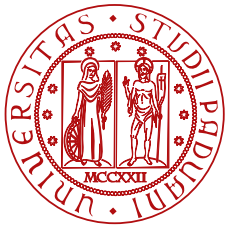


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Control Systems Engineering

A Learning-Based Framework for Quadrotor Modeling and Control

Master Candidate

Mattia Scarpa

Student ID 2005826

Supervisor

Dott. Alberto Dalla Libera

University of Padova

Academic Year
2022/2023

In memory of my grandparents,
Ada, Giuseppe, Mirella, and Alberto,
to whom I dedicate this work!

Abstract

This work focuses on the study of controlling complex systems, such as quadrotors, through machine learning techniques. Specifically, it explores the implementation of the Model-Based Reinforcement Learning (MBRL) algorithm known as Monte Carlo Probabilistic Inference for Learning and COntrol (MC-PILCO), an extension of the PILCO algorithm, which leverages Gaussian Process Regression (GPR) to accurately learn the dynamics of the quadrotor and investigates its potential applications in flight control.

The context of the research concerns the need to develop effective and flexible control methods for quadrotors, which must confront complex dynamics, model uncertainties, and operating environment uncertainties. In this respect, MC-PILCO, which relies on GPR, emerges as a promising technique for learning the system's dynamics.

The research involved studying the equations of quadrotor dynamics and using them to simulate the learning environment. The GPR model, used within the MC-PILCO algorithm, was trained using an RBF kernel and a speed integration model. Furthermore, the use of the Subset of Data method to approximate the GPR model was investigated, analyzing the effect of reducing the number of data points on prediction accuracy.

Experimental results in simulated environments show that the GPR model improves with an increase in data and that the Subset of Data method provides acceptable results within a certain degree of approximation. The effectiveness of MC-PILCO was demonstrated by comparing its control results with those of other existing traditional controllers, showing substantial improvements.

This demonstrates that the proposed MC-PILCO approach, which incorporates GPR, represents a valid alternative to the currently available simulation and control methods for quadrotors, paving the way for future research in this area.

Sommario

Questo elaborato si concentra sullo studio del controllo di sistemi complessi, come i quadrotor, attraverso l'uso di tecniche di apprendimento automatico. In particolare, viene esplorata l'implementazione dell'algoritmo di Model-Based Reinforcement Learning (MBRL) chiamato Monte Carlo Probabilistic Inference for Learning and COntrol (MC-PILCO), un'estensione dell'algoritmo PILCO, che sfrutta la Regressione del Processo Gaussiano (GPR) per apprendere accuratamente la dinamica del quadrotor e indagare le sue potenziali applicazioni nel controllo del volo.

Il contesto della ricerca riguarda la necessità di sviluppare metodi di controllo efficaci e flessibili per i quadrotor, i quali devono affrontare dinamiche complesse, incertezze del modello e incertezze dell'ambiente operativo. A questo proposito, MC-PILCO, che si basa su GPR, emerge come una tecnica promettente per apprendere la dinamica del sistema.

La ricerca ha comportato lo studio delle equazioni di dinamica del quadrotor e il loro utilizzo per simulare l'ambiente di apprendimento. Il modello GPR, utilizzato all'interno dell'algoritmo MC-PILCO, è stato addestrato utilizzando un kernel RBF e un modello di integrazione della velocità. Inoltre, è stata esaminata l'uso del metodo Subset of Data per approssimare il modello GPR, analizzando l'effetto della riduzione del numero di punti dati sulla precisione della previsione.

I risultati sperimentali, in ambienti simulati, mostrano che il modello GPR migliora con un aumento dei dati e che il metodo Subset of Data fornisce risultati accettabili entro un certo grado di approssimazione. L'efficacia del MC-PILCO è stata dimostrata confrontando i suoi risultati di controllo con quelli di altri controllori tradizionali già esistenti, mostrando miglioramenti notevoli.

Questo dimostra che l'approccio MC-PILCO proposto, che incorpora GPR, rappresenta una valida alternativa ai metodi di simulazione e controllo attualmente disponibili per i quadrotor, aprendo la strada per future ricerche in questo settore.

Contents

List of Figures

List of Acronyms

UAVs Unmanned Aerial Veichles

ML Machine Learning

RL Reinforcement Learning

MBLR Model-Based Reinforcement Learning

ODE Ordinary Differential Equation

GPs Gaussian Processes

GPR Gaussian Process Regression

SE Squared Exponential

RBF Radial Basis Function

BFF Base Fixed-Frame

INF Earth-Fixed Inertial Frame

DoF Degree of Freedom

BLDCM Brushless DC Motor

RBF Radial Basis Function

MSE Mean Squared Error

RMSE Root Mean Squared Error

SoD Subset of Data

SoR Subset of Regressor

PILCO Probabilistic Inference for Learning COntrol

EMA Exponential Moving Average

1

Introduction

In the contemporary engineering landscape, the domain of autonomous guidance is assuming escalating significance. The systems to which these techniques are applied are typically complex and nonlinear. A prime example includes unmanned aerial vehicles, specifically the quadrotor.
The distinct structure of these vehicles endows them with superior flexibility and stability compared to other aerial vehicles. Quadrotors have experienced increased popularity in various sectors such as agriculture, surveillance, cinema, and military applications in recent years.
Nevertheless, the challenges of system stabilization and maintaining robustness amidst environmental variations render the control of quadrotors a nontrivial task. Autonomous operations for aerial vehicles lean heavily on onboard stabilization and trajectory tracking capabilities, thus requiring considerable effort to ensure these vehicles attain stable flight capabilities.

Given the increasing demand for these vehicles across a variety of applications, the complexity of control challenges is consistently escalating. It is thus critical to develop progressively more efficient and effective methodologies.

To solve the issue of position control in a quadrotor, initial efforts gravitated towards methods of linearization and model simplification due to their computational simplicity and stable hover flight [1, 6]. Following advancements in modeling techniques and growing computational power, more sophisticated control techniques were developed that referred to nonlinear models [12, 3].

In recent years, the emergence of advanced control techniques, specifically Model-Predictive Control (MPC), has begun to shift the frontier of control [16,

5, 22]. These techniques have demonstrated significant improvements in the domain of trajectory control.

Reinforcement Learning algorithms have opened a new frontier in control. These have delivered impressive performances in numerous applications, offering the opportunity to learn control laws without any model knowledge, relying solely on interactions with the environment. However, in real-world robotics contexts, the necessity of environment interaction can pose considerable limitations. In the case of a quadrotor, a policy that is not yet perfected could interact with the environment leading to disastrous outcomes.
The focus of this thesis is to study, analyze, and test the techniques of Model-Based Reinforcement Learning, a strategy to augment the efficiency of the collected data, applied to quadrotor control. We aim to implement MC-PILCO [2], which has already exhibited excellent results in robotic control applications, and evaluate its suitability for this application and analyze the results.

This thesis work is primarily divided into five chapters:

1. *Chapter 2*: This chapter begins with a qualitative introduction to the operating principles of a quadrotor. From these principles, we derive its nonlinear model while discussing potential challenges with its representation and offering alternative solutions. In the end, we design straightforward PID controllers to test and validate the derived models [1, 14].
2. *Chapter 3*: In this chapter, we introduce the predictive model used to estimate the evolution of the quadrotor over time, specifically leveraging the Gaussian Process Regression approach. We first introduce various models, then specifically discuss training for learning the dynamics of the quadrotor. The results of the training phase will be presented, analyzing the precision for the quantity of data used. We will conclude with an analysis of model approximation methods, which will enable predictions with a lesser amount of data - a capability playing a crucial role in subsequent chapters.
3. *Chapter 4*: This chapter provides a brief introduction to Reinforcement Learning algorithms and Model-Based Reinforcement Learning, with a particular focus on MC-PILCO.
4. *Chapter 5*: This chapter compiles all the results obtained by applying the algorithm in two different contexts. Initially, on a nominal model described by the ODE derived in Chapter 2, and subsequently, the algorithm has been

readjusted and utilized for controlling a quadrotor on a simulator capable of simulating real-world uncertainties with higher fidelity.

5. *Chapter 6*: This final chapter draws conclusions and presents the future developments of this project.

2

Quadrotor Model

Designing a controller of any kind necessitates the derivation of a precise model of our quadrotor as a foundational step. This process starts with an examination of the quadrotor's attributes, the reference frame necessary to depict its pose, and a method to represent its orientation. Subsequently, we establish the quadrotor dynamics model employing the Newton-Euler method, disregarding environmental noise and aerodynamic impacts. Once the model was completed, we could construct a controller and simulate its behavior utilizing the quadrotor state's differential equation.

Our aim was to design two distinct PID controllers. The primary purpose of the first one is to stabilize the quadrotor at a specific altitude and orientation, although, as will be seen, it allows the quadrotor to move freely in the xy plane. This controller will solely be used for the validation of model learning. Conversely, the second PID will have a slightly more intricate structure, but its architecture enables it to control the orientation to move and stabilize the quadrotor at a specified position.

2.1 Quadrotor Characteristic

A quadrotor should be classified as a rotary-wing aircraft, which in our scenario, is propelled by four rotors positioned at the ends of arms extending from a symmetric frame [25]. To study the dynamics, we first establish the reference frames. For our purpose, are sufficient two right-handed coordinate systems, an Earth-Fixed Inertial Frame (INF) coordinates system and a Base

Fixed-Frame (BFF) which is attached to the quadrotor body.
Different alignment of the reference frame with respect to their body is reported in the literature. Thus, we chose to place both *z-axis* oriented downwards [19, 20, 25]. For the body-fixed frame attached to the quadrotor, we chose the plus configuration [25] with the axes x and y in line with two arms and its origin placed at its center of mass. The final reference frames choice is shown in (Figure 2.1) where at the beginning, the two orientations are aligned. The motion space of

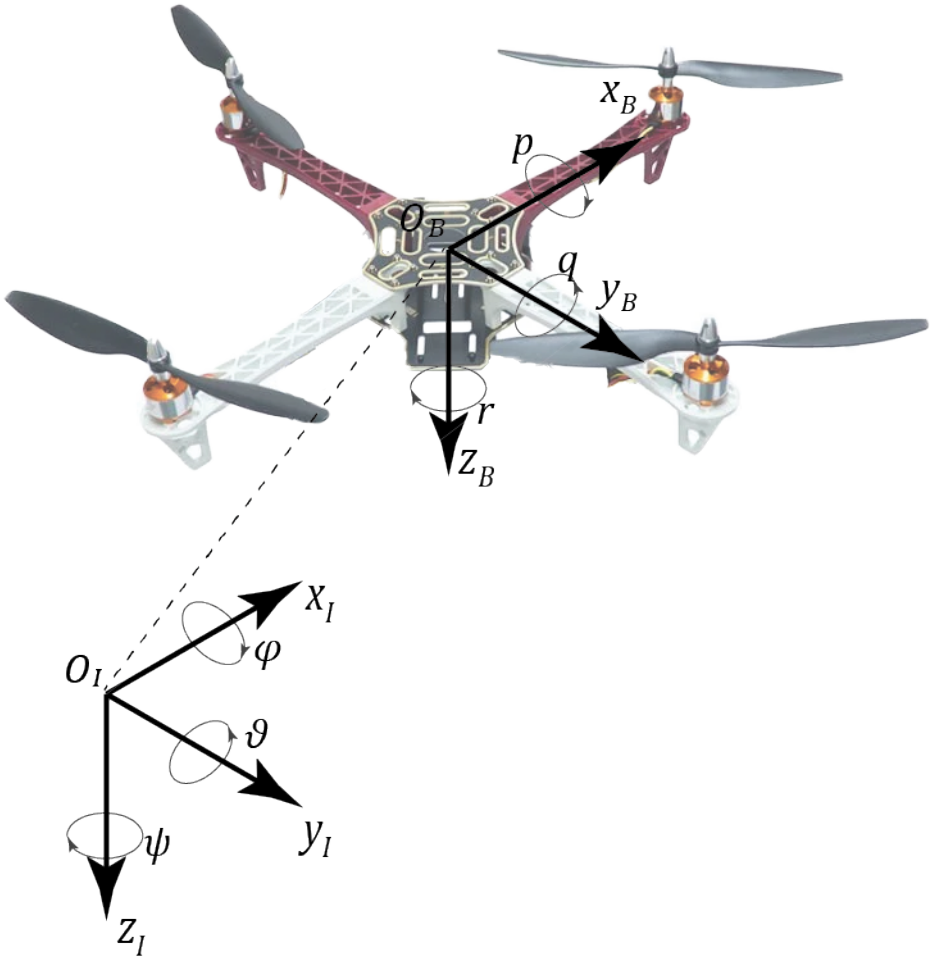


Fig. 2.1: Quadrotors body-fixed and inertial coordinate systems

the rigid body aircraft can be described by its position and its orientation with respect to its center of mass, thus, requiring six Degree of Freedom (DoF) to define the pose in time.

Quadrotor Motion Now that we have defined how the pose is described, we are interested in how we can control its position and orientation specifically.

Our actuation is provided by the propeller whose rotational speed generates a thrust force and a reactive torque [20, 25] that can be exploited for our purpose. Our quadrotor is made of four motors, each controlling a propeller. This means that our model will have six DoF but only four inputs to control its pose. This makes our quadrotor an under-actuated system.
The main maneuver, as illustrated in Figure 2.2, includes forward/backward and left/right translation, altitude variation, and the yaw rotation (*z-axis*), and each of this motion can be obtained from the relative speed difference between each motor. To move the quadrotor forward along the x_B-axis, a rotation around the y_B-axis is required, this inclination can be achieved by reducing the forward motor speed and increasing the rear one. This will create a difference in thrust generated by the two motors, lower for the front and higher rear motor, thus applying a torque on the y_B axis that produces the rotation needed. For a translation along the y_B, the same reasoning is applied, using this time the left/right motor. This thrust variation will generate the torque on the x_B axis, rotating the quadrotor according to the speed difference. For a rotation of the yaw angle, namely a rotation around the z_B axis, a different principle is exploited. This motion, indeed, is realized from the reactive torque produced by the rotor. The blade rotates in the reverse direction alternatively, and a change in the speed of the opposite pairs will produce a reactive torque that will make the body rotate on its z_B axis. Lastly, when the four-rotor speeds are exactly the same, the quadrotor has all thrust in balance. Therefore, no rotation is produced. Then, by changing the speed of all the motors simultaneously, it is possible to modify its altitude. These basic control sequences can then be combined to create a complex and sophisticated trajectory.

Rotors As said the main forces and torques are given by rotors driven by motors. These rotors are composed of a Brushless DC Motor (BLDCM) and a double-blade propeller. The main characteristic of this type of motor is how the commutation is achieved with a change of the input voltage, instead of the mechanical commutator, brush, of the classical DC motor [19]. This guarantee to the BLDCM a longer life and a higher torque/weights ratio and, despite there being additional problems related to its voltage control circuit.

