculture/`, 2017. Accessed: 2017-09-30.

D. Mercado, R. Castro, and R. Lozano. Quadrotors flight formation control using a leader-follower approach. In *Control Conference (ECC), 2013 European*, pages 3858–3863. IEEE, 2013.

A. B. Milhim. Modeling and fault tolerant pid control of a quad-rotor uav. *Master's Thesis, Concordia University, Montreal, Quebec, Canada*, 2010.

K. Ogata. *Modern Control Engineering*. Prentice Hall, 5th edition, 2010.

M. Oosterom and R. Babuška. Design of a gain-scheduling mechanism for flight control laws by fuzzy clustering. *Control Engineering Practice*, 14(7):769–781, 2006.

P. Poksawat, L. Wang, and A. Mohamed. Gain scheduled attitude control of fixed-wing uav with automatic controller tuning. *IEEE Transactions on Control Systems Technology*, 2017.

PX4. Parameter reference. `https://dev.px4.io/en/advanced/parameter_reference.html`, 2017. Accessed: 2017-09-17.

J. Reiner, G. J. Balas, and W. L. Garrard. Flight control design using robust dynamic inversion and time-scale separation. *Automatica*, 32(11):1493–1504, 1996.

S. Salazar, I. Gonzalez-Hernandez, R. Lopez, and R. Lozano. Simulation and robust trajectory-tracking for a quadrotor uav. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 1167–1174. IEEE, 2014.

P. J. Sanchez-Cuevas, G. Heredia, and A. Ollero. Multirotor uas for bridge inspection by contact using the ceiling effect. In *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 767–774, June 2017. doi: 10.1109/ICUAS.2017.7991412.

S. Sieberling, Q. Chu, J. Mulder, et al. Robust flight control using incremental nonlinear dynamic inversion and angular acceleration prediction. *Journal of guidance, control, and dynamics*, 33(6):1732, 2010.

P. V. M. Simplício. *Helicopter Nonlinear Flight Control*. Master's thesis, Instituto Superior Técnico - Universidade de Lisboa, October 2011.

J. J. E. Slotine and W. Li. *Applied nonlinear control*. Prentice hall, 1991. ISBN 0130408905.

E. J. Smeur, Q. Chu, and G. C. de Croon. Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles. *Journal of Guidance, Control, and Dynamics*, 2015.

P. Smith. A simplified approach to nonlinear dynamic inversion based flight control. In *AIAA Atmospheric Flight Mechanics Conference and Exhibit*, pages 762–770, 1998.

M. L. Steinberg and A. B. Page. Nonlinear adaptive flight control with a backstepping design approach. In *AIAA Guidance, Navigation and Exhibit*, pages 728–738, 1998.

D. Stilwell and W. J. Rugh. Interpolation methods for gain scheduling, 02 1998.

A. Strange. Autonomous medical drones will soon transport emergency organ donations. `http://mashable.com/2016/05/04/medical-drones/#1WZmCQpk4ZqM`, 2017. Accessed: 2017-09-30.

P. van Gils, E. van Kampen, C. C. de Visser, and Q. P. Chu. Adaptive incremental backstepping flight control for a high-performance aircraft with uncertainties. In *AIAA Guidance, Navigation, and Control Conference*, page 1380, 2016.

R. Verrier. Drones are providing film and tv viewers a new perspective on the action. `http://www.latimes.com/entertainment/envelope/cotown/la-et-ct-drones-hollywood-20151008-story.html`, 2017. Accessed: 2017-09-30.

J. Vincent. Amazon's vision for the future: delivery drone beehives in every city. `https://www.theverge.com/2017/6/23/15860668/amazon-drone-delivery-patent-city-centers`, 2017. Accessed: 2017-09-30.

H. Wang, Y. Zhang, Y. Yi, J. Xin, and D. Liu. Nonlinear tracking control methods applied to qball-x4 quadrotor uav against actuator faults. In *Control and Decision Conference (CCDC), 2016 Chinese*, pages 3478–3483. IEEE, 2016.

# Appendix A

# Extra information for quadrotor model and controller implementation

This appendix presents some extra information used during the quadrotor modeling process, as well as the implementation of the NDI and INDI controllers.

## A.1 Spyro Quadrotor Characteristics

Some physical characteristics, as well as operational capabilities, can be depicted in Table A.1.

Table A.1: UX-Spyro quadrotor characteristics. [source: https://www.uavision.com/]

| **Characteristic** | **Parameter Value** |
|---|---|
| Wingspread | 183 [$cm$] (including propellers) |
| Maximum Speed | 55 [$km/h$] |
| Service ceiling | 1000 [$m$] |
| Maximum Payload | 4 [$kg$] |
| Range | 8 [$km$] (in autonomous navigation) |
| Mass | 8.222 [$kg$] |
| Moment of Inertia $I_{xx}$ | 0.4688 [$kg\,m^2$] |
| Moment of Inertia $I_{yy}$ | 0.7639 [$kg\,m^2$] |
| Moment of Inertia $I_{zz}$ | 0.4930 [$kg\,m^2$] |

## A.2 Derivative with respect to a rotating frame of reference

In accordance with section 15.10 of [Johnston et al., 2010] the rate of change of a vector $\mathbf{V}$ with respect to a fixed frame $XYZ$ can be written relative to a rotating frame $xyz$ as in Equation (A.1), where the rotating frame has an angular velocity $\omega$ with respect to the fixed frame.

$$\dot{\mathbf{V}}_{XYZ} = \dot{\mathbf{V}}_{xyz} + \omega \times \mathbf{V} \tag{A.1}$$

In the quadrotor case the rotating frame is the body frame and the fixed frame represents the inertial NED frame. Newton's second law is written relative to the inertial frame as in Equation (A.2). Writing it with respect to the rotating body frame is straightforward, obtaining expression (A.3) because mass is a constant, and where $\mathbf{b}$ subscript indicates that the quantity is written with respect to the rotating frame.

$$\mathbf{F} = \frac{\mathrm{d}}{\mathrm{d}t}(m\mathbf{V}) \tag{A.2}$$

$$\mathbf{F_b} = m\left[\frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{V_b}) + \omega_\mathbf{b} \times \mathbf{V_b}\right] = m\dot{\mathbf{V}}_\mathbf{b} + m\left(\omega_\mathbf{b} \times \mathbf{V_b}\right) \tag{A.3}$$

The same reasoning can be applied to the rotational dynamics equation with the moments acting on the quadrotor. Similarly to the forces this equation is written with respect to the inertial frame as in Equation (A.4).

$$\mathbf{M} = \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{I}\omega) \tag{A.4}$$

This formulation must be rewritten with respect to the body frame because it is the only frame of the two where the inertia matrix $\mathbf{I}$ is a constant. Using again transformation (A.1) it is possible to obtain Equation (A.5), where once again $b$ subscript indicates that the quantity is written with respect to the body frame.

$$\mathbf{M_b} = \mathbf{I_b}\left[\frac{\mathrm{d}}{\mathrm{d}t}(\omega_\mathbf{b}) + \omega_\mathbf{b} \times \omega_\mathbf{b}\right] = \mathbf{I_b}\,\dot{\omega}_\mathbf{b} + \omega_\mathbf{b} \times \mathbf{I_b}\,\omega_\mathbf{b} \tag{A.5}$$

## A.3 Selection of parameters for linear controllers

A MATLAB/Simulink model of a second order system was created to help in the selection of the parameters for roll and pitch linear controllers (Equations (4.16)).

The influence of the damping ratio can be depicted in Figure A.1, where the time response for a unit step with unit natural frequency ($\omega_n = 1\ [rad/s]$) and varying damping ratios between 0.1 and 1, is shown.

Analyzing the previous Figure it is possible to conclude on one hand that a bigger damping ratio decreases the oscillations around the final value, as well as the maximum overshoot. The latter does not exist for $\xi > \frac{1}{\sqrt{2}} \approx 0.707$, as one can see in the graph, with a formal proof in chapter 5 of [Ogata, 2010]. On the other hand a bigger damping ratio also increases the rising time and, for $\xi > 0.707$, the settling times are also bigger. In conclusion higher damping ratio values are preferable, since avoiding overshoot and oscillatory behaviours are more important objectives than a small rising time. For this roll and pitch controllers $\xi = 0.9$ was chosen. A detailed explanation of the parameters used to describe the time response of systems, such as the rising time, settling time, maximum overshoot, peak time and delay time, can also be found in chapter 5 of [Ogata, 2010].

Choosing a value for the natural frequency is not as intuitive as choosing the damping ratio, and

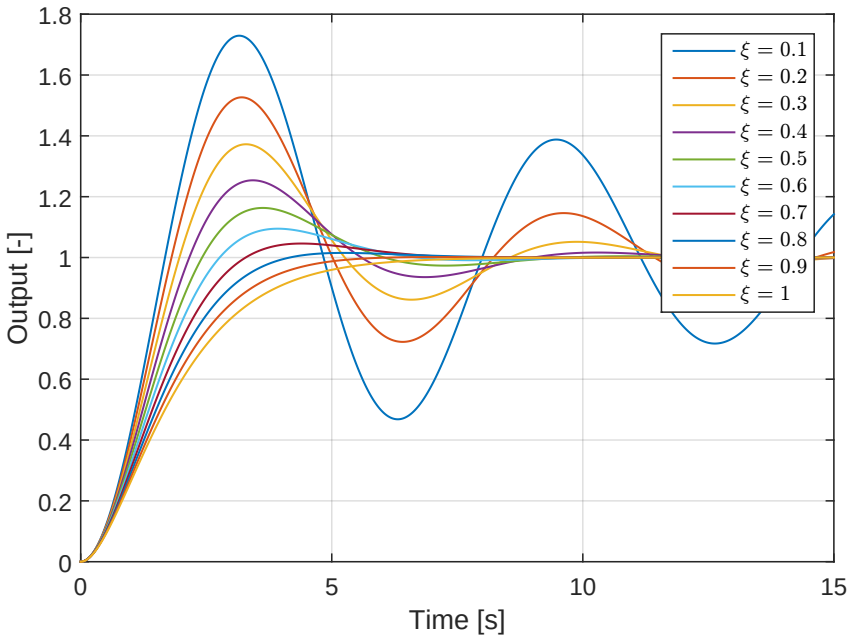


Figure A.1: Time response of second order system to unit step.

therefore one can choose a value to a more meaningful parameter and then convert it to obtain the desired natural frequency. In this case the most adequate parameter is the settling time ($t_s$), which describes the time that the output takes to be fully enclosed around the final value. For example, for a unit step, a settling time at $2\%$ is the time that the output of the system takes to be fully enclosed in the interval $[0.98,\ 1.02]$. The expression for the settling time at $2\%$, taken from [Ogata, 2010], can be depicted in Equation (A.6). Similarly to the damping, a graph with the time response to a unit step, damping ratio $\xi = 0.9$ and varying settling times at $2\%$ between 0.3 and 3 is shown in Figure A.2.

$$t_{s\ [2\%]} = \frac{4}{\xi \omega_n} \tag{A.6}$$

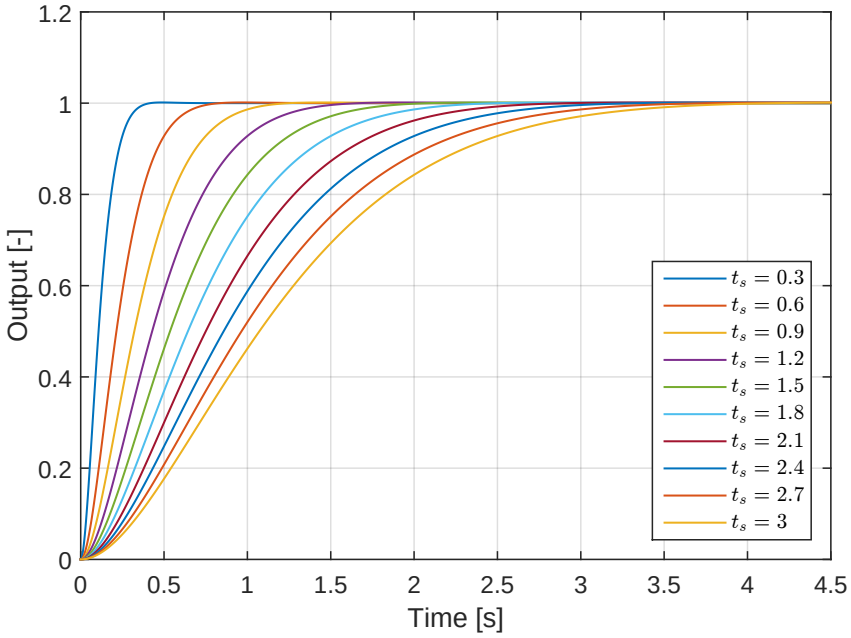


Figure A.2: Time response of second order system to unit step.

Observing the previous Figure one must conclude that the smallest sampling time is the better, given that the remaining properties, as overshoot and oscillations, are kept constant. In reality considerably small settling times will cause an oscillatory response, due to the existence of noise and disturbances. For this reason an initial value of $t_s = 1\ [s]$ was selected and, if needed in future simulations and experimental tests, this value can be modified.

The same approach was used for the gain of the first order system present in yaw rate and vertical velocity controllers. The time response of a first order system to a unit step, with varying gains since 0.1 until 10 is shown in Figure A.3.

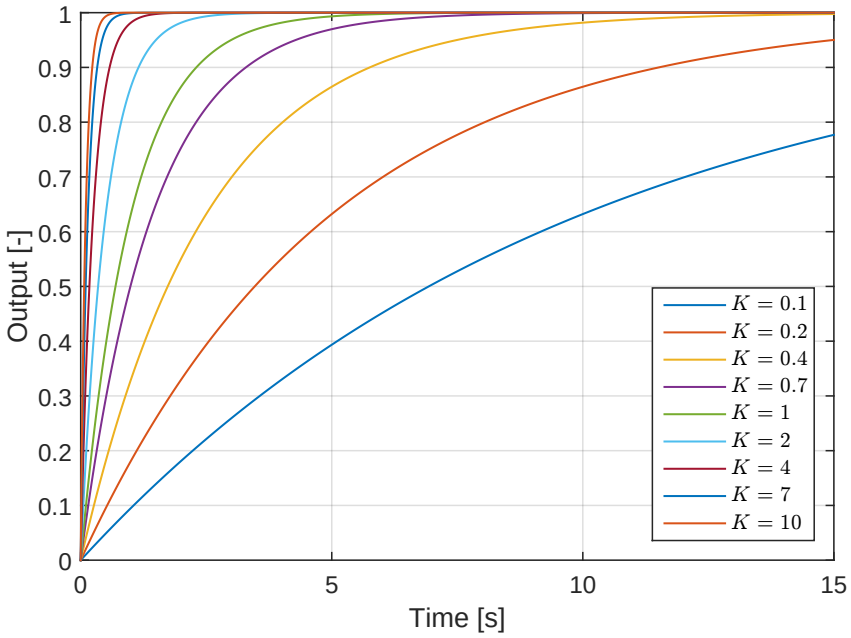


Figure A.3: Time response of first order system to unit step.

Observing the previous Figure one can say that a higher gain leads to a faster, thus preferable, response. In first order systems the time constant $\tau$ (time that takes to reach $1 - \frac{1}{e} \approx 63.21\%$ of the final value) is related with the gain $K$ by the following expression: $\tau = \frac{1}{K}$. For example, for $K = 1$ it is possible to observe in the graph that the time constant is around $1\ [s]$. Using the time constant as the choice parameter instead of the gain seems a more intuitive approach, given that it has a meaning in the time response. Equivalently to the second order filters, when noise and disturbances are introduced a higher gain could lead to instabilities and oscillatory behaviour. Hence, a first choice for the time constants of both yaw rate and vertical velocity controllers will be $\tau = 0.5\ [s]$, which gives $K_{\dot{\psi}} = K_{v_z} = 2$.

In section 5.4 one needed to select an integral gain to control a first order system. Thus, a MATLAB/Simulink time response of a PI controller was built, with fixed proportional gain $K_p = 2\ [s^{-1}]$ (corresponding to the chosen value in chapter 4) and variable integrative gain. The results can be depicted in Figure A.4. Similarly to Figure A.3 with the varying proportional gain, a higher integrative gain results in a lower rising time, but a more oscillatory response with higher settling time. Taking into account that the objective is to minimize the oscillations and the settling time of the response a gain $K_i = 0.2\ [s^{-2}]$ was selected.

$$\dot{v}_z = K_{v_z}\left(v_{z_d} - v_z\right) + K_i \int v_{z_d} - v_z\ dt \tag{A.7}$$

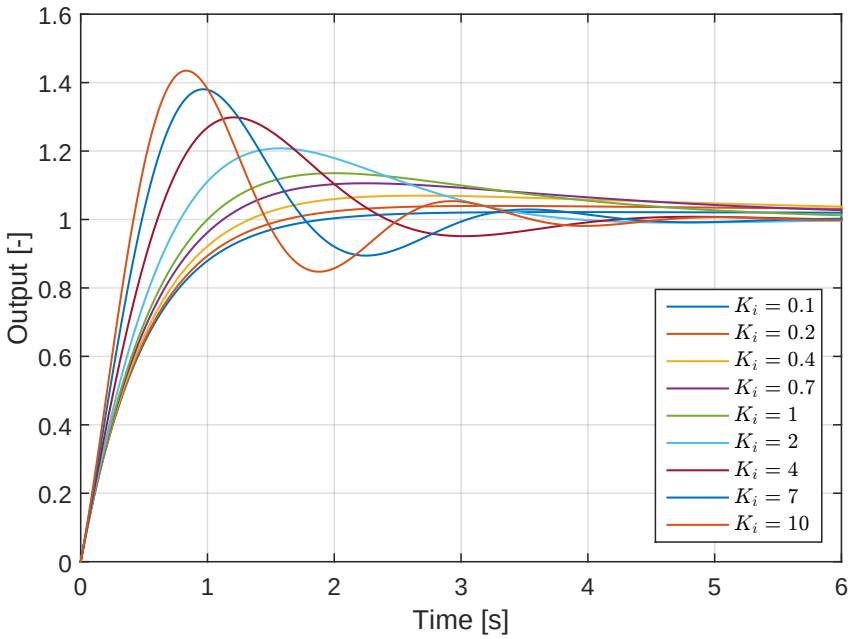


Figure A.4: Time response of first order system with PI control to unit step.