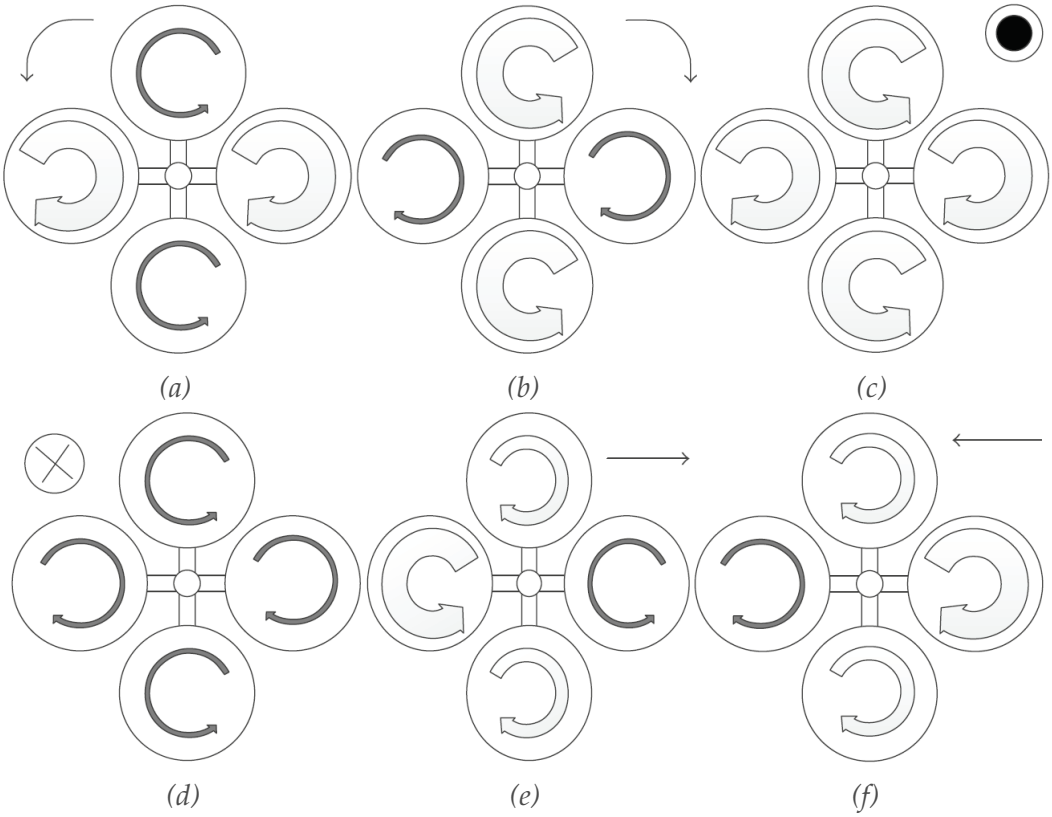


Fig. 2.2: Quadrotor motion dynamics [25]: (a), (b) difference in torque to manipulate the yaw angle ψ; (c), (d) altitude motion due to balanced torques; (e), (f) difference in thrust to manipulate roll (ϑ) and pitch (φ)

2.2 Euler's Angles

The quadrotor can be assumed and described as a rigid body. The way that best describes its orientation with respect to the INF is with the Euler's angle. This rule, developed by Leonard Euler, allows us to perfectly describe the orientation of a rigid body by means of three angles that make a 3-dimensional space [21]. We will refer to the Roll-Pitch-Yaw (RPY) angles that are typically used in the (aero)nautical fields to describe the attitude of an aircraft. For our purpose we will resort to the XYZ angles $\boldsymbol{\Theta} = \left[\varphi, \vartheta, \psi\right]$ defined as $\varphi \in [-\pi, \pi]$ for rotation around x-axis, $\vartheta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right[$ for rotation around y-axis and $\psi \in [-\pi, \pi]$ for rotation around z-axis. This angle represents a sequence of three elemental rotations, and together, they are a minimal representation of the body BFF orientation seen by the INF, which is assumed as a known orientation. This

rotation is given by the matrices:

$$\boldsymbol{R}_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\varphi) & -s(\varphi) \\ 0 & s(\varphi) & c(\varphi) \end{bmatrix}, \tag{2.1}$$

$$\boldsymbol{R}_y(\vartheta) = \begin{bmatrix} c(\vartheta) & 0 & s(\vartheta) \\ 0 & 1 & 0 \\ -s(\vartheta) & 0 & c(\varphi) \end{bmatrix}, \tag{2.2}$$

$$\boldsymbol{R}_z(\psi) = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.3}$$

where, for notations necessity we used $c(\varphi) = \cos\varphi$, $c(\vartheta) = \cos\vartheta$, $c(\psi) = \cos\psi$, $s(\varphi) = \sin\varphi$, $s(\vartheta) = \sin\vartheta$ and $s(\psi) = \sin\psi$.
To find the resulting frame orientation of the rigid body with respect to the INF, we have to consider the composition of these three rotation matrices, i.e.

$$\begin{aligned} \boldsymbol{R}_B^W &= \boldsymbol{R}_B^W(\psi, \vartheta, \varphi) = \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\vartheta)\boldsymbol{R}_x(\varphi) = \\ &= \begin{bmatrix} c(\vartheta)c(\psi) & s(\varphi)s(\vartheta)c(\psi) - c(\varphi)s(\psi) & c(\varphi)s(\vartheta)c(\psi) + s(\varphi)s(\psi) \\ c(\vartheta)s(\psi) & s(\varphi)s(\vartheta)s(\psi) + c(\varphi)c(\psi) & c(\varphi)s(\vartheta)s(\psi) - s(\varphi)c(\psi) \\ -s(\vartheta) & s(\varphi)c(\vartheta) & c(\varphi)c(\vartheta) \end{bmatrix}. \end{aligned} \tag{2.4}$$

Singularities These angles are widely used to describe the dynamics of a quadrotor. Despite the fact, that they are very intuitive and easy to interpret and visualize, they suffer from singularities [8]. This happens specifically when we reach the pitch $\vartheta = \pi/2$ obtaining the rotation matrix

$$\begin{aligned} \boldsymbol{R}_B^W(\psi, \pi/2, \varphi) &= \boldsymbol{R}_z(\psi)\boldsymbol{R}_y(\pi/2)\boldsymbol{R}_x(\varphi) = \\ &= \begin{bmatrix} 0 & s(\varphi - \psi) & c(\varphi - \psi) \\ 0 & c(\varphi - \psi) & -s(\varphi - \psi) \\ -1 & 0 & 0 \end{bmatrix}. \end{aligned} \tag{2.5}$$

We observe that the resulting matrix is a function of both φ and ψ, but there is only one DoF because only the difference $\varphi - \psi$ affects the resulting rotation.

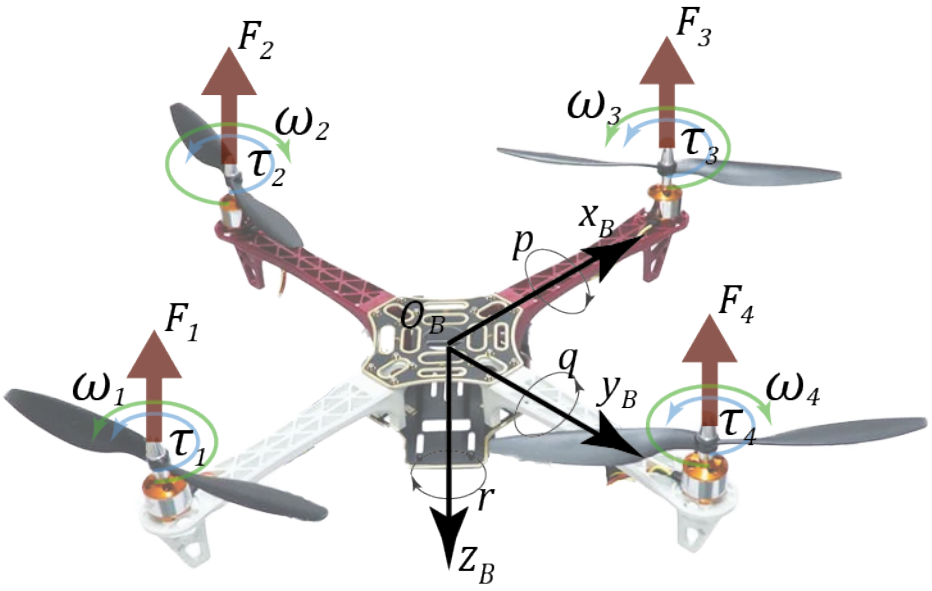


Fig. 2.3: Quadrotor rotors torques and forces

These conditions lead to mathematical issues for the dynamic model with that specific orientation. Moreover, the representation by RPY angles goes hand in hand with the computation of sine and cosine, which increases the computational cost. A valid alternative, more efficient and singularity-free, might be obtained by describing the quadrotor dynamic exploiting quaternion. This tool will be discussed in Section 2.4, also presenting an alternative with quaternions in addition to the classical dynamic model.

2.3 Quadrotor Dynamic Model

We now want to look for the dynamic model of the quadrotor. Two different methods can be applied to solve the dynamic problem: the Euler-Lagrange and the Newton-Euler formalism. It has been noted that the Newton-Euler method is easy to be understood and accepted physically despite the compact formulation and generalization shown by Euler-Lagrange formalism [25].
As we can see in Figure 2.3, each rotor will provide its contribution in terms of force and torque, according to the motor speed ω_i and blade configuration. Our goal is to find how the interactions of the quadrotor with the environment generate these forces and how they contribute to the whole body dynamics.

2.3.1 Generalized Coordinates

We define, as a first step, the generalized coordinates

$$\begin{bmatrix} \mathbf{\Gamma} \\ \mathbf{\Theta} \end{bmatrix} = \begin{bmatrix} x & y & z & \varphi & \vartheta & \psi \end{bmatrix}^T \in \mathbb{R}^6, \tag{2.6}$$

where the vector $\mathbf{\Gamma} = [x, y, z]^T \in \mathbb{R}^3$ denotes the position of the quadrotor center of mass with respect to the INF and $\mathbf{\Theta} = [\varphi, \vartheta, \psi]^T \in \mathbb{R}^3$ are the three Euler angles (respectively Yaw, Pitch, Roll) as defined in Section 2.2. For the BFF, we will consider its linear and angular velocity, respectively

$$\begin{aligned} \boldsymbol{\eta} &= [u \ v \ w]^T, \\ \mathbf{\Omega} &= \begin{bmatrix} p & q & r \end{bmatrix}^T. \end{aligned} \tag{2.7}$$

The coordinates defined in ((2.6)) and ((2.7)) are related together by the relation

$$\dot{\mathbf{\Gamma}} = \boldsymbol{R}_B^W \cdot \boldsymbol{\eta} \tag{2.8}$$

$$\dot{\mathbf{\Theta}} = \boldsymbol{T} \cdot \mathbf{\Omega} \tag{2.9}$$

with $\dot{\mathbf{\Gamma}} = [\dot{x}, \dot{y}, \dot{z}]^T \in \mathbb{R}^3$ and $= \dot{\mathbf{\Theta}} = [\dot{\varphi}, \dot{\vartheta}, \dot{\psi}]^T \in \mathbb{R}^3$. For angular velocities transformation the matrix T come, due to integral interpretation, from the relation [11]

$$\mathbf{\Omega} = \begin{bmatrix} \dot{\varphi} \\ 0 \\ 0 \end{bmatrix} + \boldsymbol{R}_x(\varphi)^T \begin{bmatrix} 0 \\ \dot{\vartheta} \\ 0 \end{bmatrix} + \boldsymbol{R}_x(\varphi)^T \boldsymbol{R}_y(\vartheta)^T \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} := T_\Theta^{-1} \dot{\mathbf{\Theta}}, \tag{2.10}$$

thus

$$T_\Theta^{-1} = \begin{bmatrix} 1 & 0 & -s(\vartheta) \\ 0 & c(\varphi) & c(\vartheta)s(\varphi) \\ 0 & -s(\varphi) & c(\vartheta)c(\varphi) \end{bmatrix} \implies T_\Theta = \begin{bmatrix} 1 & s(\varphi)t(\vartheta) & c(\varphi)t(\vartheta) \\ 0 & c(\varphi) & -s(\varphi) \\ 0 & s(\varphi)/c(\vartheta) & c(\varphi)/c(\vartheta) \end{bmatrix} \tag{2.11}$$

where $t(\cdot) = tan(\cdot)$.

2.3.2 Torques and Forces

As previously said the kinematics and dynamics model of a quadrotor will be derived based on Newton-Euler formalism. We will make the following assumptions [14, 24]:

- The structure is rigid and symmetrical.
- The center of gravity of the quadrotor coincides with the body fixed frame origin.
- The propellers are rigid.
- Thrust and drag are proportional to the square of the propellers speed.

From the first assumption we can simplify the inertia matrix used, namely, we have the inertia tensor defined as [19]

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix}. \tag{2.12}$$

Now to derive the dynamic equations, we must consider all the forces and the torques that act on the body [19]. In the BFF, the torque moment must also account for the Coriolis effect, thus

$${}^{B}\boldsymbol{\tau} = \frac{d\boldsymbol{L}}{dt} + \boldsymbol{\Omega} \times \boldsymbol{L} = \boldsymbol{I} \cdot \dot{\boldsymbol{\omega}}_B + \boldsymbol{\Omega} \times (\boldsymbol{I} \cdot \boldsymbol{\Omega}). \tag{2.13}$$

Developing the cross-product, we find the equations

$$\begin{aligned} {}^{B}\tau_x &= I_x \dot{p} + (I_z - I_y) qr \\ {}^{B}\tau_y &= I_y \dot{q} + (I_x - I_z) rp \\ {}^{B}\tau_z &= I_z \dot{r} + (I_y - I_x) pq \end{aligned} \tag{2.14}$$

Similarly, the translation forces by the time derivative of the momentum $\boldsymbol{P} = m \cdot \boldsymbol{\eta}$ thus, for the translation in the BFF, we have

$${}^{B}\boldsymbol{F} = \left(\frac{d\boldsymbol{P}}{dt}\right) + \boldsymbol{\Omega} \times \boldsymbol{P} = m\left(\dot{\boldsymbol{\eta}} + \boldsymbol{\Omega} \times \boldsymbol{\eta}\right) \tag{2.15}$$

leading to the equations

$$\begin{aligned} {}^{B}F_x &= m\left(\dot{u} + qw - vr\right) \\ {}^{B}F_y &= m\left(\dot{v} + ru - wp\right). \\ {}^{B}F_z &= m\left(\dot{w} + pv - qu\right) \end{aligned} \tag{2.16}$$

To wrap up, until now, we have considered the quadrotor as a rigid body whose motion equations are subject to external force ${}^{B}\boldsymbol{F} \in \mathbb{R}^3$ and torque ${}^{B}\boldsymbol{\tau} \in \mathbb{R}^3$. According to the Newton-Euler equations we arrive can write ((2.14)) and ((2.16)) in the compact form [25] given by

$$\begin{bmatrix} mI_{3\times3} & \mathbf{0} \\ \mathbf{0} & I_{3\times3} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{\eta}} \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\Omega} \times m\boldsymbol{\eta} \\ \boldsymbol{\Omega} \times I\boldsymbol{\Omega} \end{bmatrix} = \begin{bmatrix} {}^{B}\boldsymbol{F} \\ {}^{B}\boldsymbol{\tau} \end{bmatrix}. \tag{2.17}$$

We now consider the forces and the torques that will be present in equations ((2.17)). The main interactions acting between the quadrotor and the environment are [19]

1. gyroscopic effect;
2. propeller drag torque;
3. thrust;
4. earth gravity;
5. aerodynamic force.

Initially, we will neglect point 5, however, this is only a first assumption and it will be considered in a further section.

Gyroscopic Effect

The gyroscopic effect is the tendency of the rotating body to maintain a steady direction of its axis of rotation. For each rotor, we consider its rotational frame similar to the BFF. Thus, considering rotor with positive angular speed $\boldsymbol{\omega}_j$, $j = \{2, 4\}$, we can write the gyroscopic moment as

$${}^{B}_{G}\boldsymbol{\tau}_j = \boldsymbol{I}_j \cdot \dot{\boldsymbol{\omega}}_j + \boldsymbol{\Omega} \times \left(\boldsymbol{I}_j \cdot \boldsymbol{\omega}_j\right), \tag{2.18}$$

where $\boldsymbol{I}_j$ is the gyroscopic inertia of the j-th rotor. We now consider that the direction of $\boldsymbol{\omega}_j$ coincide with the z-axis of the BFF coordinates system, thus

$I_{j,x} = I_{j,y} = 0$. Moreover, we assume all the rotor to be identical and therefore $I_{j,z} = I_G$, finally leading to the equations

$$\begin{aligned} {}^{B}_{G}\tau_{j,x} &= I_G \omega_j q \\ {}^{B}_{G}\tau_{j,y} &= -I_G \omega_j p \\ {}^{B}_{G}\tau_{j,z} &= I_G \dot{\omega}_j \end{aligned} \quad . \tag{2.19}$$

Note that this torque is found for clockwise rotation while for motor $j = \{1, 3\}$ with a counterclockwise rotation is sufficient to change the sign of ${}^{B}_{G}\boldsymbol{\tau}_{j,x}$ and ${}^{B}_{G}\boldsymbol{\tau}_{j,y}$.

Air Drag

Propellers, depending on their shape and rotation direction, are subject to the reaction of the air onto them. This interaction is called *drag* torque. Again we consider the propeller rotation axis to be aligned with the BFF z-axis, The drag torque for the clockwise motor is modeled as

$${}^{B}_{D}\tau_j = c_D \rho A R^2 \left(\omega_j R\right)^2 = c_D \rho \pi R^5 \omega_j^2 = k_D \omega_J^2 \tag{2.20}$$

where

- c_D is the non-dimensional drag torque coefficient;
- ρ $[kg/m^3]$ is the air density;
- A $[m^2]$ is the area of the propeller disk;
- R $[m]$ is the propeller radius;
- k_D $[kgm^2]$ is the resulting dimensional drag torque coefficient.

Now we can sum all torques that act on the rotor for the z-axis of the BFF. For simplicity notation, we will drop the ^{B} frame reference and the j index, for instance, we will consider $\tau_D =^{B}_{D} \tau_z$. Summing all the torques that act on the rotor, we obtain

$$\sum \tau = I_G \dot{\omega} = M_E - M_B - M_L \tag{2.21}$$

where

- $M_E = k_t i_a$ is the electromagnetic torque,
- $M_B = B_a \omega$ is the friction torque,

- $M_L = M_D$ is the load torque that we consider to be only the drag force.

Considering now all body-fixed coordinate system components, the magnetic torque generated by the BLDCM is $\boldsymbol{M}_E = \boldsymbol{M}_G + \boldsymbol{M}_D + \boldsymbol{M}_B$. This consideration give, for rotors $j = \{2, 4\}$, the equation from the air-frame perspective

$$\begin{aligned} {}^{B}_{E}\tau_{j,x} &= -I_G \omega_j q \\ {}^{B}_{E}\tau_{j,y} &= I_G \omega_j p \\ {}^{B}_{E}\tau_{j,z} &= -\left(I_G \dot{\omega}_j + k_d \omega_j^2 + B_a \omega_j\right) \end{aligned} \quad . \tag{2.22}$$

For the rotors $j = \{1, 3\}$, we have only to change signs in equation (2.22) due to the different rotation directions.

Thrust

The thrust force is probably the main force that acts on the quadrotor. It is generated by the propeller rotation, and it is used to lift and transnational purposes. We will consider its direction to be always aligned with the BFF z-axis. The thrust force, for a rotor $j = \{1, 2, 3, 4\}$ is modeled as

$$T_j = {}^{B}_{T} F_{z,j} = -c_T \rho \pi R^4 \omega_j^2 = -k_T \omega_j^2 \tag{2.23}$$

where the coefficient is defined in the same way as the drag torque. Now considering all the rotors, the total thrust force is

$${}^{B}_{T}F_z = \sum_{j=1}^{4} T_j = -k_T \sum_{j=1}^{4} \omega_j^2. \tag{2.24}$$

We define with l_a the length of each arm, assumed equal to all the arms, namely the distance in the XY BFF plane from the rotor rotational axis to the drone center of mass. The thrust difference generated by the propeller in the x-axis produces a torque on the y-axis and vice-versa

$$\begin{aligned} {}^{B}_{T}\tau_x &= l_a(T_4 - T_2) \\ {}^{B}_{T}\tau_y &= l_a(T_1 - T_3) \\ {}^{B}_{T}\tau_z &= 0 \end{aligned} \quad . \tag{2.25}$$

Earth Gravity

Gravity applies a weight force to the center of mass that is modeled in the inertial frame as

$$ {}^{W}_{g}\boldsymbol{F} = \begin{bmatrix} 0 & 0 & mg \end{bmatrix}^{T}, \tag{2.26} $$

where $g = 9.81 m/s^2$. Now resorting to the inverse rotation matrix defined in equation (2.4), i.e. $\boldsymbol{R}^{B}_{W} = \boldsymbol{R}^{WT}_{B}$, the gravity force upon the quadrotor is obtained as

$$ {}^{B}_{g}\boldsymbol{F} = \boldsymbol{R}^{B}_{W} \cdot {}^{W}_{g}\boldsymbol{F} = \begin{bmatrix} -mg \cdot c(\vartheta) \\ mg \cdot s(\varphi)c(\vartheta) \\ mg \cdot c(\varphi)c(\vartheta) \end{bmatrix}. \tag{2.27} $$

Assuming that the quadrotor mass and gravitational centers coincide, no moment is generated by its weight force [19].

2.3.3 State-Space Model

At this time, we defined all the force and torque components that interact with the quadrotor, thus, exploiting the relation given by equation (2.17). By that, we are able to build the state space model with the equation of motion. For our model, we will consider the state from the variable defined by equation (2.6) and (2.7), so

$$ \boldsymbol{x} = \begin{bmatrix} x & y & z & \varphi & \vartheta & \psi & u & v & w & p & q & r \end{bmatrix}^{T} \in \mathbb{R}^{12}. \tag{2.28} $$

For the input, we choose the 4 motors' speed

$$ \boldsymbol{u} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 & \omega_4 \end{bmatrix}^{T} \in \mathbb{R}^{4}. \tag{2.29} $$

Now that we defined the state of our model and its input we can define its dynamic evolution. Specifically, we are interested in the vector

$$ \dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\varphi} & \dot{\vartheta} & \dot{\psi} & \dot{u} & \dot{v} & \dot{w} & \dot{p} & \dot{q} & \dot{r} \end{bmatrix}^{T} \in \mathbb{R}^{12}. \tag{2.30} $$

The first six states have already been defined in (2.8) and (2.9), while for the evolution of the velocities of the quadrotor concerning its BFF, we must resort to the equations and assumption done in section 2.3.2. Considering first the

moments, we take equation (2.22), with an inverted sign to have it as perceived by the quadrotor air-frame, [19], and equation (2.25). From equation (2.14), isolating the angular acceleration we get

$$\begin{cases} \dot{p} = \frac{l_a}{I_x}(T_4 - T_2) + \frac{1}{I_x}\sum_{j=1}^{4} {}^{B}_{E}\tau^{-}_{j,x} + \frac{(I_y - I_z)}{I_x} qr \\ \dot{q} = \frac{l_a}{I_y}(T_1 - T_3) + \frac{1}{I_y}\sum_{j=1}^{4} {}^{B}_{E}\tau^{-}_{j,y} + \frac{(I_x - I_z)}{I_y} pr \\ \dot{r} = \frac{1}{I_z}\sum_{j=1}^{4} {}^{B}_{E}\tau^{-}_{j,z} \end{cases} . \tag{2.31}$$

Following the same procedure, we can exploit equations (2.24) and (2.27), inserting them in (2.16) and isolating the translational accelerations, yields the force equations set as

$$\begin{cases} \dot{u} = vr - wq - g \cdot s(\vartheta) \\ \dot{v} = wp - ur + g \cdot s(\varphi)c(\vartheta) \\ \dot{w} = uq - vp + g \cdot c(\varphi)c(\vartheta) + \frac{1}{m}\sum_{j=1}^{4} T_j \end{cases} . \tag{2.32}$$

These equations give all the necessary to define the quadrotor dynamic described by its motion equations (2.8), (2.9), (2.31) and (2.32). As the last step, we can make explicit the last two equations set as a function of motor speed obtaining

the final state space evolution

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{u}) = \begin{cases} \dot{x} = \left[c(\vartheta)c(\psi)\right] u + \left[s(\varphi)s(\vartheta)c(\psi) - c(\varphi)s(\psi)\right] v + \\ \qquad\qquad + \left[c(\varphi)s(\vartheta)c(\psi) + s(\varphi)s(\psi)\right] w \\ \dot{y} = \left[c(\vartheta)s(\psi)\right] u + \left[s(\varphi)s(\vartheta)s(\psi) + c(\varphi)c(\psi)\right] v + \\ \qquad\qquad + \left[c(\varphi)s(\vartheta)s(\psi) - s(\varphi)c(\psi)\right] w \\ \dot{z} = \left[-s(\vartheta)\right] u + \left[s(\varphi)c(\vartheta)\right] v + \left[c(\varphi)c(\vartheta)\right] w \\ \dot{\varphi} = p + \left[s(\varphi)t(\vartheta)\right] q + \left[c(\varphi)t(\vartheta)\right] r \\ \dot{\vartheta} = \left[c(\varphi)\right] q + \left[-s(\varphi)\right] r \\ \dot{\psi} = \left[\frac{s(\varphi)}{c(\vartheta)}\right] q + \left[\frac{c(\varphi)}{c(\vartheta)}\right] r \\ \dot{u} = vr - wq - g \cdot s(\vartheta) \\ \dot{v} = wp - ur + g \cdot s(\varphi)c(\vartheta) \\ \dot{w} = uq - vp + g \cdot c(\varphi)c(\vartheta) - \frac{k_T}{m} \sum_{j=1}^{4} \omega_j^2 \\ \dot{p} = \frac{l_a}{I_x} k_T \left(\omega_2^2 - \omega_4^2\right) + \frac{I_G}{I_x} q \sum_{j=1}^{4} \omega_j (-1)^j + \frac{(I_y - I_z)}{I_x} qr \\ \dot{q} = \frac{l_a}{I_y} k_T \left(\omega_3^2 - \omega_1^2\right) - \frac{I_G}{I_y} p \sum_{j=1}^{4} \omega_j (-1)^j + \frac{(I_z - I_x)}{I_y} pr \\ \dot{r} = \frac{1}{I_z} \sum_{j=1}^{4} \left(I_G \dot{\omega}_j + k_D \omega_j^2 + B_a \omega_j\right) (-1)^j + \frac{(I_x - I_y)}{I_z} pq \end{cases} . \tag{2.33}$$

2.4 Quaternion Modeling

As anticipated in Section 2.2 Euler's representation may fall in a singular configuration. As we can see by expression (2.5) if the pitch angle is $\pi/2$ the rotation matrix will depend only on the difference $\varphi - \psi$ translating in a loss of a DoF. To deal with this problem we need a different representation for the rotation that allows singular-free modeling of our quadrotor dynamics. Quaternions will allow us to describe our quadrotor orientation, namely, its RPY angles, overcoming singularity issues [8].

2.4.1 Quaternion Math and Properties

We can talk of a quaternion as a particular description of a rotation around a vector. Quaternion is defined as a hyper-complex number [16, 13] of rank 4.

The most popular approach to describe a quaternion is

$$\begin{aligned} \boldsymbol{q} &= q_0 + q_1\boldsymbol{i} + q_2\boldsymbol{j} + q_3\boldsymbol{k} \\ \boldsymbol{q} &= \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T \end{aligned}, \tag{2.34}$$

where the quaternion elements from q_1 to q_3 are called the vector part of the quaternion, while q_0 is the scalar part. Multiplication of two quaternions $\boldsymbol{p}$, $\boldsymbol{q}$ if both express a rotation represents a combined rotation. Their product is performed by the Kronecker product and, just as rotation, is a non-commutative operation

$$\boldsymbol{p} \otimes \boldsymbol{q} = \begin{bmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2 \\ p_0q_2 - p_1q_3 + p_2q_0 + p_3q_1 \\ p_0q_3 + p_1q_2 - p_2q_1 + p_3q_0 \end{bmatrix}. \tag{2.35}$$

For our purpose, we will assume all quaternion with unitary norms [13], namely

$$Norm(\boldsymbol{q}) = ||\boldsymbol{q}|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1, \tag{2.36}$$

and the inverse of the unitary quaternion is

$$Inv(\boldsymbol{q}) = \boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{||\boldsymbol{q}||} = \boldsymbol{q}^*. \tag{2.37}$$

To conclude our properties analysis we want to define the quaternion derivative, which will become particularly useful later. From the possible representation [13] we are considering as if our angular velocity vector $\boldsymbol{\Omega}$ is in the body frame of reference [8], specifically

$$\dot{\boldsymbol{q}}_{\boldsymbol{\Omega}}(\boldsymbol{q}, \boldsymbol{\Omega}) = \frac{1}{2} \begin{bmatrix} 0 \\ \boldsymbol{\Omega} \end{bmatrix} \otimes \boldsymbol{q} \tag{2.38}$$

Quaternion Elementary Rotation A unitary quaternion can be used as a rotation operator, however, the transformation resort both to the quaternion and its conjugate. If we want rotate a vector $\boldsymbol{n}$ according to the representation given by $\boldsymbol{q}$ we have the equation

$$\boldsymbol{m} = \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ \boldsymbol{n} \end{bmatrix} \otimes \boldsymbol{q}^*. \tag{2.39}$$

To find the elementary rotation expressed by quaternion it is sufficient to replace $\boldsymbol{n}$ with the x, y, and z axis, obtaining the vector part

$$\begin{aligned} R_x(\boldsymbol{q}) = \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \otimes \boldsymbol{q}^* &= \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 \\ 2\left(q_1 q_2 + q_0 q_3\right) \\ 2\left(q_1 q_3 - q_0 q_2\right) \end{bmatrix} \\ R_y(\boldsymbol{q}) = \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \otimes \boldsymbol{q}^* &= \begin{bmatrix} 2\left(q_1 q_2 - q_0 q_3\right) \\ q_0^2 - q_1^2 + q_2^2 - q_3^2 \\ 2\left(q_2 q_3 + q_0 q_1\right) \end{bmatrix}. \\ R_z(\boldsymbol{q}) = \boldsymbol{q} \otimes \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \otimes \boldsymbol{q}^* &= \begin{bmatrix} 2\left(q_1 q_3 + q_0 q_2\right) \\ 2\left(q_2 q_3 - q_0 q_1\right) \\ q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \end{aligned} \tag{2.40}$$

To represent the quaternion for our purpose we exploit the property [13] that relates with the Euler angle $(\varphi, \vartheta\psi)$ as

$$\boldsymbol{q} = \begin{bmatrix} c(\varphi/2)c(\vartheta/2)c(\psi/2) + s(\varphi/2)s(\vartheta/2)s(\psi/2) \\ s(\varphi/2)c(\vartheta/2)c(\psi/2) - c(\varphi/2)s(\vartheta/2)s(\psi/2) \\ c(\varphi/2)s(\vartheta/2)c(\psi/2) + s(\varphi/2)c(\vartheta/2)s(\psi/2) \\ c(\varphi/2)c(\vartheta/2)s(\psi/2) - s(\varphi/2)s(\vartheta/2)c(\psi/2) \end{bmatrix}, \tag{2.41}$$

while from quaternion to Euler angle is

$$\begin{bmatrix} \varphi \\ \vartheta \\ \psi \end{bmatrix} = \begin{bmatrix} atan2\left(2(q_0 q_1 + q_2 q_3), q_0^2 - q_1^2 - q_2^2 + q_3^2\right) \\ asin\left(2(q_0 q_2 - q_3 q_1)\right) \\ atan2\left(2(q_0 q_3 + q_1 q_2), q_0^2 + q_1^2 - q_2^2 - q_3^2\right) \end{bmatrix} \tag{2.42}$$

Quaternion Rotation Matrix We can now combine the elementary rotation expression found in (2.40) to obtain the rotation matrix of a vector as a function

of quaternion [8, 13]

$$
\begin{aligned}
{}^{Q}R_{B}^{W} &= \begin{bmatrix} R_x(\boldsymbol{q}) & R_y(\boldsymbol{q}) & R_z(\boldsymbol{q}) \end{bmatrix} = \\
&= \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2\left(q_1q_2-q_0q_3\right) & 2\left(q_1q_3+q_0q_2\right) \\ 2\left(q_1q_2+q_0q_3\right) & q_0^2-q_1^2+q_2^2-q_3^2 & 2\left(q_2q_3-q_0q_1\right) \\ 2\left(q_1q_3-q_0q_2\right) & 2\left(q_2q_3+q_0q_1\right) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} .
\end{aligned} \tag{2.43}
$$