# Appendix B

# Simulation Block Diagrams

This appendix shows the block diagrams used in MATLAB/Simulink simulation environment.

## B.1 NDI Block Diagram

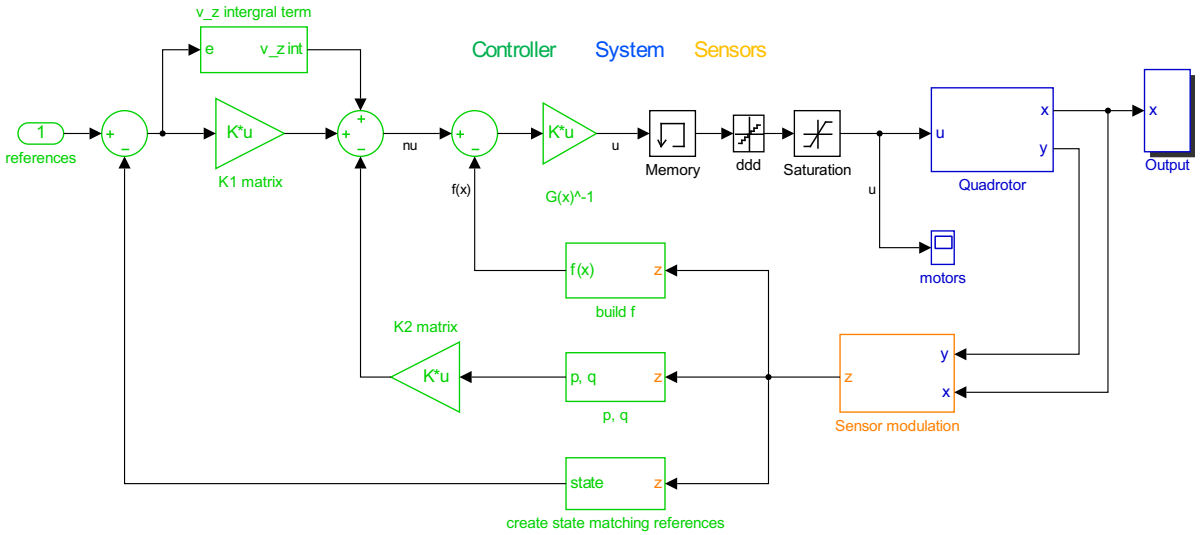



Figure B.1: MATLAB/Simulink block diagram for NDI controller.

The control diagram is very similar to the one designed in implementation (Figure 4.3), adding only a *Memory* block in order to avoid algebraic loops during simulation. Furthermore, it also has a discretization (*ddd*) block that emulates the discrete nature of the input to the motors and a *saturation* between $0$ and $1$ that guarantees an input to the motors in that range. The last addition consists in the *sensor modulation* block, described in section 5.2.

## B.2 INDI Block Diagram

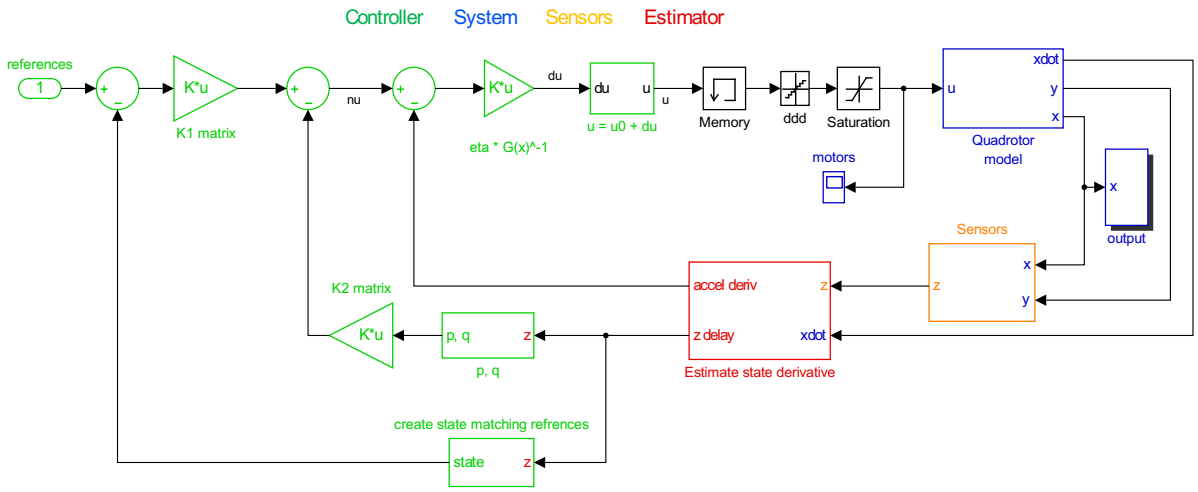



Figure B.2: MATLAB/Simulink block diagram for INDI controller.

Besides the additions already mentioned for the NDI block diagram this INDI diagram has a block that sums the incremental control input $\mathbf{u_0}$ given by the controller to the last iteration control input $\mathbf{u_0}$. This task was implemented using an *S-Function* Simulink block. It also presents a block diagram that estimates the state derivative, as described in section 5.3.

## B.3 Position + INDI controllers Block Diagram

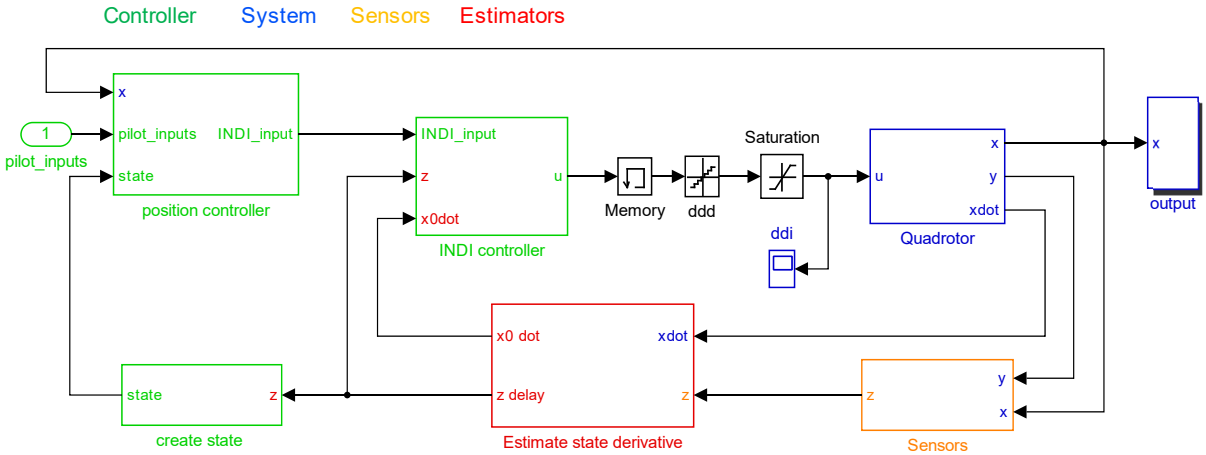



Figure B.3: MATLAB/Simulink block diagram for position and INDI controller.

The final block diagram used in MATLAB/Simulink simulation environment combines the INDI controller from Figure B.2 and the position controller described in section 4.4.