Another interesting characteristic of the rotation matrix expressed in terms of a quaternion is the fact that its element does not
On the other hand, the inverse transformation is derived directly by the quaternion inverse, defined in (2.37) and is given by

$$
\begin{aligned}
{}^{Q}R_{W}^{B} &= \left({}^{Q}R_{B}^{W}\right)^{-1} = \left({}^{Q}R_{B}^{W}\right)^{T} = \\
&= \begin{bmatrix} q_0^2+q_1^2-q_2^2-q_3^2 & 2\left(q_1q_2+q_0q_3\right) & 2\left(q_1q_3-q_0q_2\right) \\ 2\left(q_1q_2-q_0q_3\right) & q_0^2-q_1^2+q_2^2-q_3^2 & 2\left(q_2q_3+q_0q_1\right) \\ 2\left(q_1q_3+q_0q_2\right) & 2\left(q_2q_3-q_0q_1\right) & q_0^2-q_1^2-q_2^2+q_3^2 \end{bmatrix} .
\end{aligned} \tag{2.44}
$$

2.4.2 Quaternion Based Dynamic Model

To define the quadrotor model, we keep the assumption of its structure and the dynamic unchanged. It is necessary to modify the state that no longer resorts to the RPY angle but is replaced with its quaternion representation. Our new state is

$$
\boldsymbol{x}^{Q} = \begin{bmatrix} x & y & z & \boldsymbol{q}^T & u & v & w & p & q & r \end{bmatrix}^T \in \mathbb{R}^{13}, \tag{2.45}
$$

where $\boldsymbol{q}$ is defined as in equation (2.41). To derive the new model of the quadrotor dynamics, we must modify the evolution defined by the set of equations

(2.33) according to the new state representation, thus

$$\dot{\boldsymbol{x}}^{Q} = \begin{cases} \dot{x} = \left[q_0^2 + q_1^2 - q_2^2 - q_3^2\right] u + \left[2\left(q_1 q_2 - q_0 q_3\right)\right] v + \\ \qquad + \left[2\left(q_1 q_3 + q_0 q_2\right)\right] w \\ \dot{y} = \left[2\left(q_1 q_2 + q_0 q_3\right)\right] u + \left[q_0^2 - q_1^2 + q_2^2 - q_3^2\right] v + \\ \qquad + \left[2\left(q_2 q_3 - q_0 q_1\right)\right] w \\ \dot{z} = \left[2\left(q_1 q_3 - q_0 q_2\right)\right] u + \left[2\left(q_2 q_3 + q_0 q_1\right)\right] v + \\ \qquad + \left[q_0^2 - q_1^2 - q_2^2 + q_3^2\right] w \\ \dot{q_0} = \frac{1}{2}\left[-q_1 p - q_2 q - q_3 r\right] \\ \dot{q_1} = \frac{1}{2}\left[q_0 p - q_3 q + q_2 r\right] \\ \dot{q_2} = \frac{1}{2}\left[q_3 p + q_0 q - q_1 r\right] \\ \dot{q_3} = \frac{1}{2}\left[-q_2 p + q_1 q + q_0 r\right] \\ \dot{u} = vr - wq - g \cdot 2\left(q_1 q_3 - q_0 q_2\right) \\ \dot{v} = wp - ur + g \cdot 2\left(q_2 q_3 + q_0 q_1\right) \\ \dot{w} = uq - vp + g \cdot \left(q_0^2 - q_1^2 - q_2^2 + q_3^2\right) + \frac{k_D}{m}\sum_{j=1}^{4}\omega_j^2 \\ \dot{p} = \frac{l_a}{I_x} k_T\left(\omega_4^2 - \omega_2^2\right) + \frac{I_G}{I_x} q \sum_{j=1}^{4}\omega_j (-1)^j + \frac{\left(I_y - I_z\right)}{I_x} qr \\ \dot{q} = \frac{l_a}{I_y} k_T\left(\omega_1^2 - \omega_3^2\right) + \frac{I_G}{I_y} p \sum_{j=1}^{4}\omega_j (-1)^j + \frac{\left(I_z - I_x\right)}{I_y} pr \\ \dot{r} = \frac{1}{I_z}\sum_{j=1}^{4}\left(I_G \dot{\omega}_j + k_D \omega_j^2 + B_a \omega_j\right)(-1)^j + \frac{\left(I_x - I_y\right)}{I_z} pq \end{cases} . \tag{2.46}$$

At this point, we have the quadrotor dynamic model with orientation based on quaternion representation. It is clear how this expression does not suffer from problem-related to singularity. Moreover, it is not required anymore to compute any trigonometric function as sin or cos, lightening the computational cost.

Alternative to quaternions In the following sections, our primary task remains to learn a model that can best approximate the dynamics of the quadrotor. However, it will be seen that it can be practical to exploit only the state elements representing speed and to integrate the position elements. A representation with quaternions would make the integration step unusable, so we prefer to opt for representations without quaternions. To avoid the occurrence of singularities, we will adopt a representation of the system's orientation by augmenting the state with the sine and cosine functions in place of the respective angles.

2.5 Quadrotor Model in a Single Reference Frame

Up to now, we found the state-space model defined by the state in (2.28) or, resorting to quaternions, (2.45). Looking at that equation, it is immediately noticed that they describe a classical physical system, where the system pose, and the corresponding velocities make the state. However, this information is expressed in two different reference frames. The position refers to the INF while the velocities are expressed consistently to the BFF. This expression is convenient for finding the equation of motion and exploiting them for the model simulation process. On the other hand, however, for model learning, we might want a different state representation where both pose, and velocities are expressed in the INF.

2.5.1 State Redefinition

In Chapter 2, we found the state-space model defined by the state in (2.28) or, resorting to quaternions, (2.45). Looking at that equation, it is immediately noticed that they describe a classical physical system, where the system pose, and the corresponding velocities make the state. However, this information is expressed in two different reference frames. The position refers to the INF while the velocities are expressed consistently to the BFF. This expression is convenient for finding the equation of motion and exploiting them for the model simulation process. On the other hand, however, for model learning, we might want a different state representation where both pose, and velocities are expressed in the INF.
According to that, we aim to define our new state

$$\boldsymbol{x} = \begin{bmatrix} x & y & z & \varphi & \vartheta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\varphi} & \dot{\vartheta} & \dot{\psi} \end{bmatrix}^T \in \mathbb{R}^{12}, \tag{2.47}$$

whose dynamic will become

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & \dot{\varphi} & \dot{\vartheta} & \dot{\psi} & \ddot{x} & \ddot{y} & \ddot{z} & \ddot{\varphi} & \ddot{\vartheta} & \ddot{\psi} \end{bmatrix}^T \in \mathbb{R}^{12}. \tag{2.48}$$

To derive the entire dynamic, we can start from equations about torques and forces expressed in Section 2.3.2, keeping in mind that they are expressed concerning the BFF. Thus, to move in the INF, we will exploit the rotation matrix

(2.4) [14, 15] and the transformation (2.11).

Linear Equation of Motion The translational equation of motion should take into account the thrust force generated by the propeller and the gravity force, which in contrast to the previous case, we don't need to translate in the BFF. The equations of motion are now

$$m\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = \begin{bmatrix}0\\ 0\\ mg\end{bmatrix} + \boldsymbol{R}_B^W \begin{bmatrix}0\\ 0\\ -\sum_{j=1}^{4} T_j\end{bmatrix} \tag{2.49}$$

obtaining

$$\begin{cases} \ddot{x} = -\left[c(\varphi)s(\vartheta)c(\psi) + s(\varphi)s(\psi)\right]\frac{k_T}{m}\sum_{j=1}^{4}\omega_j^2 \\ \ddot{y} = -\left[c(\varphi)s(\vartheta)s(\psi) - s(\varphi)c(\psi)\right]\frac{k_T}{m}\sum_{j=1}^{4}\omega_j^2 \\ \ddot{z} = g - \left[c(\varphi)c(\vartheta)\right]\frac{k_T}{m}\sum_{j=1}^{4}\omega_j^2 \end{cases}. \tag{2.50}$$

Rotational Equations of Motion For the rotational variables at the moment, we try considering the total moments acting on the quadrotors similarly as in Section 2.3.2, namely

$$\begin{bmatrix}I_x & 0 & 0\\ 0 & I_y & 0\\ 0 & 0 & I_z\end{bmatrix}\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} + \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times \begin{bmatrix}I_x & 0 & 0\\ 0 & I_y & 0\\ 0 & 0 & I_z\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} + \begin{bmatrix}p\\ q\\ r\end{bmatrix} \times \begin{bmatrix}0\\ 0\\ I_G\cdot\Omega_r\end{bmatrix} = \begin{bmatrix}l_a\cdot\tau_x\\ l_a\cdot\tau_y\\ \tau_z\end{bmatrix}, \tag{2.51}$$

where the $\Omega_r = \sum_{j=1}^{4}\omega_j(-1)^j$ is the relative velocity.
From the expression (2.51), it is mandatory to notice that we are still obtaining our dynamics considering the angular velocities of our quadrotor. Expanding the last equations leads to

$$\begin{cases} \dot{p} = \frac{l_a}{I_x}k_T\left(\omega_4^2 - \omega_2^2\right) - \frac{I_G}{I_x}q\sum_{j=1}^{4}\omega_j(-1)^j + \frac{(I_y - I_z)}{I_x}qr \\ \dot{q} = \frac{l_a}{I_y}k_T\left(\omega_1^2 - \omega_3^2\right) + \frac{I_G}{I_y}\omega_\varphi\sum_{j=1}^{4}\omega_j(-1)^j + \frac{(I_z - I_x)}{I_y}pr \\ \dot{r} = \frac{1}{I_z}\sum_{j=1}^{4}\left(I_G\dot{\omega}_j + k_D\omega_j^2 + B_a\omega_j\right)(-1)^j + \frac{(I_x - I_y)}{I_z}pq \end{cases}. \tag{2.52}$$

We must notice that what we have just found are the equations of the angular acceleration rate, thus leading to the state defined as

$$\boldsymbol{x} = \begin{bmatrix} x & y & z & \varphi & \vartheta & \psi & \dot{x} & \dot{y} & \dot{z} & p & q & r \end{bmatrix}^T \in \mathbb{R}^{12}. \tag{2.53}$$

For the next phase, specifically during the behavior simulation, we must integrate our derivatives just found, resulting in a meaningless operation for the angular velocities p, q, and r. This is because angular velocities are not scalar quantities but vector quantities. In other words, angular velocity describes the change in orientation of an object around an axis of rotation and therefore has a specific direction and sense. On the other hand, the rotation angle is a scalar quantity obtained by integrating the angular velocity along a specific path. Integrating angular velocities without considering these factors can lead to inconsistent or incorrect results.
Differently, if we maintain the state as in (2.53), it is possible to integrate angular acceleration rates to calculate the change in angular velocity of an object around an axis of rotation. Angular acceleration rates represent the change in the angular velocity of an object over time. Since angular velocity is a vector quantity, angular acceleration is also a vector quantity. By integrating angular acceleration rates over time, one can obtain the change in angular velocity over time, i.e., the change in orientation of the object around the axis of rotation.
To overcome this problem, we keep our state defined by the RPY Euler angle and its respective angular velocities relating them through a transformation matrix defined as in (2.11). While integrating our state, this matrix allows us to map the Euler's angle derivative with their angular velocities. In this way, it is possible to define a coherent state accounting for the relation between these quantities. Wrapping together all our considerations, we come up with the state behavior

with the final state space dynamic

$$\dot{\boldsymbol{x}} = \begin{cases} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\varphi} = p + \left[s(\varphi)t(\vartheta)\right] q + \left[c(\varphi)t(\vartheta)\right] r \\ \dot{\vartheta} = \left[c(\varphi)\right] q + \left[-s(\varphi)\right] r \\ \dot{\psi} = \left[\frac{s(\varphi)}{c(\vartheta)}\right] q + \left[\frac{c(\varphi)}{c(\vartheta)}\right] r \\ \ddot{x} = -\left[c(\varphi)s(\vartheta)c(\psi) + s(\varphi)s(\psi)\right] \frac{k_T}{m} \sum_{j=1}^{4} \omega_j^2 \\ \ddot{y} = -\left[c(\varphi)s(\vartheta)s(\psi) - s(\varphi)c(\psi)\right] \frac{k_T}{m} \sum_{j=1}^{4} \omega_j^2 \\ \ddot{z} = g - \left[c(\varphi)c(\vartheta)\right] \frac{k_T}{m} \sum_{j=1}^{4} \omega_j^2 \\ \dot{p} = \frac{l_a}{I_x} k_T \left(\omega_4^2 - \omega_2^2\right) - \frac{I_G}{I_x} q \sum_{j=1}^{4} \omega_j \left(-1\right)^j + \frac{\left(I_y - I_z\right)}{I_x} qr \\ \dot{q} = \frac{l_a}{I_y} k_T \left(\omega_1^2 - \omega_3^2\right) + \frac{I_G}{I_y} \omega_\varphi \sum_{j=1}^{4} \omega_j \left(-1\right)^j + \frac{\left(I_z - I_x\right)}{I_y} pr \\ \dot{r} = \frac{1}{I_z} \sum_{j=1}^{4} \left(I_G \dot{\omega}_j + k_D \omega_j^2 + B_a \omega_j\right) \left(-1\right)^j + \frac{\left(I_x - I_y\right)}{I_z} pq \end{cases} . \tag{2.54}$$

2.5.2 Input Redefinition

Propellers driven with BLDCM generate quadrotor forces and moments. Thus, we always considered the input as each motor's angular speed ω_j. However, looking at our quadrotor structure, some considerations might be helpful to rewrite the input more conveniently. From equation (2.33) observe that the interaction of the propellers with the environment can generate thrust f_z and torques $\{\tau_x, \tau_y, \tau_z\}$. The overall thrust generated by propellers acts only on the z-axis acceleration, and the speed difference between each motor has a role in changing the three torque considered. For now, we neglect the presence of friction and consider only the steady-state behavior for step reference so that the speed derivative is zero. We can now define a new input vector given by

$$\begin{bmatrix} f_z \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} k_T & k_T & k_T & k_T \\ 0 & k_T & 0 & -k_T \\ -k_T & 0 & k_T & 0 \\ k_D & -k_D & k_D & -k_D \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} . \tag{2.55}$$

According to this new input, we aim to directly control the torques and forces that produce the basic quadrotor maneuver. Then we can translate our forces in the required speed according to the inverse of equation (2.55). After some computation, we come up with the velocities

$$\begin{aligned}
\omega_1 &= \sqrt{\tfrac{1}{4k_T} f_z - \tfrac{1}{2k_T}\tau_y - \tfrac{1}{4k_D}\tau_z} \\
\omega_2 &= \sqrt{\tfrac{1}{4k_T} f_z + \tfrac{1}{2k_T}\tau_x + \tfrac{1}{4k_D}\tau_z} \\
\omega_3 &= \sqrt{\tfrac{1}{4k_T} f_z + \tfrac{1}{2k_T}\tau_y - \tfrac{1}{4k_D}\tau_z} \\
\omega_4 &= \sqrt{\tfrac{1}{4k_T} f_z - \tfrac{1}{2k_T}\tau_x + \tfrac{1}{4k_D}\tau_z}
\end{aligned} . \tag{2.56}$$

At this point, we could see how physics can affect the quadrotor attitude seen by the INF.
The evolution of the state then can be easily described by the equations just found, since for the position and orientation, their change is given by the respective velocity, which is the other states' components. The dynamic of the quadrotor now depends on those forces as

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \ddot{\varphi} \\ \ddot{\vartheta} \\ \ddot{\psi} \end{bmatrix} = \begin{cases} -\left[c(\varphi)s(\vartheta)c(\psi) + s(\varphi)s(\psi)\right]\frac{f_z}{m} \\ -\left[c(\varphi)s(\vartheta)s(\psi) - s(\varphi)c(\psi)\right]\frac{f_z}{m} \\ g - \left[c(\varphi)c(\vartheta)\right]\frac{f_z}{m} \\ \frac{l_a}{I_x}\tau_x + \frac{I_G}{I_x}\dot{\vartheta}\cdot\Omega_r + \frac{(I_y - I_z)}{I_x}\dot{\vartheta}\dot{\psi} \\ \frac{l_a}{I_y}\tau_y + \frac{I_G}{I_y}\dot{\varphi}\cdot\Omega_r + \frac{(I_z - I_x)}{I_y}\dot{\varphi}\dot{\psi} \\ \frac{1}{I_z}\tau_z + \frac{(I_x - I_y)}{I_z}\dot{\varphi}\dot{\vartheta} \end{cases} . \tag{2.57}$$

2.6 Quadrotor Control

We can now move to the PID controller design that aims to reach a desired pose in terms of orientation and z altitude. While the control input of the former can be seen as a speed offset, ideally not null only when required to adjust the orientation, the latter is directly related to the propeller rotational speed. We can easily translate also the force f_z to be seen as an offset by setting a default initial speed that can generate enough thrust to compensate for the gravitational acceleration. According to the equation (2.54), all propellers must have the same

speed obtained from the relation

$$\omega_j^* = \sqrt{\frac{m}{4\left[c(\varphi)c(\vartheta)\right]k_T}g}. \tag{2.58}$$

Applying this as an initial speed guarantees the simulation starts at a stable altitude, and thus, the control output is just a propeller angular speed offset. This configuration can also be extended to the case of the PID responsible for the position control.