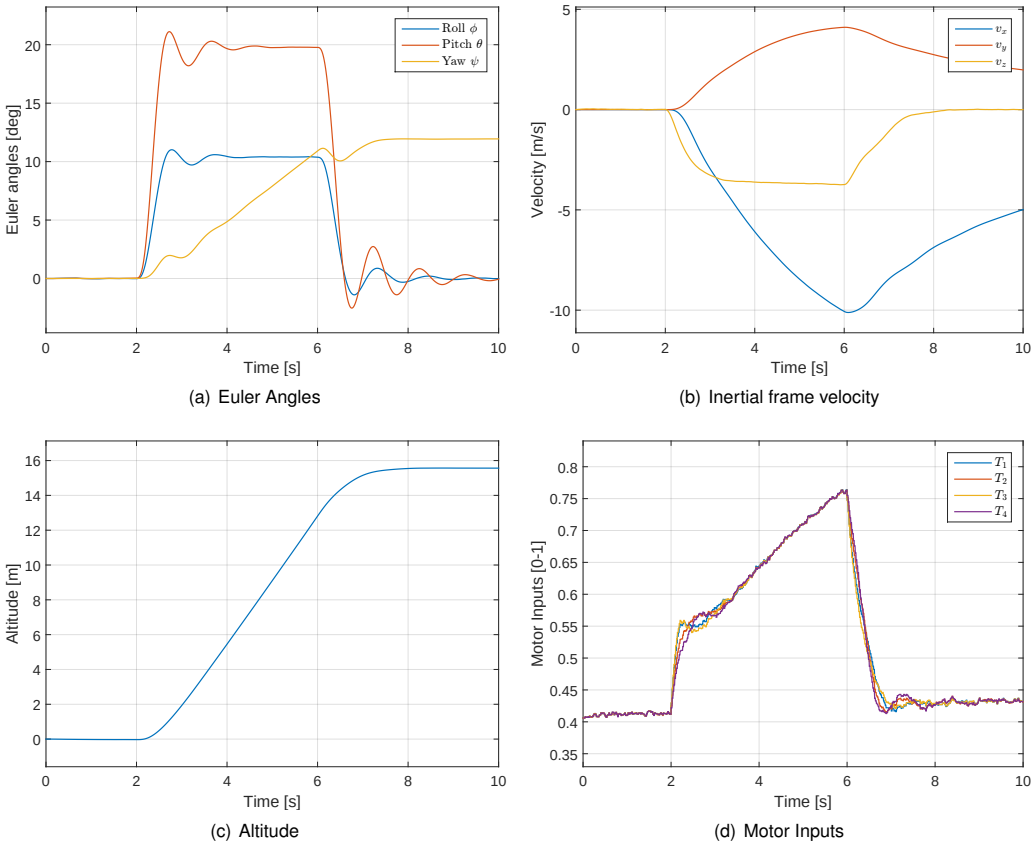
# Appendix C

# Parametric simulation results

The complete simulation results of the parametric analyses are shown in this appendix.

## C.1 Selection of $\eta$ parameter

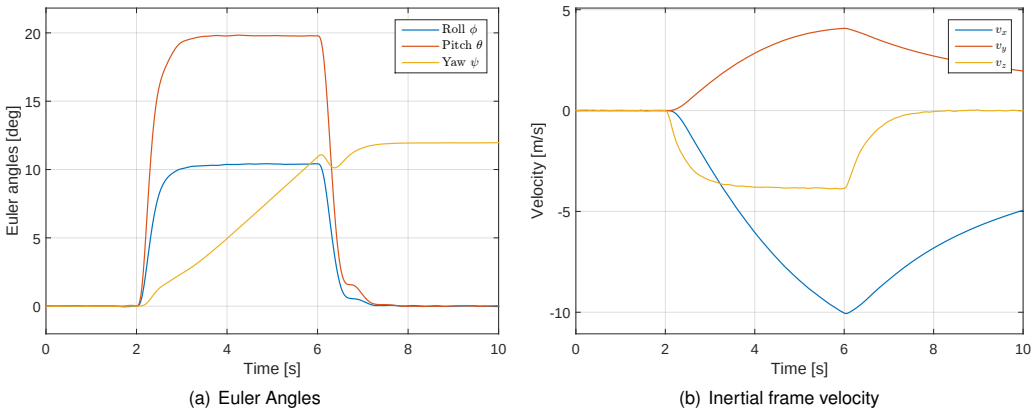
For the $\eta$ parameter that scales the incremental input action simulations results are shown for $\eta = 0.1$, $\eta = 0.2$, $\eta = 0.3$, $\eta = 0.4$, $\eta = 0.5$, $\eta = 0.6$ and $\eta = 1$. The sampling time is 0.5 for all simulations.



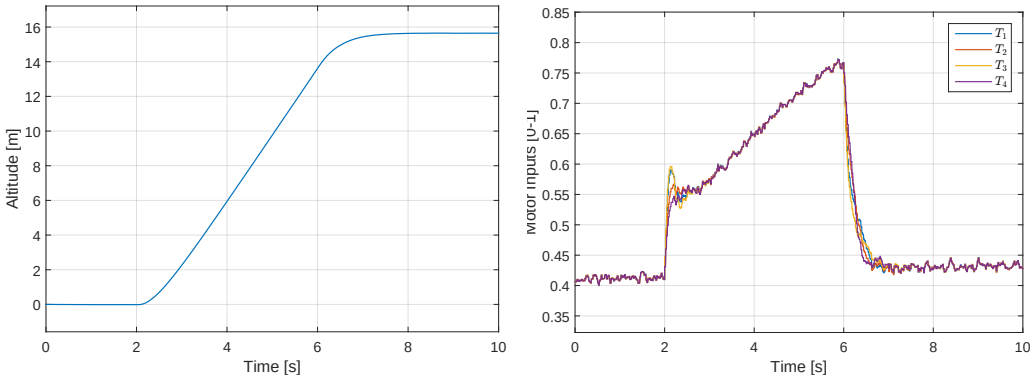

(a) Euler Angles (b) Inertial frame velocity

(c) Altitude (d) Motor Inputs

Figure C.1: Type 2 simulation results for INDI controller with $\eta = 0.1$.

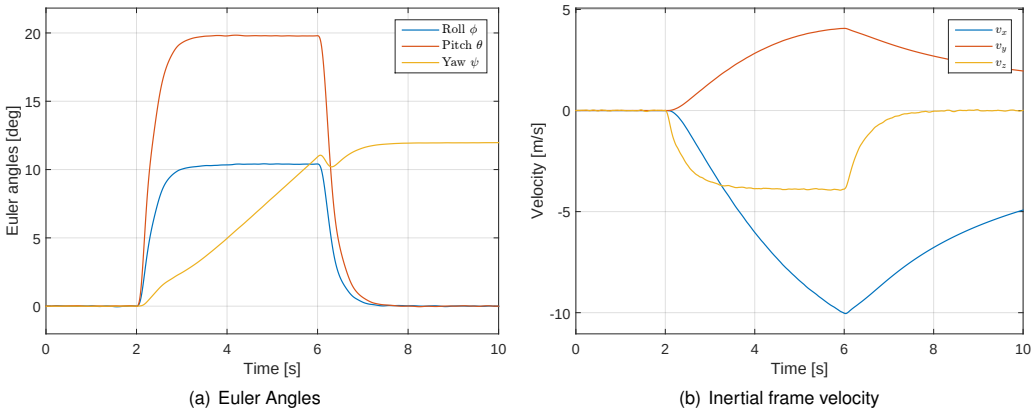



(a) Euler Angles (b) Inertial frame velocity

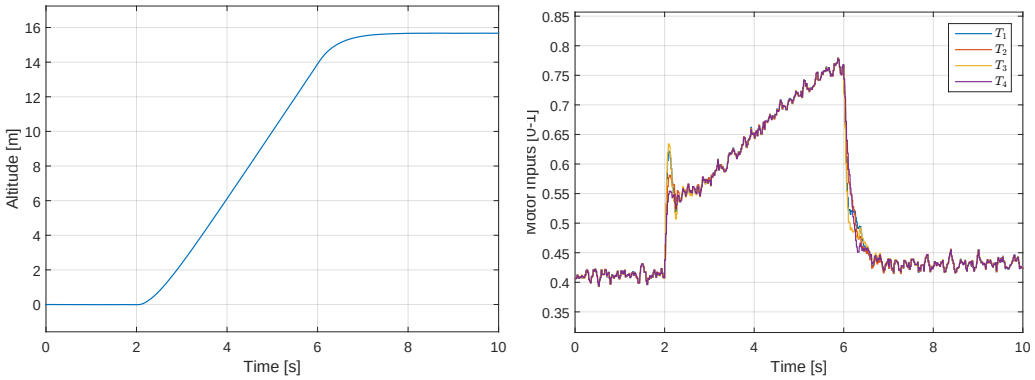



(c) Altitude (d) Motor Inputs

Figure C.2: Type 2 simulation results for INDI controller with $\eta = 0.2$.

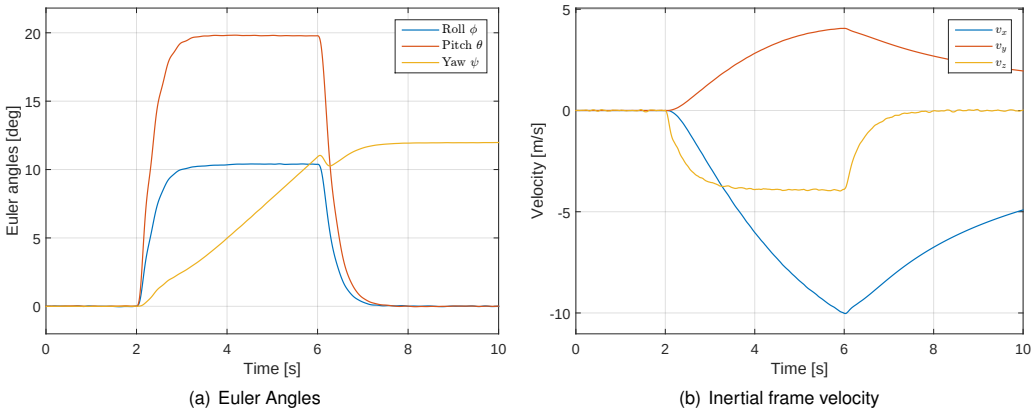



(a) Euler Angles (b) Inertial frame velocity

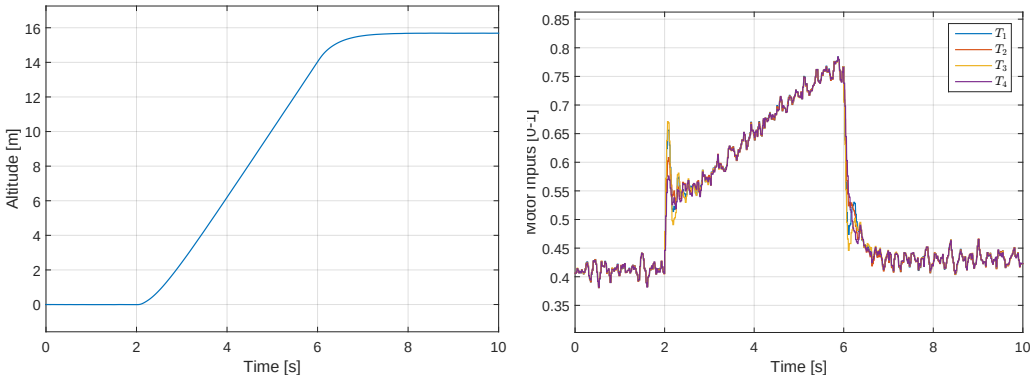



(c) Altitude (d) Motor Inputs

Figure C.3: Type 2 simulation results for INDI controller with $\eta = 0.3$.

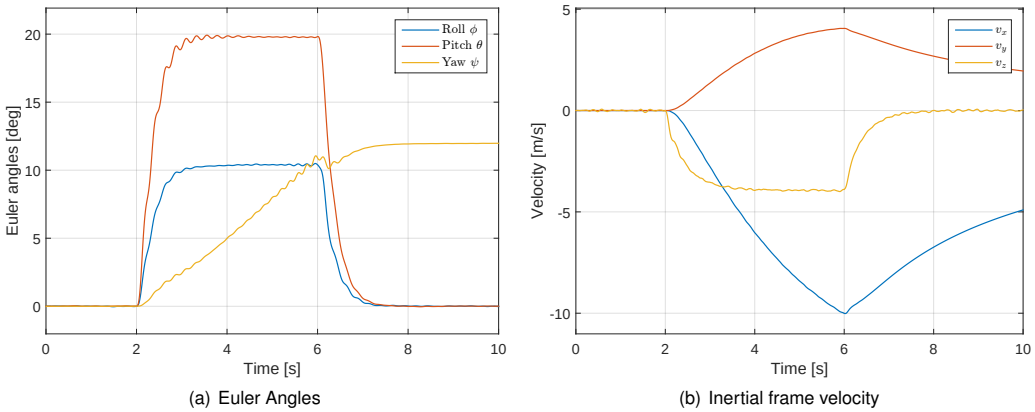



(a) Euler Angles (b) Inertial frame velocity

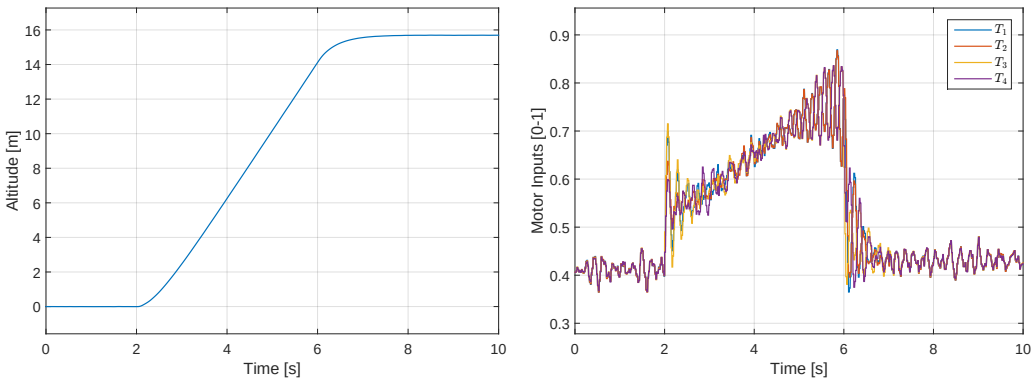



(c) Altitude (d) Motor Inputs

Figure C.4: Type 2 simulation results for INDI controller with $\eta = 0.4$.

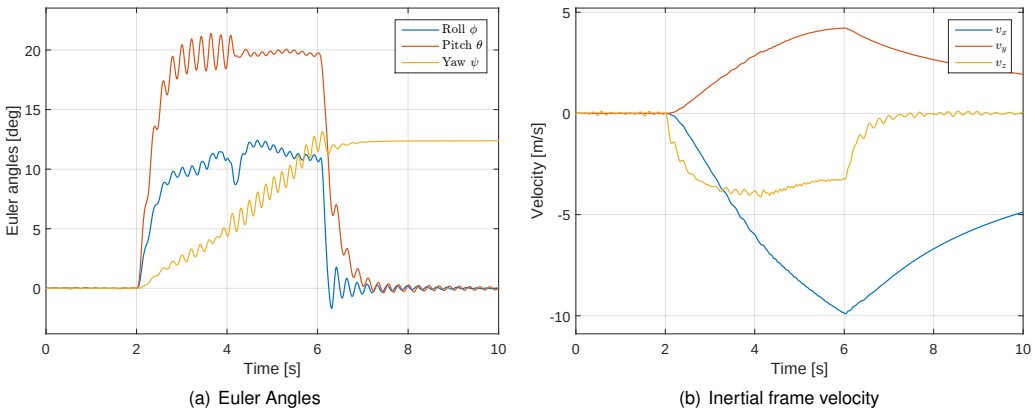



(a) Euler Angles (b) Inertial frame velocity

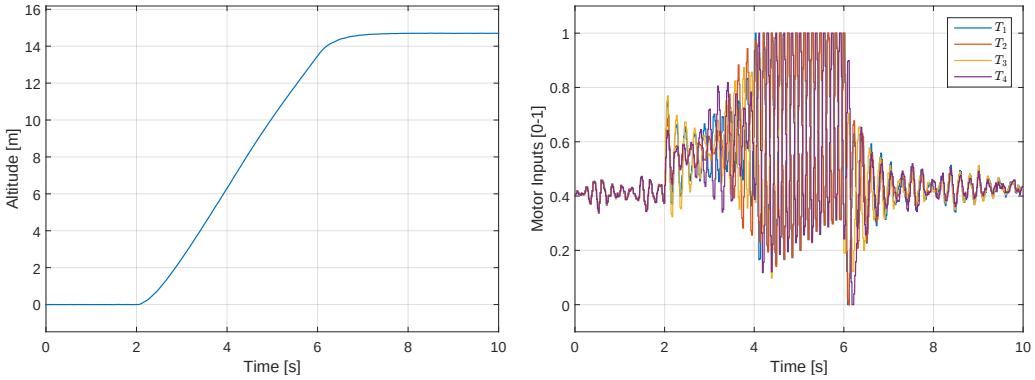



(c) Altitude (d) Motor Inputs

Figure C.5: Type 2 simulation results for INDI controller with $\eta = 0.5$.

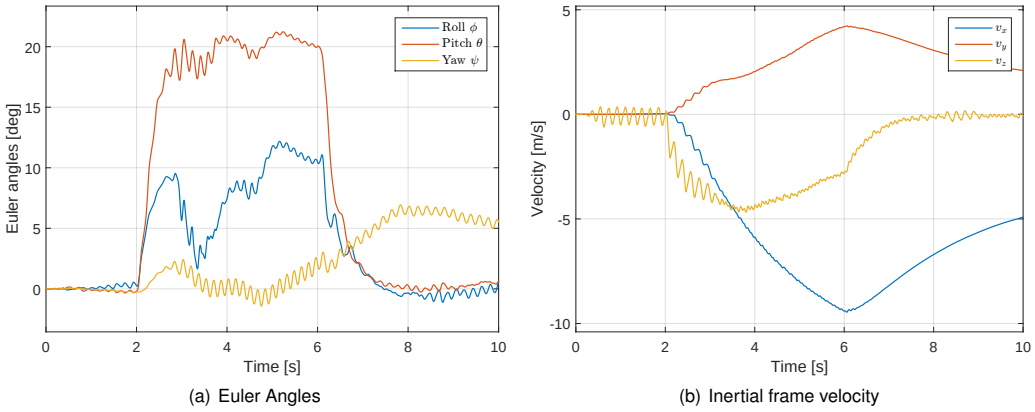



(a) Euler Angles (b) Inertial frame velocity

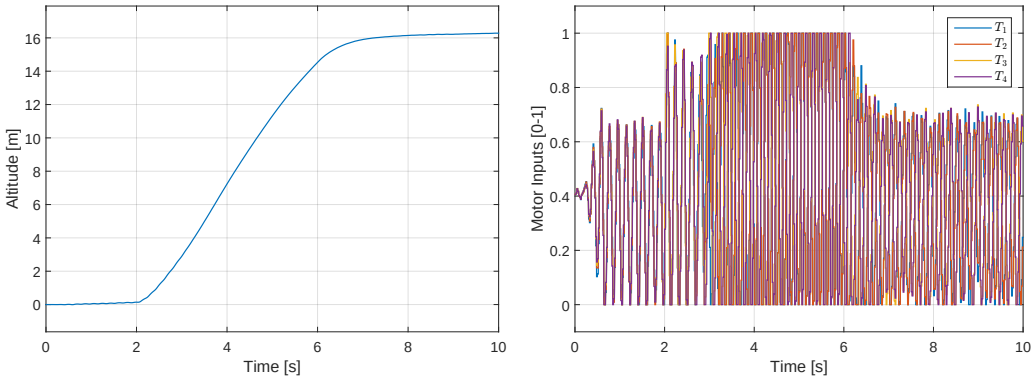



(c) Altitude (d) Motor Inputs

Figure C.6: Type 2 simulation results for INDI controller with $\eta = 0.6$.