2.6.1 PID Orientation-Altitude Control

We have already introduced the goal of our first controller is to validate the models designed to ensure that all its representations return the same behavior. We only focus on controlling the altitude and quadrotor orientation acting on forces described in 2.5.2.
Since the force and torques act independently, we can design four different control structures so that each will contribute to each propeller according to equation (2.56). Mathematically speaking, we have

$$\begin{cases} f_z = K_{P,z}e_z(t) + K_{I,z}\int e_z(t) + K_{D,z}\frac{d}{dt}e_z(t), & e_z(t) = z(t) - z^* \\ \tau_x = K_{P,\varphi}e_\varphi(t) + K_{I,\varphi}\int e_\varphi(t) + K_{D,\varphi}\frac{d}{dt}e_\varphi(t), & e_\varphi(t) = \varphi^* - \varphi(t) \\ \tau_y = K_{P,\vartheta}e_\vartheta(t) + K_{I,\vartheta}\int e_\vartheta(t) + K_{D,\vartheta}\frac{d}{dt}e_\vartheta(t), & e_\vartheta(t) = \vartheta^* - \vartheta(t) \\ \tau_z = K_{P,\psi}e_\psi(t) + K_{I,\psi}\int e_\psi(t) + K_{D,\psi}\frac{d}{dt}e_\psi(t), & e_\psi(t) = \psi^* - \psi(t) \end{cases} \tag{2.59}$$

where z^*, φ^*, ϑ^* and ψ^* are the reference altitude and orientation desired. To set the controller parameters, we manually tuned the PID controllers. We initially tuned the only proportional parameters $K_{P,i}$ to have an oscillatory response. Then we refined the integrative and derivative parameters to give stability and improve performances as much as possible.

PID-orientation Controller Simulation We are now interested to see how our so-designed controllers suit the model in the previous section. To do so, we simulate the model under the input driven by the PID. The algorithm 1 shows the steps required to observe another behavior and build the PID for the z axis altitude control. The procedure is equivalent to equations (2.60). Finally, the

Algorithm 1 PID Algorithm for z altitude

Require: z^*, z, dt, e_{-1}, e_+
$e \leftarrow z - z^*$
$e_+ \leftarrow e_+ + e$ {Cumulative Error Increment}
$e_d \leftarrow (e - e_{-1})/dt$ {Differential Error Update}
$(K_{P,z}, K_{I,z}, K_{D,z}) \leftarrow (5.5, 1.75, 6.85)$
$u_z \leftarrow K_{P,z} \cdot e + K_{I,z} \cdot e_+ + K_{D,z} \cdot e_d$
if $u_z \leq 0$ **then**
  $u_z \leftarrow 0$ {avoid negative motor speed}
end if
return u_z

x, y position has been left free since they are strictly related to the quadrotor orientation. Thus, controlling the angles also means controlling the xy-plane position.
Despite speed performances not being a requirement for our purpose, we still want the controller to have a reasonable transient period. Several tests have been done for tuning and tracking evaluation providing different target altitudes and orientations.

Figure 2.4 shows a meaningful evolution of the quadrotor model for two different targets. From this example, we can make some considerations. Firstly, the controller can reach the desired target in a reasonable time according to generic quadrotor dynamics, reaching the target pose in a few seconds under a reasonable error. Secondly, the orientation variation strongly affects the position in the xy plane, confirming the relationship between the quadrotor states, also shown in the differential equations. Moreover, we can observe that a change in the orientation affects, as a perturbation, all the poses proportionally to the magnitude of this variation. These relationships between orientation and translation are at the base of quadrotor motion, and combining them allows the tracking of very complex trajectories. In the second half of the simulation, we can observe another behavior due to the changing of the orientation. Specifically, with the rotation of the ψ angle, we can see a change in the translation along the x axis. Specifically, the position change will start to reverse its direction. This is explained by the fact that we represent our orientation with absolute angles. Thus, the thrust responsible for the position change will be translated in the three axes according to the rotation matrix (2.4).

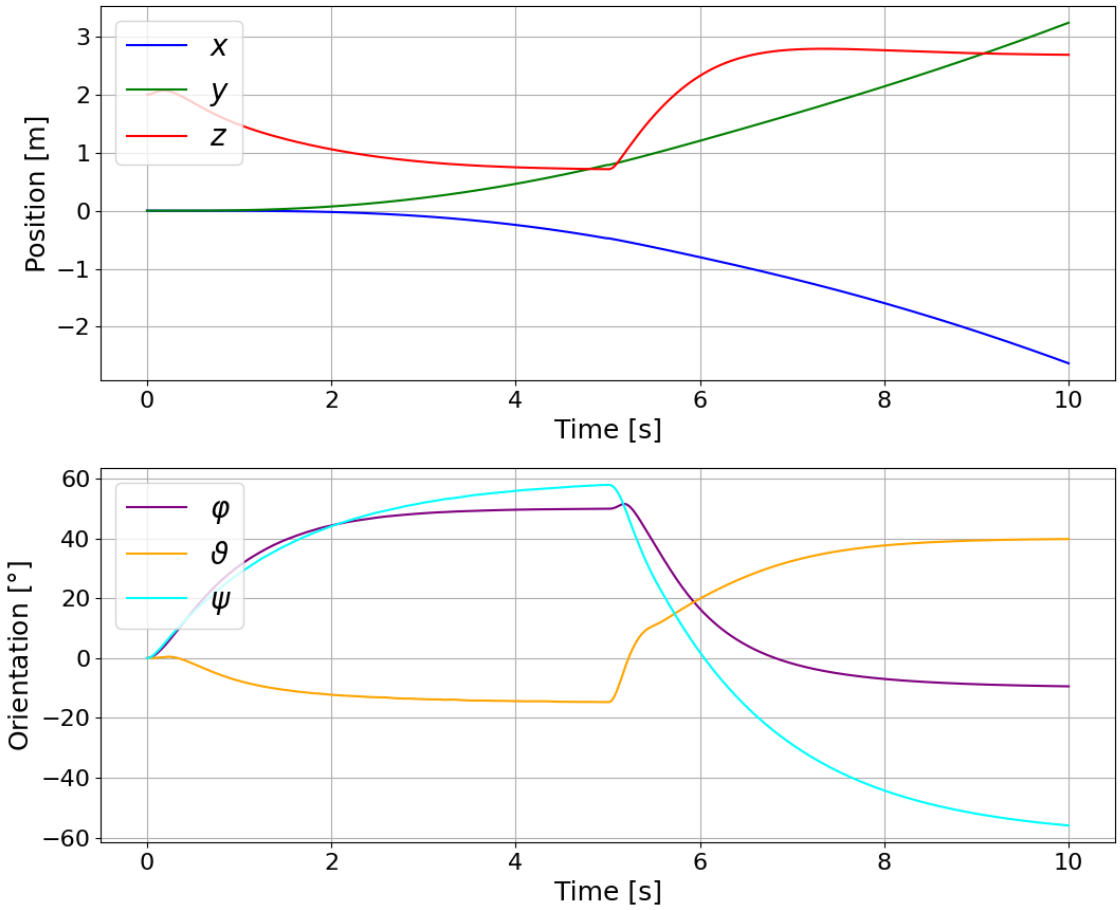


Fig. 2.4: PID Orientation Controller Response

2.6.2 PID Position Control

The PID controller designed for orientation can reach its goal quickly. However, reducing the goal to the altitude and orientation greatly limited the position control capability. In the next sections, we might want a controller capable of directly acting on the position for exploration purposes. To do so, a slight modification of the design is required.

Similar to the prior controller, forces and torques are provided by the same law as in (2.60). In this case, the main difference is due to the target angles φ^* and ϑ^* that are not fixed anymore but will be a function of a dedicated PID controller depending on the position error.

The new control law is now expressed as

$$\begin{cases} \vartheta_t^* = K_{P,x} e_x(t) + K_{I,x} \int e_x(t) + K_{D,x} \frac{d}{dt} e_x(t), & e_x(t) = x^* - x(t) \\ \varphi_t^* = K_{P,y} e_y(t) + K_{I,y} \int e_y(t) + K_{D,y} \frac{d}{dt} e_y(t), & e_y(t) = y^* - y(t) \\ f_z = K_{P,z} e_z(t) + K_{I,z} \int e_z(t) + K_{D,z} \frac{d}{dt} e_z(t), & e_z(t) = z(t) - z^* \\ \tau_x = K_{P,\varphi} e_\varphi(t) + K_{I,\varphi} \int e_\varphi(t) + K_{D,\varphi} \frac{d}{dt} e_\varphi(t), & e_\varphi(t) = \varphi_t^* - \varphi(t) \\ \tau_y = K_{P,\vartheta} e_\vartheta(t) + K_{I,\vartheta} \int e_\vartheta(t) + K_{D,\vartheta} \frac{d}{dt} e_\vartheta(t), & e_\vartheta(t) = \vartheta_t^* - \vartheta(t) \\ \tau_z = K_{P,\psi} e_\psi(t) + K_{I,\psi} \int e_\psi(t) + K_{D,\psi} \frac{d}{dt} e_\psi(t), & e_\psi(t) = \psi^* - \psi(t) \end{cases} \tag{2.60}$$

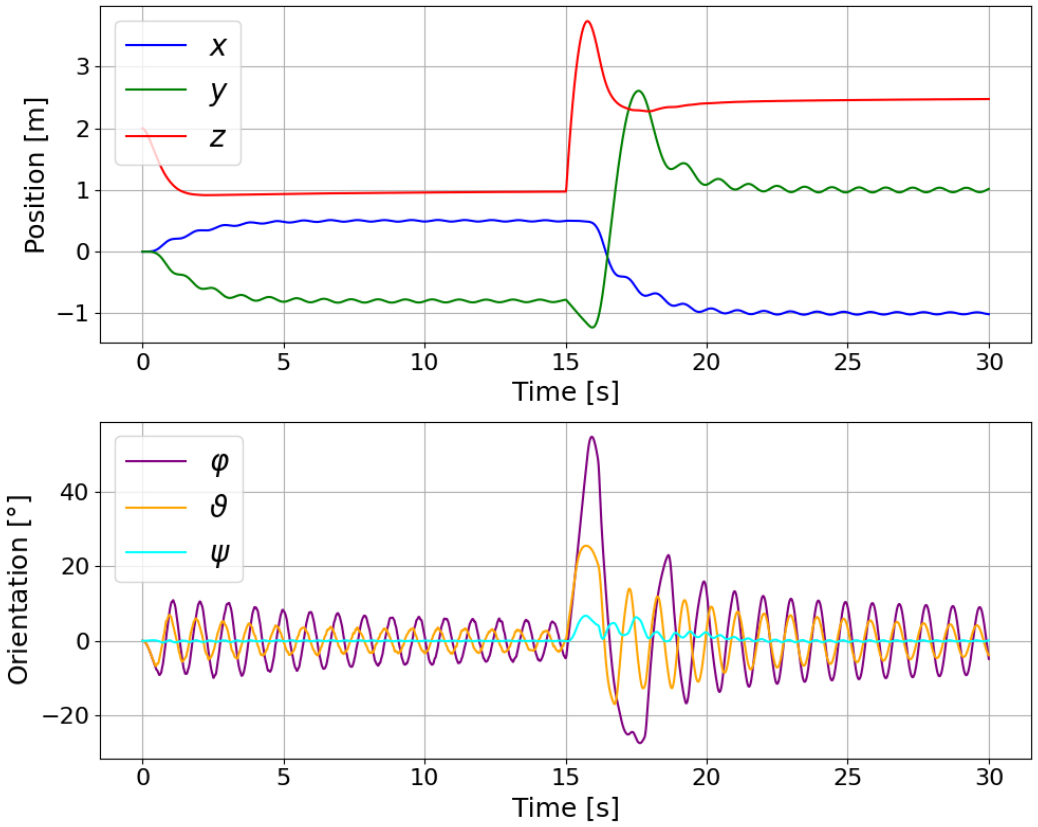


Fig. 2.5: PID Position Controller Response

PID-position Controller Simulation It is time to validate the new controller to check if it can track the reference position. As for the orientation PID controller, we test a sequence of two different reference targets, shown in Figure 2.5. The controller design has achieved its objective, but it's evident that it takes longer to stabilize at the desired coordinates, and the quadrotor's motion tends to be not robustly stable. The observed behavior is due to the higher complexity required

to control the angle with respect to the current position error that continuously changes according to the system position. This cascade dependency effect makes the quadrotor harder to stabilize, with more challenging parameters tuning.
Despite this type of controller being capable of achieving the established objective, its rise time and oscillatory behavior make it physically impractical for real-world applications. In such cases, the quadrotor will no longer operate in a nominal environment. Still, it will also have to deal with external disturbances caused by aerodynamic effects or environmental noise, such as wind.

3

Gaussian Processes for Regression

In machine learning tasks, regression is one of the fundamental algorithms for prediction.
Regression is a statistical tool to guess a real-valued continuous output based on input features. Given an observation set, defined as $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) \,|i = 1, \ldots, n\}$, the regression goal is to build a model that can fit the available data best so that can accurately predict the output value of new and unseen input data. The simplest model is the linear regression and is given by

$$f(\boldsymbol{x}) = \boldsymbol{x}^T \boldsymbol{w} \tag{3.1}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is the input vector describer by its d features, $\boldsymbol{w} \in \mathbb{R}^d$ is the weights vector associated with the input and $f(\cdot)$ is the model function. A biased term is often introduced. However, it can be easily integrated into the vector to maintain the compact form above. The final task is to find a line that employs a linear combination of its input vector to approximate the observed outcome $y \in \mathbb{R}$.

In Gaussian Process (GP) for linear regression is with the Bayesian analysis that sees the model as shown in (3.1) but considers the model with Gaussian noise, namely

$$y = f(\boldsymbol{x}) + \varepsilon \tag{3.2}$$

where ε is used to model a white noise that will be discussed, with further analysis, in Section 3.1.

Despite the simplicity of this model type, it presents limited flexibility for our purpose. Since the model we have to construct has a non-linear input-output

relationship, it's convenient to resort to a different interpretation that no longer considers the weights vector but works directly in the function space. This approach will be discussed in Section 3.2.

Projection of Inputs into Features Space Sometimes, as we just introduced, the regression model may lack expressiveness. A technique to overcome this limit is to project the input vector into some feature vector of higher dimension resorting to some function fixed for all projections, to preserve linearity. For instance, if we consider a scalar input x, a possible projection can be given by the space of powers of x: $\phi(x) = \left(1, x, x^2, \dots\right)^T$ that allows the implementation of polynomial regression. In conclusion, we can introduce the function ϕ, which maps a d-dimensional input vector $\boldsymbol{x}$ into an N dimensional feature space and, in matrix form, we can see $\Phi(\boldsymbol{X}) \in \mathbb{R}^{n \times N}$ as the aggregation of columns of $\phi(\boldsymbol{x}) \in \mathbb{R}^N$ for all n cases in the training set.

3.1 Weight-Space Model

We start our initial analysis from the more straightforward case, which is the Bayesian standard linear regression model with Gaussian noise, as defined in equation (3.1), (3.2). From the beginning, we assumed that the observation is corrupted by an additive noise which is modeled as an independent distributed Gaussian with zero mean and variance σ_n^2, namely

$$\varepsilon \sim \mathcal{N}\left(0, \sigma_n^2\right). \tag{3.3}$$

This last assumption on the model allows us to derive the *likelihood*, which describes the probability density of the observation given the parameters, and, due to the independence assumption on the training points, return

$$\begin{aligned} p\left(\boldsymbol{y}|X, \boldsymbol{w}\right) &= \prod_{i=1}^{n} p\left(y_i|\boldsymbol{X}_i, \boldsymbol{w}\right) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_n} \exp\left(-\frac{(y_i - \boldsymbol{x}_i^T \boldsymbol{w})^2}{2\sigma_n^2}\right) \\ &= \frac{1}{(2\pi\sigma_n^2)} \exp\left(-\frac{1}{2\sigma_n^2}|\boldsymbol{y} - X^T\boldsymbol{w}|^2\right) = \mathcal{N}\left(X^T\boldsymbol{w}, \sigma_n^2\boldsymbol{I}\right) \end{aligned}. \tag{3.4}$$

In addition, for the Bayesian formalism, we need to specify a prior for the distribution of the parameters, where typically is set to zero mean and covariance

matrix $\Sigma_p \in \mathbb{R}^{N\times N}$, so

$$\boldsymbol{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p) \in \mathbb{R}^N. \tag{3.5}$$

Prediction of new outcomes come from the posterior distribution over the weights, found resorting to Bayes' rule

$$p(\boldsymbol{w}|\boldsymbol{y}, X) = \frac{p(\boldsymbol{y}|X, \boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{y}|X)} \propto p(\boldsymbol{y}|X, \boldsymbol{w})p(\boldsymbol{w}) \sim \mathcal{N}(\bar{\boldsymbol{w}} = \frac{1}{\sigma_n^2}A^{-1}X\boldsymbol{y}, A^{-1}), \tag{3.6}$$

where $A = \sigma_n^2 XX^T + \sigma_p^{-1}$.

To conclude our analysis, reviewing the prediction procedure from the weights just found is interesting. We will consider the model with the inputs projected into feature space, so

$$f(\boldsymbol{x}) = \phi(\boldsymbol{x})^T\boldsymbol{w}. \tag{3.7}$$

This analysis is similar to the standard linear model, computed in [7], but substituted $X \in \mathbb{R}^{n\times d}$ for $\Phi(X) \in \mathbb{R}^{n\times N}$. For notation simplicity consider $\boldsymbol{\phi} = \phi(\boldsymbol{x})$ and $f_* = f(\boldsymbol{x}_*)$. Thus the predictive distribution, for a new unseen input $\boldsymbol{x}_*$, become

$$f_*|\boldsymbol{x}_*, X, \boldsymbol{y} \sim \mathcal{N}\left(\frac{1}{\sigma_n^2}\phi(\boldsymbol{x}_*)^T A^{-1}\boldsymbol{\phi}\boldsymbol{y}, \phi(\boldsymbol{x}_*)^T A^{-1}\phi(\boldsymbol{x}_*)\right) \tag{3.8}$$

with $A = \sigma_n^2\Phi\Phi^T + \Sigma_p^{-1}$. It is immediate to note that to make predictions with this equation is required to invert the matrix A, which might be an expensive procedure if the number of features is large. To overcome this problem, the last equation can be rewritten by exploiting the kernel tool K. However, this method will not be explored further for the current case since it will be discussed widely in Section 3.2.

3.2 Function-Space Model

A different approach is using a Gaussian Process (GP) to describe a distribution over function, so we are inferring directly in function space. We start again from the function $f(\boldsymbol{x})$, which describes a real process, assuming this time that it is completely described by its mean function $m(\boldsymbol{x})$ and its covariance function

$k(\boldsymbol{x}, \boldsymbol{x}')$ as:

$$\begin{cases} m(\boldsymbol{x}) = \mathbb{E}\left(f(\boldsymbol{x})\right) \\ k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}\left(\left(f(\boldsymbol{x}) - m(\boldsymbol{x})\right)\left(f(\boldsymbol{x}') - m(\boldsymbol{x}')\right)\right) \end{cases} . \tag{3.9}$$

We will then write our Gaussian process as

$$f(\boldsymbol{x}) \sim \mathcal{GP}(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}')). \tag{3.10}$$

and it is clear how the random variables represent the value of the function $f(\boldsymbol{x})$ at location $\boldsymbol{x}$. To better explain the Gaussian process, we consider the Bayesian linear regression model seen in Section 3.1 $f(\boldsymbol{x}) = \phi(\boldsymbol{x})^T \boldsymbol{w}$ with prior $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \Sigma_p)$. The mean and the covariance are given by:

$$\begin{cases} \mathbb{E}\left[f(\boldsymbol{x})\right] = \phi(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{w}] = 0 \\ \mathbb{E}\left[f(\boldsymbol{x}) f(\boldsymbol{x}')\right] = \phi(\boldsymbol{x})^T \mathbb{E}[\boldsymbol{w}\boldsymbol{w}^T] \phi(\boldsymbol{x}') = \phi(\boldsymbol{x})^T \Sigma_p \phi(\boldsymbol{x}') \end{cases} \tag{3.11}$$

showing that $f(\boldsymbol{x})$ and $f(\boldsymbol{x}')$ are jointly Gaussian with zero mean and covariance $\phi(\boldsymbol{x})^T \Sigma_p \phi(\boldsymbol{x}')$. It can be then shown that choosing a *squared exponential* (SE) covariance function

$$cov(\boldsymbol{x}_p, \boldsymbol{x}_q) = k(\boldsymbol{x}_p, \boldsymbol{x}_q) = \lambda^2 \exp\left(-\frac{1}{2}\left|\boldsymbol{x}_p - \boldsymbol{x}_q\right|^2\right), \tag{3.12}$$

it corresponds to a Bayesian linear regression model with infinite basis functions. We can also obtain the SE covariance function from the linear combination of an infinite number of Gaussian-shaped basis functions. The specification of the covariance function implies a distribution over functions. To show that one can take samples from the distribution of function evaluated at any number of points. Choosing some inputs point, $X_* \in \mathbb{R}^{n_* \times N}$, and resorting element-wise to (3.12) to write the covariance matrix. Then is possible to generate a random Gaussian vector with the covariance matrix just defined

$$f_* \sim \mathcal{N}\left(\boldsymbol{0}, K(X_*, X_*)\right), \tag{3.13}$$

where $K(X_*, X_*) = \left\{k(\boldsymbol{x}_p, \boldsymbol{x}_q)_{ij}, \forall i, j\right\} \in \mathbb{R}^{n_* \times n_*}$.
In the end, the main objective is to use the so-modeled f_* to make predictions of new and unobserved data. We will do a more in-depth analysis in the next paragraph considering two different scenarios: the case of noise-free and the

case of noisy observation.

Prediction with Noise-free Observation The simpler case to incorporate knowledge provided by the data about the function is by considering them noiseless and assuming the model

$$y = f(\boldsymbol{x}) \in \mathbb{R}. \tag{3.14}$$

Saying that observations are noise-free is equivalent to saying that we have the pairs $\{(\boldsymbol{x}_i, f_i | i = 1, \ldots, n)\}$, while the test outputs, used to make a prediction, are denoted with $\boldsymbol{f}_*$. By that the joint distribution of training outputs $\boldsymbol{f}$ and the test outputs $\boldsymbol{f}_*$ according to the prior is

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(X, X) & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix}\right). \tag{3.15}$$

Note that the covariance matrix is found evaluating all the pairs of n training and n_* test points so that $K(X, X_*) \in \mathbb{R}^{n \times n_*}$ denotes the $n \times n_*$ matrix and similarly for the other entries $K(X, X) \in \mathbb{R}^{n \times n}$, $K(X_*, X) \in \mathbb{R}^{n_* \times n}$ and $K(X_*, X_*) \in \mathbb{R}^{n_* \times n_*}$. To get the posterior distribution over functions, we need to restrict this prior joint distribution to contain only those functions which follow the observed data points. This operation is straightforward, corresponding to conditioning the joint Gaussian prior distribution on the observation, obtaining

$$\boldsymbol{f}_* | X, \boldsymbol{y}, X_* \sim \mathcal{N}\left(\hat{\boldsymbol{m}}_{\boldsymbol{f}_*}, \hat{\boldsymbol{\Sigma}}_{\boldsymbol{f}_*}\right), \tag{3.16}$$

with

$$\begin{aligned} \hat{\boldsymbol{m}}_{\boldsymbol{f}_*} &= K(X_*, X)K(X, X)^{-1}\boldsymbol{y}, \\ \hat{\boldsymbol{\Sigma}}_{\boldsymbol{f}_*} &= K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*) \end{aligned}. \tag{3.17}$$

Now the prediction of $\boldsymbol{f}_*$ can be sampled from the joint posterior distribution evaluating the mean and the covariance matrix from (3.16).

Prediction with Noisy Observation Realistically, we have no access to real function values but only to its corresponding noisy observation, namely $y = f(\boldsymbol{x}) + \varepsilon$. Assuming additive independent and identically distributed Gaussian noise ε with variance σ_s^2, the prior on the noisy observation becomes

$$cov(\boldsymbol{y}) = K(X, X) + \sigma_n^2 \boldsymbol{I}, \tag{3.18}$$

Due to the independence assumption, the noise is added on the diagonal of the covariance, compared with the noise-free model (3.14). The joint distribution of the observation with noise and the test values under the prior is seen as

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\boldsymbol{0}, \begin{bmatrix} K(X,X) + \sigma_n^2 \boldsymbol{I} & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right). \tag{3.19}$$

The conditional distribution derived, as equation (3.16), directly drive the predictive equations for GP regression

$$\boldsymbol{f}_* | X, \boldsymbol{y}, X_* \sim \left(\hat{\bar{\boldsymbol{m}}}_{\boldsymbol{f}_*}, \hat{\bar{\boldsymbol{\Sigma}}}_{\boldsymbol{f}_*}\right), \tag{3.20}$$

where

$$\begin{aligned} \hat{\bar{\boldsymbol{m}}}_{\boldsymbol{f}_*} &= \mathbb{E}\left[\boldsymbol{f}_* | X, \boldsymbol{y}, X_*\right] = K(X_*,X)\left[K(X,X) + \sigma_n^2\right]^{-1} \boldsymbol{y} \\ \hat{\bar{\boldsymbol{\Sigma}}}_{\boldsymbol{f}_*} &= K(X_*,X_*) - K(X_*,X)\left[K(X,X) + \sigma_n^2\right]^{-1} K(X,X_*) \end{aligned}. \tag{3.21}$$

The expression now can be written in a compact form with $K = K(X,X)$ and $K_* = (X_*,X_*)$, while for a single test input $\boldsymbol{x}_*$ we write $\boldsymbol{k}_*(\boldsymbol{x}_*) = \boldsymbol{k}_*$ denoting the covariance vector between the test point and the n training points. Notice that K can be computed once offline and then used for all successive predictions so it is possible to find the vector

$$\boldsymbol{\alpha} = \left[K + \sigma_n^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y}. \tag{3.22}$$

Using this compact notation, the prediction of a single test point reduces to

$$\begin{aligned} \bar{f}_* &= \boldsymbol{k}_*^T \boldsymbol{\alpha} \\ \mathbb{V}\left[\boldsymbol{f}_*\right] &= k(\boldsymbol{x}_*,\boldsymbol{x}_*) - \boldsymbol{k}_*^T (K + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{k}_* \end{aligned}. \tag{3.23}$$

It is clear how the mean prediction is a linear combination of n kernel functions, each one centered in a training point, namely

$$\bar{f}(\boldsymbol{x}_*) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}_*). \tag{3.24}$$

This final result shows how the flexibility of the function-view space model is vastly improved concerning the parameter-view space since the function shape can be arbitrarily modeled from the choice of the kernel function. This increases

the model's capability to deal with non-homogeneous and nonlinear data. However, it is essential to notice that this choice must be made appropriately to the problem.

3.3 Quadrotor Model Learning

We now have introduced the framework that will be considered to learn the model of a quadrotor system defined as in (2.54). Specifically, we will use the function-space model presented in 3.2 with noisy observation.

3.3.1 Kernel Choice

In the previous section, we introduced, among the many existing, the Squared Exponential (SE) kernel, or Radial Basis Function (RBF), which is a very popular covariance function. For several reasons, it might be convenient initially to keep this choice for our problems, defined in its general form

$$K(\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j) = \sigma_f^2 \exp\left(-\left(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j\right)^T \Lambda \left(\tilde{\boldsymbol{x}}_i - \tilde{\boldsymbol{x}}_j\right)\right). \tag{3.25}$$

In the function above, $\tilde{\boldsymbol{x}}_i, \tilde{\boldsymbol{x}}_j$ are two points in the input space for $i, j \in \{1, \ldots, n\}$, γ is a positive hyperparameter that controls the amplitude of the function (inversely proportional to the width of the radial basis functions), and σ^2 is a hyperparameter that controls the vertical amplitude of the covariance function. The first advantage is that the RBF kernel can approximate almost any continuous function, providing sufficient data. Secondly, this kernel can catch nonlinear relations between data points, which is a significant advantage considering our system's nature. Finally, another remarkable advantage is its computational simplicity since it only requires Euclidean norms between vectors and its exponential result. Yet, it offers other interesting properties, such as integrated regularization, transformation invariance, and only two hyperparameters. These characteristics make this kernel the best candidate to learn our Gaussian Processes (GPs) model. It must be taken into account, however, that the kernel performance might suffer from the higher dimensionality of the system. Before moving on to policy analysis and optimization, we must investigate whether the choice suits our problem and the computational cost regarding time and data requirements.

Kernel Parameters Tuning For the hyperparameters optimization, we must also include the parameter σ_n representing noise associated with the observation. In the learning task, our goal is to find the best parameters $\left(\sigma_f^2, \Lambda, \sigma_n\right)$ that let our model generalize our observation better. We can estimate the best parameter following the log-marginal likelihood maximization approach. We assume Λ to be a diagonal matrix, with its diagonal matrix, with its diagonal elements named length scales.

$$\log p(\boldsymbol{x}|\tilde{X}, \boldsymbol{\theta}) = -\frac{1}{2}\boldsymbol{x}^T K_{\boldsymbol{x}}^{-1}\boldsymbol{x} - \frac{1}{2}\log|K_{\boldsymbol{x}}| - \frac{n}{2}\log 2\pi \tag{3.26}$$

where $K_{\boldsymbol{x}} = K_{\boldsymbol{f}} + \sigma_n^2 I$ is the covariance matrix of the noisy $\boldsymbol{x}$ provided that K_f is the covariance matrix for the noise-free latent $\boldsymbol{f}$. We now explicitly write the marginal likelihood conditioned on the hyperparameters $\boldsymbol{\theta}$. To maximize the hyperparameters according to (**??**), we seek the partial derivatives of the marginal likelihood w.r.t the hyperparameters, namely

$$\begin{aligned}\frac{\partial}{\partial\theta_j}\log p\left(\boldsymbol{y}|X, \boldsymbol{\theta}\right) &= \frac{1}{2}\boldsymbol{y}^T K^{-1}\frac{\partial K}{\partial\theta_j}K^{-1}\boldsymbol{y} - \frac{1}{2}tr\left(K^{-1}\frac{\partial K}{\partial\theta_j}\right) \\ &= \frac{1}{2}\left(\left(\boldsymbol{\alpha}^T\boldsymbol{\alpha} - K^{-1}\right)\frac{\partial K}{\partial\theta_j}\right)\end{aligned} \tag{3.27}$$

where $\boldsymbol{\alpha} = K^{-1}\boldsymbol{y}$. We do not go further in explaining this approach since its not our primary purpose, and more in-depth detail can be found in [7].