(a) Euler Angles (b) Inertial frame velocity

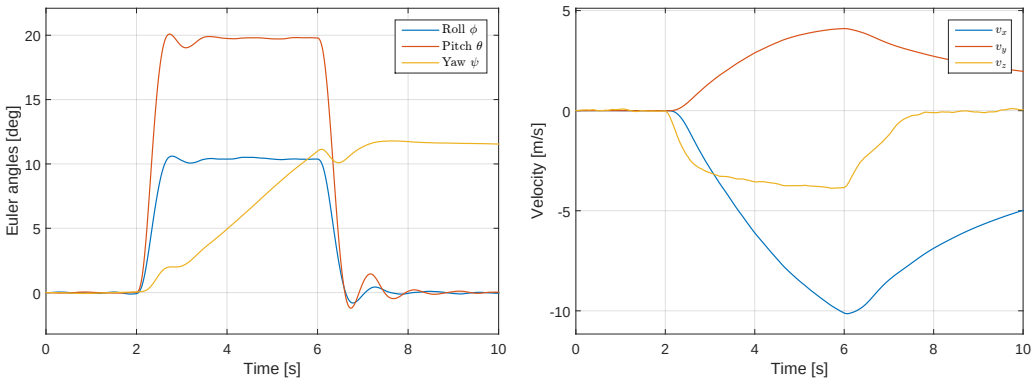



(c) Altitude (d) Motor Inputs

Figure C.7: Type 2 simulation results for INDI controller with $\eta = 1$.

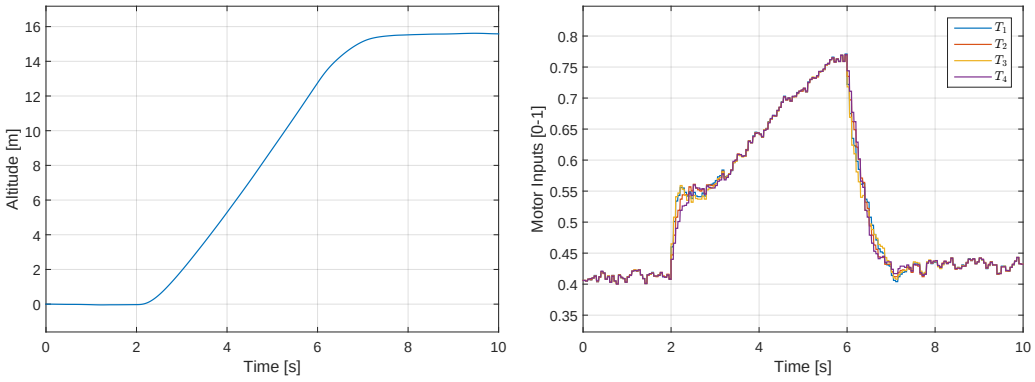
## C.2 Selection of sampling time

Simulation results for sampling times $t_s = 1/20\ [s]$, $t_s = 1/40\ [s]$, $t_s = 1/50\ [s]$, $t_s = 1/60\ [s]$, $t_s = 1/80\ [s]$, $t_s = 1/100\ [s]$ and $t_s = 1/120\ [s]$ can be depicted in the following Figures. The $\eta$ parameter is 0.3 for all simulations.




(a) Euler Angles

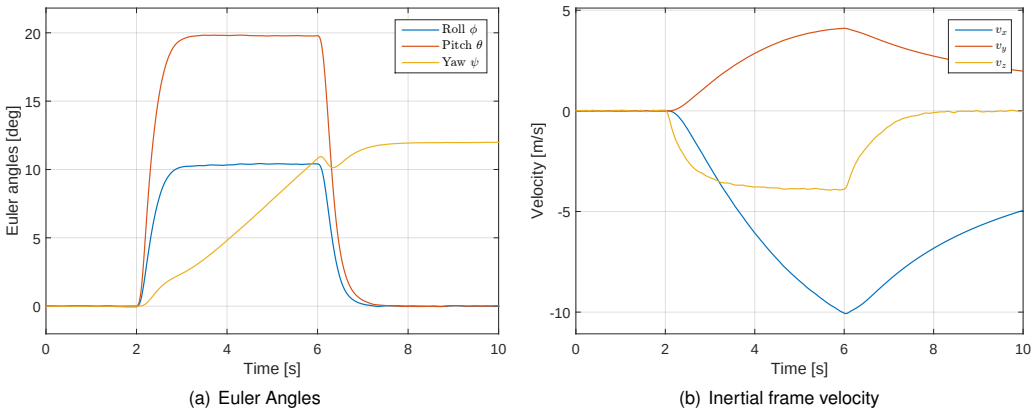
(b) Inertial frame velocity



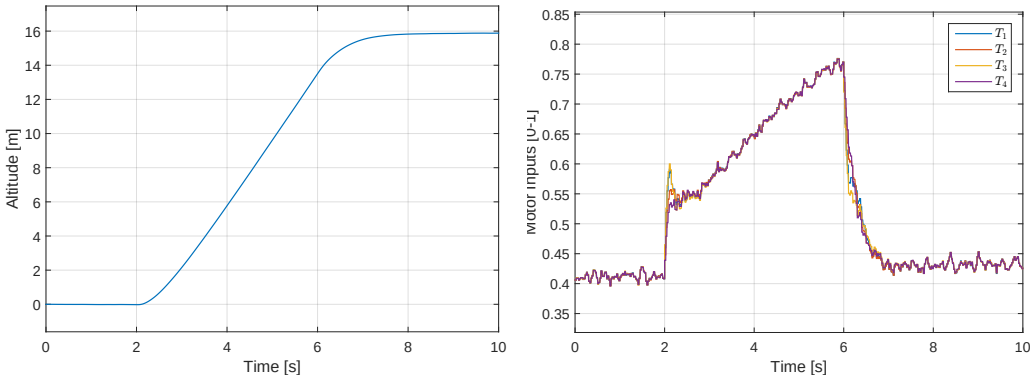

(c) Altitude

(d) Motor Inputs

Figure C.8: Type 2 simulation results for INDI controller with $t_s = 1/20\ [s]$.

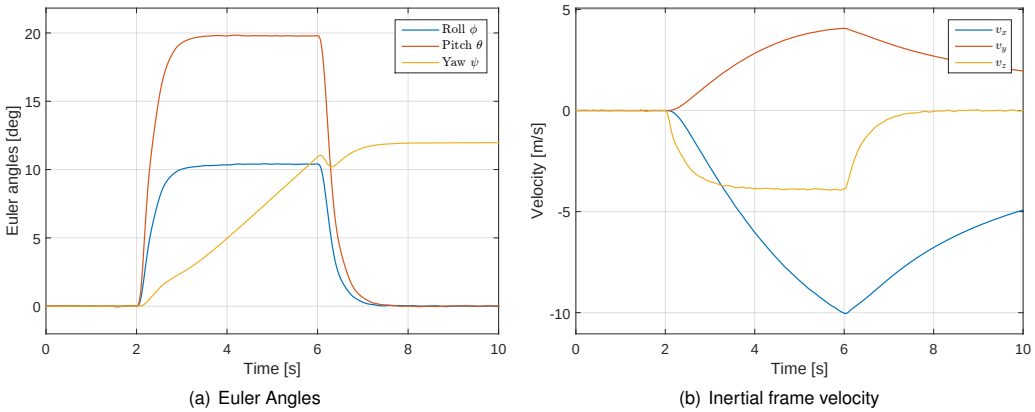



(a) Euler Angles (b) Inertial frame velocity

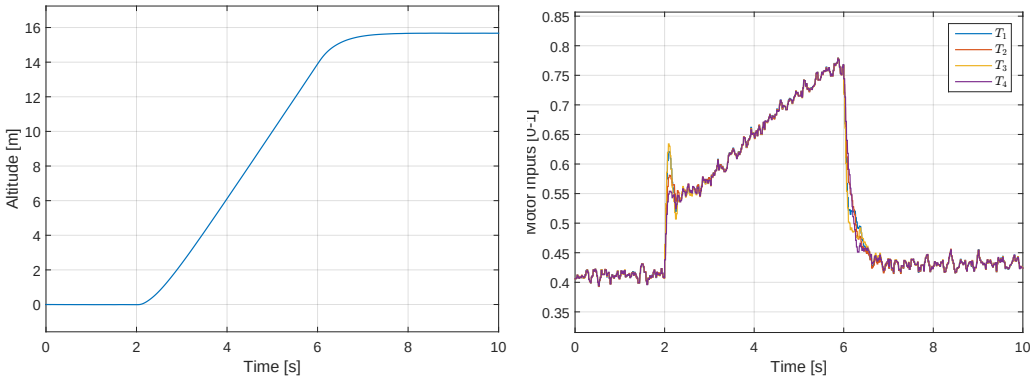



(c) Altitude (d) Motor Inputs

Figure C.9: Type 2 simulation results for INDI controller with $t_s = 1/40\ [s]$.