3.3.2 Quadrotor Trajectory Generation

To move on to the learning phase, we must generate meaningful data points. This means generating a sequence of input/output $(\boldsymbol{x}_t, \boldsymbol{u}_t)$, where our input is the motor's rotational velocity and the output is the observed state at time t. In particular the state $\boldsymbol{x}_t$ is obtained starting from the state $\boldsymbol{x}_{t-1}$ and applying the input $\boldsymbol{u}_t$. In the previous section, we introduced a PID orientation controller explicitly designed for this purpose. It is capable of tracking some reference targets defined by the z-axis coordinates and the orientation angle $(\varphi, \vartheta, \psi)$. To generate meaningful data observations, the idea is to set, as a first step, a number N_{train} of random target and exploit our PID controllers to drive the quadrotor model state to the desired pose for a certain interval T_c. We can do this procedure for each target generated using the initial state reached by the

model at the last step for the previous target. Thus, if we require more data, increasing the N_{train} value is sufficient. Once applied all generated targets we will have our state/input sequence defined as

$$\left\{(\boldsymbol{x}_t, \boldsymbol{u}_t) \,|\boldsymbol{x}_t = \boldsymbol{f}(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t), t \in [T_c, N_{train} \cdot T_c], \boldsymbol{x}_0 = \bar{\boldsymbol{x}}\right\}. \tag{3.28}$$

The above trajectory is derived by the function $\boldsymbol{f}(\cdot)$ described by the Ordinary Differential Equation (ODE) 2.54, and the initial state $\boldsymbol{x}_0$ is chosen arbitrarily. The same reasoning is also applied to the performance evaluation generating a set of N_{test} trajectories used to analyze how well the model can predict the state evolution.

3.3.3 Model Training

It is now possible to start the model training with generated data defined as $\mathcal{D} = \{(\boldsymbol{u}_i, \boldsymbol{x}_i), i = 1, \ldots, n\}$, according to the previous section, where the total number of sample is $n = \frac{T_c}{dt} N_{train}$, $\boldsymbol{u}_i$ is the input vector and $\boldsymbol{x}_i$ is the state reached applying the input i. For the trajectory generation, we set the control time at $T_c = 2s$ for each target and a sampling time $dt = 0.02s$. This means that each target will provide additional 150 data to the model for training. As done in other studies involving angles [2], to avoid singularities the state $\boldsymbol{x}_t$ is replaced in the algorithm with the representation

$$\boldsymbol{x}_t^* = \left[\boldsymbol{\Gamma}, \dot{\boldsymbol{\Gamma}}, \boldsymbol{\Omega}, \sin(\varphi), \sin(\vartheta), \sin(\psi), \cos(\varphi), \cos(\vartheta), \cos(\psi)\right]^T \in \mathbb{R}^{15}. \tag{3.29}$$

Moreover, in all the following training, we considered all the state measurements with white noise with a standard deviation of 10^3. Finally, we define the input of the GP given by the combination of the states and the actual quadrotor model input, namely

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{x}_t^* \\ \boldsymbol{u}_t \end{bmatrix} \in \mathbb{R}^{19}. \tag{3.30}$$

One Step Ahead Prediction and Speed Integration Approach Typically in Gaussian Process Regression (GPR)-based, we model each state dimension's evolution with a distinct GP. We now denote with $\Delta_t^{(i)} = x_{t+1}^{(i)} - x_t^{(i)}$ the difference between the value of the i-th state at times $t+1$ and t. It can be convenient to desire each GP objective to find the *one step ahead* prediction expressed by the Δ_t

for each state separately.
Moreover, before moving to the model evaluation, we observe that our state is defined, in a compact form, as $\boldsymbol{x}_t = \left[\boldsymbol{q}_t^T, \dot{\boldsymbol{q}}_t^T\right]^T \in \mathbb{R}^{d_{\boldsymbol{x}}}$, where $\boldsymbol{q}_t^T \in \mathbb{R}^{d_{\boldsymbol{x}}/2}$ represent the quadrotor pose at time step t and $\dot{\boldsymbol{q}}_t^T \in \mathbb{R}^{d_{\boldsymbol{x}}/2}$ represent their derivatives concerning time. By considering this representation, we may want to exploit the relation between the components $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$. Thus, under the assumption of a sufficiently small (w.r.t. the application) sampling time T_s, it is reasonable to assume a constant acceleration between two consecutive time steps obtaining the evolution of $\boldsymbol{q}_t$

$$\boldsymbol{q}_{t+1} = \boldsymbol{q}_t + T_s\dot{\boldsymbol{q}}_t + \frac{T_s}{2}(\dot{\boldsymbol{q}}_{t+1} - \dot{\boldsymbol{q}}_t). \tag{3.31}$$

This means training only $d_{\boldsymbol{x}}/2$ GPs related to the velocities and adopting some integration technique to derive the evolution of $\boldsymbol{q}_t$.

3.4 Learning Result

Let us now delve into the performances of the learning procedure. Some metrics are particularly interesting: the *one step ahead* prediction precision, the long-term prediction accuracy, and the rollout cumulative error.
The *one step ahead* prediction precision refers to the difference between the predicted step $\hat{\Delta}t^{(i)}$ and the real difference in our test state evolution $\Delta_t^{(i)} = \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$. Prediction performances will be expressed as the Mean Squared Error (MSE) of all test samples for each GP.
On the other hand, long-term prediction refers to the complete evolution of each state component over the entire control period. To observe this, we compare the rollout of the test trajectory with the rollout w.r.t the behavior of the GP predictions provided the same input sequence. To quantify the error accumulation along the control periods, we calculate the RMSE between the trajectories' errors. Since we are estimating the velocity change at each time step, even a small error in the prediction leads to a different state with respect to the true evolution. This displacement will inevitably lead to a larger prediction error due to error integration phenomena. For this reason, we want also to investigate how the error propagates in time, observing the cumulative error for the full event horizon.
To evaluate the training process, we observe the behavior of the introduced metrics as the training data is increased. Specifically, we repeat the GP training for

different N_{train} trajectories and collect the GP one step ahead prediction MSE, the rollout over periods RMSE, and the cumulative error for N_{test} trajectories. The final result is obtained on a total of $N_{test} = 35$ trajectories of $T_c = 3s$ sampled with a period of $dt = 0.02s$, which translates into 150 samples for each new

target. The first training is performed with a single trajectory under the same conditions as the test ones. We then evaluate the model prediction performances by increasing the training trajectory by five at each iteration.
The overall evolution of the error, shown in figure 3.1, is reported by the mean and standard deviation of the test trajectories, where all the data is normalized according to the maximum error of the respective prediction. The learning behaviors are promising, showing a decreasing error and becoming more precise at every iteration interval, translating into successful model training. This consideration is also confirmed by looking at the cumulative error in figure 3.3, showing, for each GP, better results while increasing the number of data points. For each model, it is also interesting to note the strong correlation between the GP and cumulative rollout errors. This is evident, for instance, when comparing the first GP, which predicts the linear velocity variation on the x axis, with the rollout on the x position.
In conclusion, comparing the two plots, it is evident that each GP error strongly correlates to its respective rollout evolution. Moreover, we can observe that the learning performance becomes remarkable after already 15 train trajectories, meaning a total of $n \geq N_{train} \cdot (T_c/dt) = 2250$ data samples.

3.5 Model Approximation

As we have seen in chapter 3, GP brings high flexibility fully probabilistic model useful for regression tasks. Specifically for our case, we consider a model with additive noise as defined in 3.2 and 3.3. Moreover, the problem stated in section **??** requires the definition of six different GP regressors, each associated with a different model velocity.
Each GP is completely determined by a covariance function, the kernel, with some hyperparameter to be tuned. The search for optimal hyperparameters is the fundamental step in the learning process. Once those have been found, the model is ready to predict new observations. The optimization aims to maximize the marginal likelihood of the observed data under the GP model. This operation requires the following step:

1. computes the covariance matrix of the GP model using the current hyper-parameters and the chosen kernel;
2. perform inference on the training data to obtain the predictive distribution of the GP model and calculate the marginal likelihood;
3. computes the gradient of the marginal likelihood with respect to each hyperparameter;
4. using a chosen optimization algorithm to update the hyperparameters based on the computed gradient;
5. repeat steps 1-4 until a pre-specified convergence criterion is reached, such as a maximum number of iterations, a minimum change in the marginal likelihood, or the gradient norm below a specified threshold.

We know that the inference is derived by the distribution 3.20, which involves the inversions of a matrix of size $n \times n$ requiring $O(n^3)$ operations, with n number of training samples. This fact implies that with modern desktop machines, the exact implementation can only be built with a few thousand observations [18].

3.5.1 Subset of Data (SOD)

The Subset of Data (SoD) method is an approximation approach for GP that aims to reduce the computational cost and complexity of operations associated with the inference and training of the GP model. The SoD method is based on selecting a subset of the available data to approximate the GP.
The main idea of SoD is to select a representative subset $\mathcal{S}$ of size $m < n$ of the input data, where n is the total number of observations in the dataset. This subset is chosen in such a way as to preserve the essential information of the original dataset while reducing the computational cost associated with the GP. The subset selection can be done in several ways, for example, random sampling, stratified sampling, clustering, and error-based selection. For our model made by the six GP of the quadrotor velocities, the error-based selection was chosen by selecting the points that maximize a measure of error between the original GP and the approximated GP. In this case, the objective will be to maintain the points that have a prediction error between the original GP and the approximated GP beyond a certain threshold. Specifically, the selection criterion used is given by:

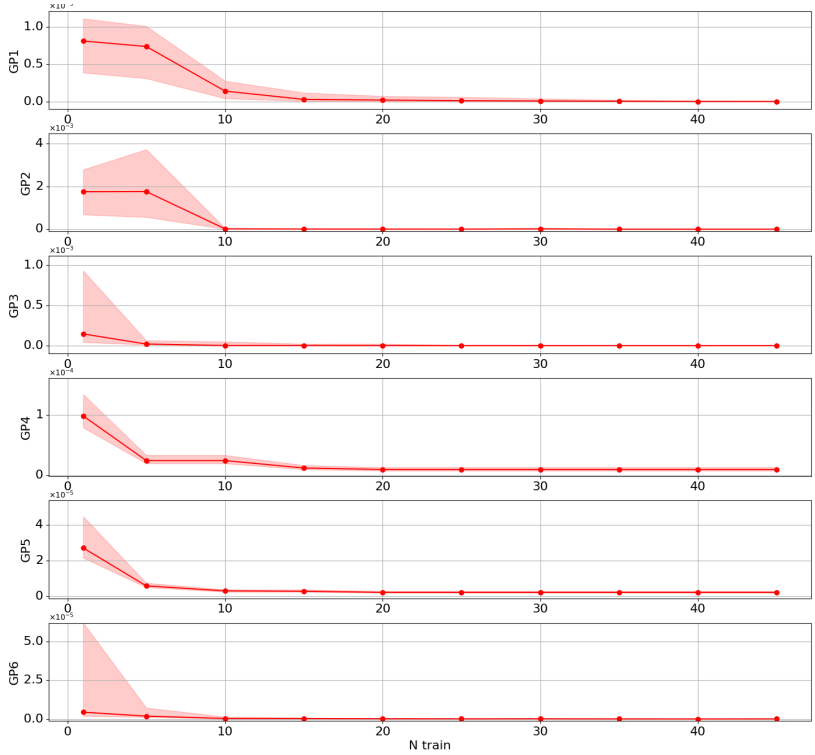
$$\left| \boldsymbol{f}(\boldsymbol{x}) - \tilde{\boldsymbol{f}}(\boldsymbol{x}) \right| > \epsilon, \quad \forall \boldsymbol{x} \in \mathcal{D}, \tag{3.32}$$

where ϵ is an arbitrarily chosen threshold, $f(\mathbf{x})$ is the prediction of the original GP for the input point $\mathbf{x}$, and $\tilde{f}(\mathbf{x})$ is the prediction of the approximated GP using the subset $\mathcal{S}$.
Error-based selection has the advantage of providing a more accurate approximation of the GP compared to other methods, as it considers each point's effect on the model's prediction. However, this method's computational cost may be higher than other methods, as it requires the calculation of the discrepancy for each point in the dataset. Moreover, it should be noted that this strategy might be sensitive to outliers.

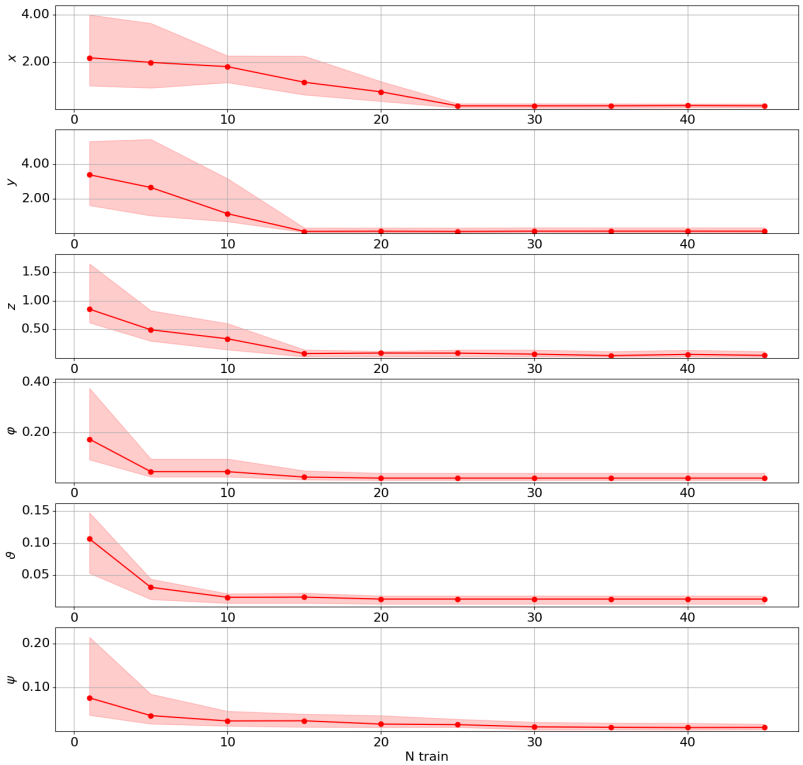
Approximation Result As we have just seen, the choice of points to select is made if the prediction error exceeds a certain threshold. This process must be repeated for each GP, which may have different selections for each threshold. We, therefore, wanted to observe, for a sequence of thresholds, how the models behaved in terms of performance, thus further evaluating the errors of the individual GP and the error of the rollouts compared to the number of selected points. For comparison, the model was trained by choosing a fixed value of samples for training generated by $N_{train} = 30$ trajectories, which, according to the model training, brings enough points to have a reliable model.
The process was repeated for the sequence of thresholds

$$\boldsymbol{\epsilon} = [0, 0.0025, 0.005, 0.01, 0.025, 0.05, 0.1, 0.25, 0.5] .$$

From the results obtained, shown in Figure 3.2, we notice, as one would expect, that as the number of selected data decreases, the performance of the model worsens. A promising result is that model performances improve with each training done with more data. This can be interpreted as a confirmation that the selection criteria is choosing the meaningful point. Evaluating this result concerning the full model, made of a total of $n = 4500$ samples, we immediately notice that the performance becomes comparable after just a few hundred data. Also, the cumulative error reported in figure 3.4 validates these results. In conclusion, looking at these results, we can now set a threshold for each GP, providing a sufficiently accurate velocity change prediction based on a small and limited subset of the original dataset. This will be a crucial step in the next section since this will improve data efficiency in terms of computational time and memory space.