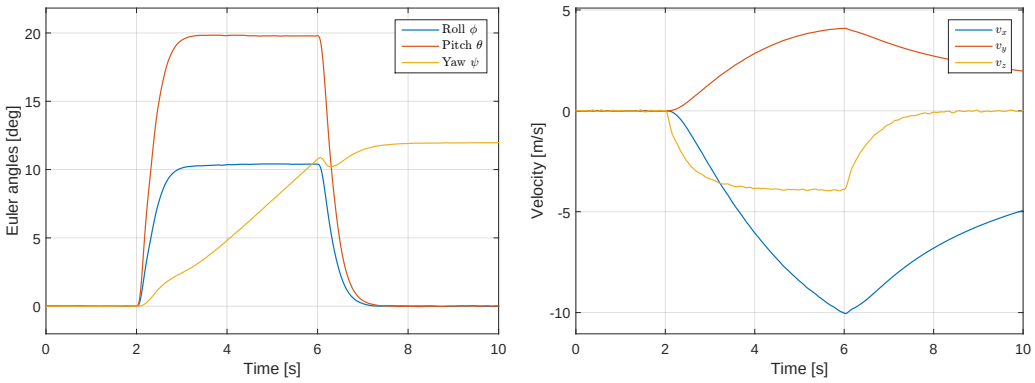



(a) Euler Angles (b) Inertial frame velocity

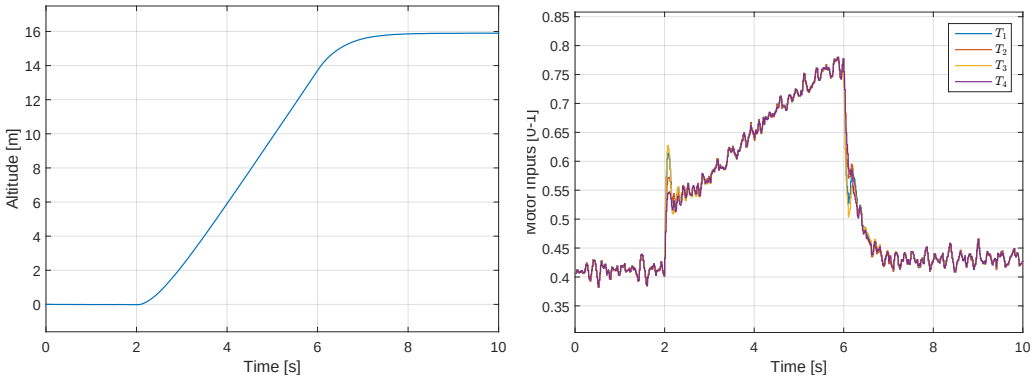



(c) Altitude (d) Motor Inputs

Figure C.10: Type 2 simulation results for INDI controller with $t_s = 1/50\ [s]$.

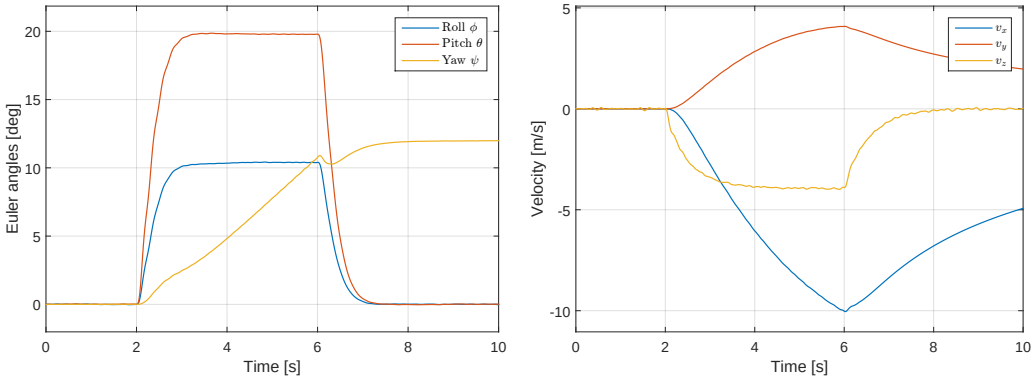



(a) Euler Angles (b) Inertial frame velocity

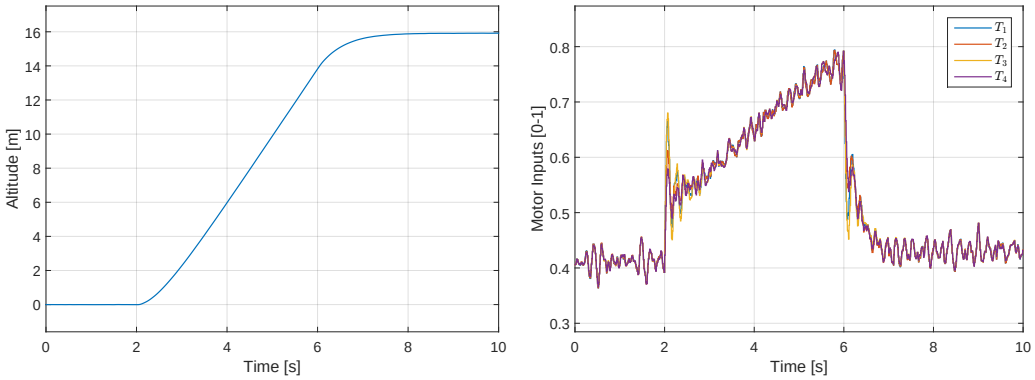



(c) Altitude (d) Motor Inputs

Figure C.11: Type 2 simulation results for INDI controller with $t_s = 1/60\ [s]$.

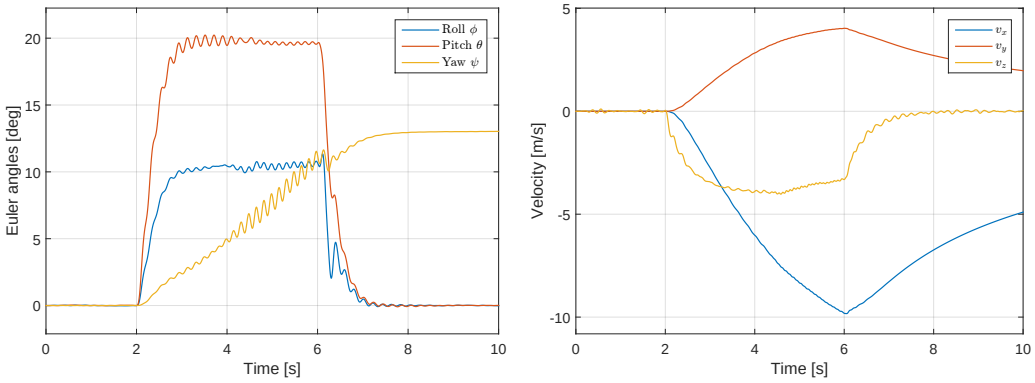



(a) Euler Angles (b) Inertial frame velocity

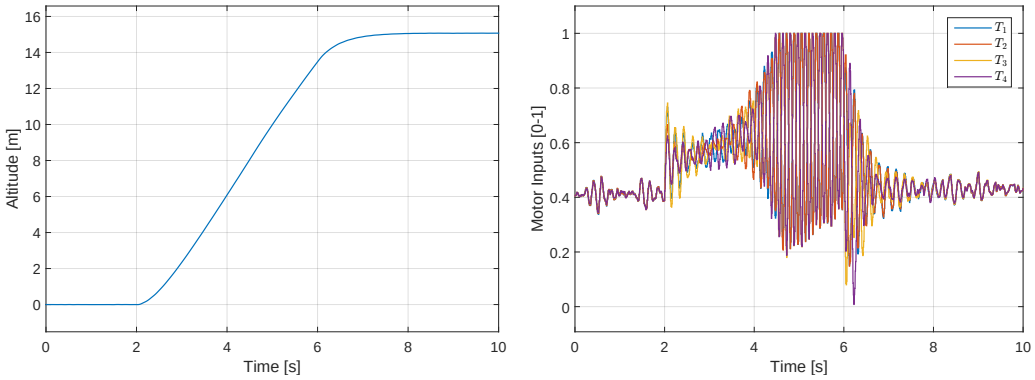



(c) Altitude (d) Motor Inputs

Figure C.12: Type 2 simulation results for INDI controller with $t_s = 1/80\ [s]$.

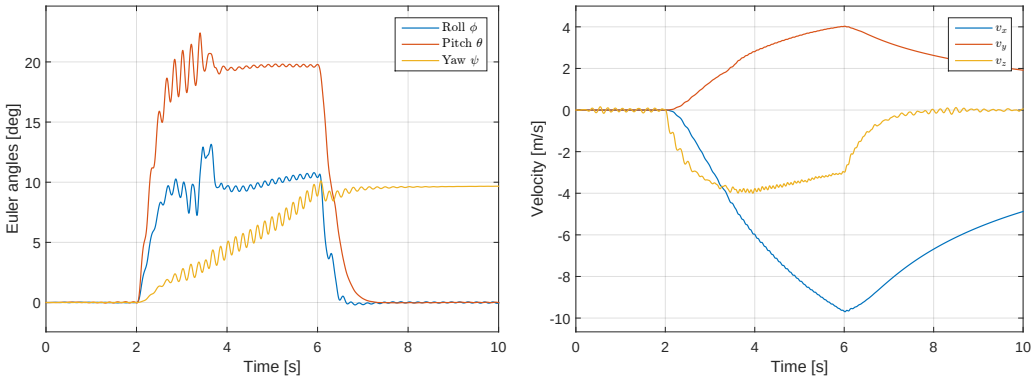



(a) Euler Angles (b) Inertial frame velocity

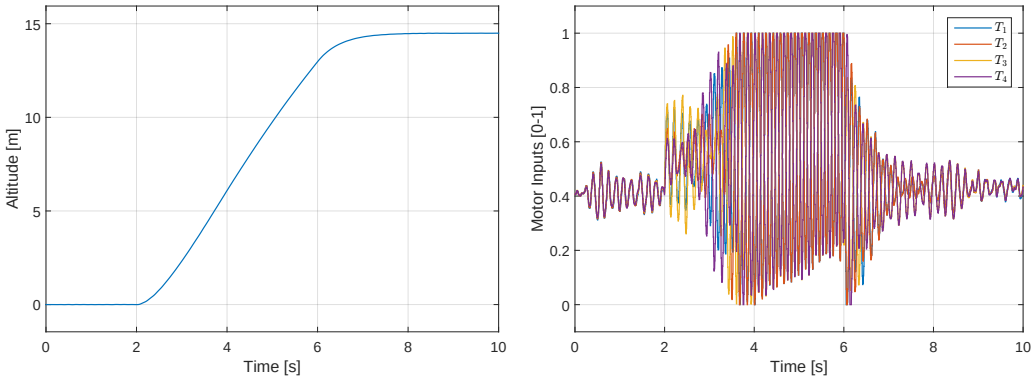



(c) Altitude (d) Motor Inputs

Figure C.13: Type 2 simulation results for INDI controller with $t_s = 1/100\ [s]$.




(a) Euler Angles (b) Inertial frame velocity




(c) Altitude (d) Motor Inputs

Figure C.14: Type 2 simulation results for INDI controller with $t_s = 1/120\ [s]$.

# Appendix D

# Technical Datasheets

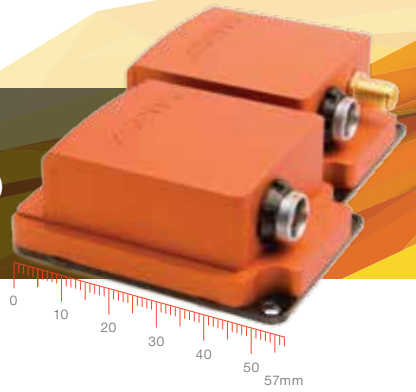
This appendix contains all the technical datasheets where information was collected.

## D.1 Xsens Datasheet

# MTi 100-series

**The most accurate and complete MEMS based IMU, VRU, AHRS and GNSS/INS**

- ✔ Best performing MEMS based sensor on the market
- ✔ High performance vibration resistant MEMS based gyroscopes (10 deg/h)
- ✔ Meets DO-160 specifications, integrated in FAA-certified applications




## MTi 100-series

- Breakthrough tracking performance
- Coning and sculling algorithms @ 2kHz
- Motion processing core for multiple sensor inputs and data sources
- High-performance XEE, beyond traditional Kalman Filtering
- Tuned for performance under vibrations and magnetic distortions
- Comprehensive SDK and straightforward system integration

## Breakthrough performance from market leader

- Next level MEMS AHRS with vibration rejecting gyroscopes
- Cutting-edge sensor fusion technology
- Market leader serving a large and high-profile customer base

## Complete product, with highest accuracy

- Low latency (<2ms) for real-time applications
- Compensation against long-lasting transient accelerations
- Able to cope with GNSS outages and magnetic distortions
- Leading innovator introducing a new class of AHRS's

## Maximum flexibility and versatility

- Available as OEM board and IP67 encased MTi
- 24-pins connector for OEM
- Extensive suite of output formats, available directly from the MTi
- Choice of several interfaces, onboard USB and GPIO's
- Xsens' industry standard open Xbus protocol or NMEA
- All products from the MTi-series are fully interchangeable

## Product overview

| | | 100-IMU | 200-VRU | 300-AHRS | 710-GNSS/INS |
|---|---|---|---|---|---|
| | | | Typ \| Max | Typ \| Max | Typ \| Max |
| **Calibrated Sensor Data** | | yes | yes | yes | yes |
| Roll/pitch | Static | - | 0.2° \| 0.25° | 0.2° \| 0.25° | 0.2° \| 0.25° |
| | Dynamic | - | 0.3° \| 1.0° | 0.3° \| 1.0° | 0.3° \| 1.0° |
| Yaw | In homogenous magnetic field | - | Active Heading Stabilization | 1.0° | 1.0° |
| **Position and velocity** | | | | | |
| Horizontal position | 1σ STD (SBAS) | - | - | - | 1.0 m |
| Vertical position | 1σ STD (SBAS, baro) | - | - | - | 2.0 m |
| Velocity | 1σ RMS | - | - | - | 0.05 m/s |

## Sensor specification

| | Gyroscopes | | Accelerometers | |
|---|---|---|---|---|
| | Typ | Max | Typ | Max |
| Standard full range | +/- 450°/s | - | 50 m/s² | - |
| Bias repeatability (1 yr) | 0.2°/s | 0.5°/s | 0.03 m/s² | 0.05 m/s² |
| In-run bias stability | 10°/h | - | 40 µg | - |
| Bandwidth (-3 dB) | 415 Hz | N/A | 375 Hz | N/A |
| Noise density | 0.01°/s/√Hz | 0.015°/s/√Hz | 80 µg/√Hz | 150 µg/√Hz |
| g-sensitivity (calibrated) | 0.003°/s/g | 0.015°/s/g | N/A | N/A |
| Non-orthogonality | 0.05 deg | - | 0.05 deg | - |
| Non-linearity | 0.01% FS | - | 0.03% FS | 0.5% FS |

| | Magnetometer | | Barometer | |
|---|---|---|---|---|
| | Typ | Max | Typ | Max |
| Standard full range | - | +/- 80 µT | - | 300-1100 hPa |
| Noise density | 200 µG/√Hz | - | 0.01 hPa/√Hz | - |
| Non-linearity | 0.1% FS | - | - | - |

### GNSS receiver (MTi-G-710 GNSS/INS only)

| | | | |
|---|---|---|---|
| Receiver type | 72ch, GPS L1 C/A code, GLONASS L10F | Horizontal accuracy (CEP) | 2.0 (2.5 m w/o SBAS) |
| Update rate | 4 Hz | Vertical accuracy (CEP) | 5 m |
| Start-up time cold start | 26 s | Velocity accuracy (@30 m/s) | 0.05 m/s |
| Tracking sensitivity | -164 dBm | DGPS | SBAS |

## System specifications

| | | | |
|---|---|---|---|
| Input voltage | 4.5 to 34V or 3V3; | Clock drift | 10 ppm (1 ppm w. GNSS) or ext. ref. |
| Typical power consumption | 675-950 mW @5V | Output frequency | Up to 2 kHz |
| Start-up time | 2.5 sec. | Latency | <2 ms |
| IP-rating | IP 67 (encased) | Interfaces | RS232/422/485/UART/USB (on board) |
| Temperature (in use) | -40 to 85 °C | GPIO's and options | SyncIn, SyncOut, 2x GPIO, Clock sync |
| Vibration and shock | MIL STD 202 / 2000g | Interface protocol | XBus or NMEA |
| Casing material | Anodized aluminum 6060 | Mounting | Free; orientation alignment available |
| Sampling frequency | 10 kHz/channel (60 kS/s) | Built-in self test (BIT) | gyroscopes, accelerometers, magnetometer |

MTi-G encased:
57x42x23.5 mm, 55g,
9-pins push-pull connector

MTi encased:
57x42x23.5 mm, 52g,
9-pins push-pull connector

OEM:
37x33x12 mm, 11g,
24-pins header