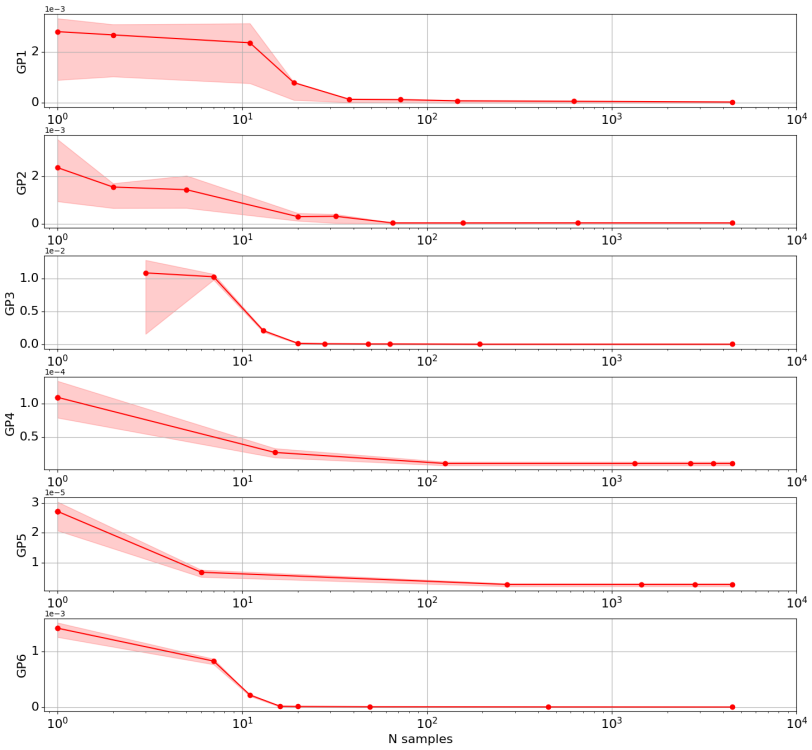


(a) GP one step ahead prediction MSE

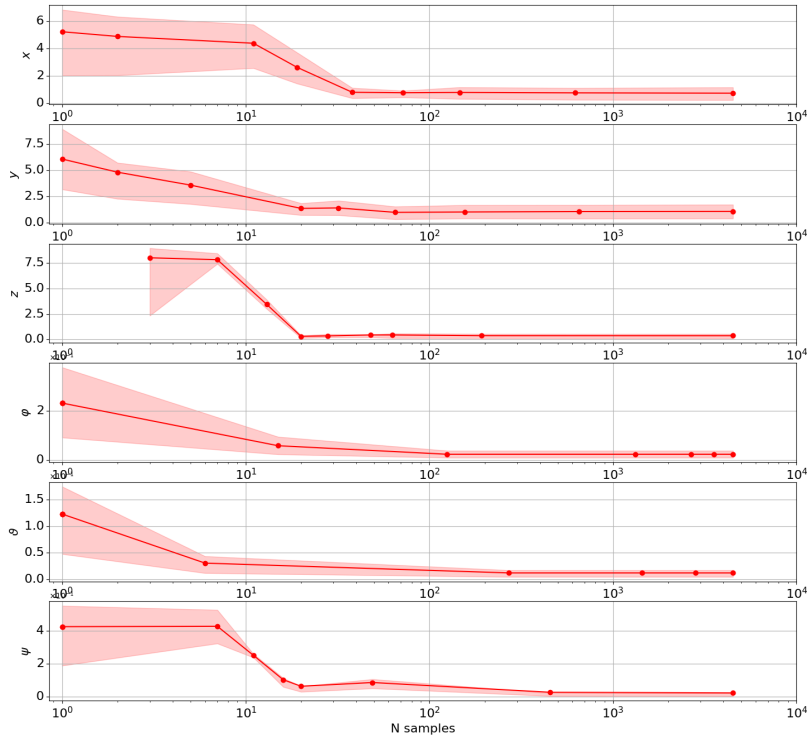


(b) Rollout evolution RMSE

Fig. 3.1: Error behavior for different train size dataset



(a) SOD - GP one step ahead prediction MSE



(b) SOD - Rollout evolution RMSE

Fig. 3.2: Error behavior for different sampled dataset

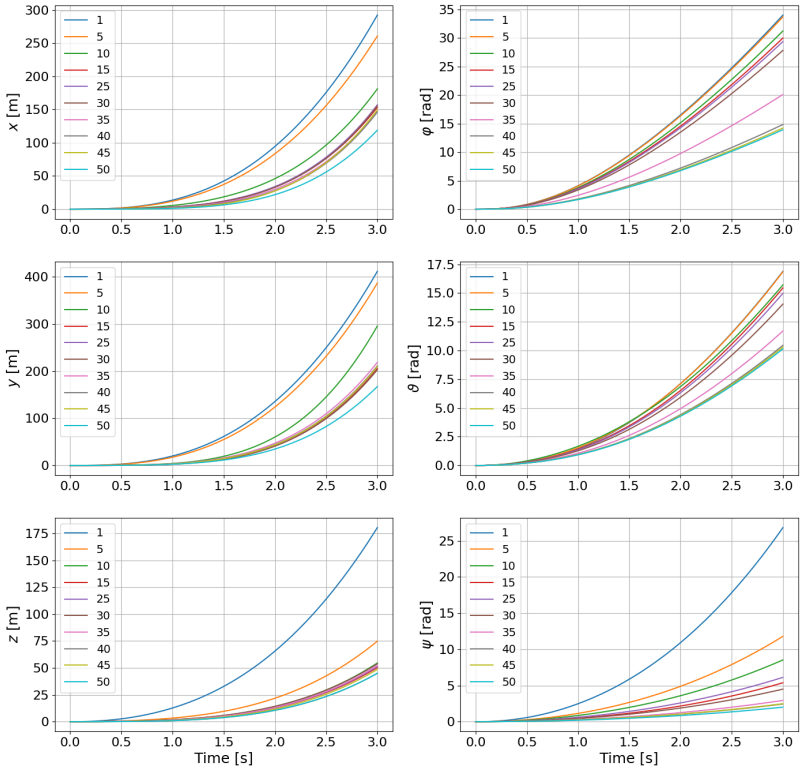


Fig. 3.3: Cumulative Error for different train size dataset

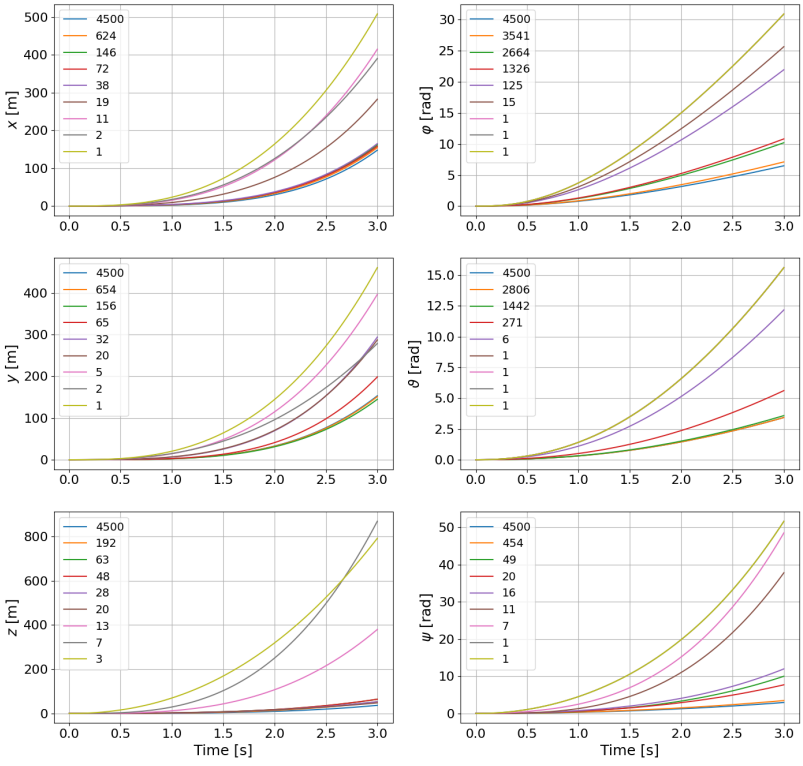


Fig. 3.4: Cumulative Error SOD for different sampled dataset

4

Quadrotor Control with MC-PILCO

In the previous section, we derived a nominal model for the quadrotor system, and we defined our model learning approach. Once the dynamic equation and effectiveness of predictions of the GPs trained model are validated, we can finally proceed to the Reinforcement Learning control strategy. In this section, we provide a broad view of what reinforcement learning is, and more specifically, we will set our Model-Based Reinforcement Learning (RL) domain. We will briefly resume the GPR model prediction goal in the context of our problem. This chapter will also discuss the control policy, its optimization, and the problem cost function.
We also present the learning algorithm results. This will be shown not only in an environment driven by de model defined in Chapter 2 but also resorting to the PyBullet physics simulator. The simulation uses an advanced and realistic quadrotor model, including flight dynamics and physical interaction considerations. This highly detailed simulation will give us an overview of the policy behavior interacting within an environment as close to reality.

4.1 Reinforcement Learning Framework

Reinforcement Learning (RL) is a specialized branch of machine learning aimed at designing an agent capable of making informed decisions within a particular environment. This environment is defined by a state space, denoted as $\boldsymbol{S}$, and the potential actions that the agent can execute are included within an action space $\boldsymbol{A}$. It's worth mentioning that the state and action spaces can be

either discrete or continuous, and this nature substantially affects the choice of RL methods employed.
The agent can observe the present state $s \in \boldsymbol{S}$ of the environment at a specific moment and respond with the appropriate action. This action consequently modifies the environment, leading to a transition into a new state, denoted as s'. The principal objective for the agent is to choose the most beneficial action based on its perception of the current state of the environment, intending to accomplish a task stipulated by the application. This task is guided by a reward function $\mathcal{R}(s, a)$ (or a cost function $C(s, a)$), which is dependent on the current state and the executed action. Consequently, the ultimate goal for the agent is to learn a policy $\pi(a|s)$ that maximizes (in the case of the reward function) or minimizes (in the case of the cost function) this function. Essentially, this policy can be understood as a mapping from states to actions, which could be either deterministic (a specific action is always taken given a state) or stochastic (actions are chosen based on specific probabilities). The policy thus determines the action to be taken based on the present observed state of the environment.
Note that we have introduced both the reward and cost functions. Using one rather than the other strongly depends on the application and the algorithm chosen. The algorithm selected for the quadrotor application relies on a cost function. Thus, from now, we will only consider a cost function as an action return.

In RL literature the agent-environment system is typically described by the *Markov Decision Process* (MDP) [23], defined as $\langle \boldsymbol{S}, \boldsymbol{A}, \mathcal{P}, C, \gamma \rangle$, where:

- $\boldsymbol{S}$: finite set of states;
- $\boldsymbol{A}$: finite set of action;
- $\mathcal{P}$: state transition probability function defined as

$$\mathcal{P}^{a}_{ss'} = \mathcal{P}\left[S_{t+1} = s' | S_t = s, A_t = a\right]; \tag{4.1}$$

- C: cost function $C^{a}_{s} = \mathbb{E}\left[C_{t+1} | S_t = s, A_t = a\right]$ returning the immediate cost of being in a specific state;
- γ: discount factor $\gamma \in [0, 1]$.

This formulation becomes particularly useful when the system is known, namely the state transition function is defined and when it is finite. Under these assump-

tions, we have the agent described by a policy function

$$\begin{aligned} \pi &: A \times S \rightarrow [0,1] \\ \pi(a|s) &= \mathcal{P}\left(a_t = a | s_t = s\right). \end{aligned} \tag{4.2}$$

The policy, then, defines the probability of the agent taking action a being the current state s. The final goal is to find a policy that minimizes the expected long-term cost

$$G_t = C_{t+1} + \gamma C_{t+2} + \gamma^2 C_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k C_{t+k+1}. \tag{4.3}$$

This value, for each possible state under the policy π, is defined by the value function

$$v_\pi(s) \doteq \mathbb{E}\left[G_t | S_t = s\right] = \sum_{a \in \boldsymbol{A}} \pi(a|s) \left(C_s^a + \gamma \sum_{s' \in \boldsymbol{S}} \boldsymbol{P}_{ss'}^a v_\pi(s') \right) \tag{4.4}$$

that is a direct derivation of the *Bellman Expectation Equation*. The value function is helpful if and only if the environment is perfectly known, meaning that the state transition probability must be well-defined. In real context, however, this scenario rarely happens. To overcome this condition is necessary to define the *action-value function*

$$q_\pi(s,a) \doteq \mathbb{E}_\pi\left[G_t | S_t = s, A_t = a\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k C_{t+k+1} | S_t = s, A_t = a \right]. \tag{4.5}$$

According to this new definition, the agent's goal becomes to find a policy that minimizes the action value function for every pair (s,a).

4.1.1 Limitation in Real Scenario

Traditional Reinforcement Learning (RL) techniques, such as those based on Markov Decision Processes (MDPs), perform well in contexts where the state and action spaces are relatively small and discrete. However, these techniques encounter significant difficulties when facing problems featuring very large or continuous state or action spaces. To begin with, the reason why these techniques face challenges in large spaces can be attributed to the so-called

"curse of dimensionality" [4]. Regarding RL, the curse of dimensionality refers to the phenomenon where the increase in the number of dimensions in the state or action space leads to an exponential increase in the number of possible state or action configurations. This makes computation difficult and learning an optimal policy challenging since the number of possible policies increases exponentially with the size of the state or action space.
Another related problem is the so-called exploration vs. exploitation problem [23]. In a large space, the agent must balance exploring new states or actions with exploiting the information it has already learned. This balance becomes increasingly difficult to manage as the size of the space increases since there are many more configurations to explore.
In our case, then, another issue arises. For quadrotor control, an RL algorithm must be applied to a problem with continuous state and action spaces, thus presenting further challenges. In a discrete space, the agent can easily enumerate and compare all possible actions in every state. However, the number of possible actions in each state is infinite in a continuous space. This makes the direct application of methods such as dynamic programming impossible, as it requires exhaustive exploration of all possible actions in each state. In a continuous space, therefore, the agent will need to learn an approximation function to represent the value function or the action-value function. This task can be challenging since the approximation function must accurately describe the value function or the action-value function at an infinite number of points in the space. This could lead to an excessively high data requirement. Advanced RL techniques have been developed to overcome these challenges, such as deep learning for approximating the value function or the action-value function and stochastic optimization methods for action selection in continuous spaces. These techniques show promising results in their ability to solve RL problems with large and continuous state or action spaces.

4.2 Model-Based Reinforcement Learning

So far, we have discussed what is referred to as Model-Free Reinforcement Learning. This approach seeks to learn an optimal policy or value function directly, relying solely on the agent's experience, which can be gathered through various techniques. Model-Free RL has its merits. It can be straightforward to implement and require low computational power during execution. However,

it tends to be data inefficient, as every decision is based on direct experience. As we have seen before, it will require an ever-increasing amount of data as the dimensionality of the environment grows. Moreover, exploration might be expensive or dangerous in real applications context. The exploration of the quadrotor environment might lead to risky maneuvers, potentially leading to damage to the system itself.

There is, however, another approach that aims to overcome this hurdle: Model-Based Reinforcement Learning (Model-Based Reinforcement Learning (MBLR)). In MBLR, the agent seeks to learn a model of the environment's dynamics. In other words, the agent tries to understand how its actions affect the state of the environment. Once the agent has learned a model of the environment, it can use this model to plan its actions. This planning can occur through various techniques, which we will explain for our problem later. The advantages of MBLR include the following:

- **Data efficiency**: MBLR can be much more data-efficient than Model-Free RL. Since the agent learns a model of the environment, it can use this model to simulate multiple possible future paths and choose the action that appears best according to the model. This means that the agent can effectively learn from fewer interactions with the environment.
- **Long-term planning**: Since the agent learns a model of the environment, it can use this model to plan its actions over a longer time horizon.
- **Adaptability**: If the environment changes, an agent using Model-Based RL can quickly update its model and adapt its behavior accordingly. This can be particularly useful, as in the case of a quadrotor, in non-stationary environments, where the dynamics of the environment can change over time.

This approach seems more suitable as it tends to overcome all the issues raised. However, we must also consider that this method has some disadvantages. Model-based planning can be computationally expensive, especially if the agent must view many possible future paths. Moreover, learning an accurate model of the environment can be a challenging task, particularly in complex environments with a large number of states and actions. As we have seen in the previous chapter, however, we have been able to develop a model capable of learning the dynamics of the quadrotor with a reasonable degree of precision, and by resort-

ing to approximation methods, we are also able to lighten the computational load during the action planning phase.

4.2.1 Model-Based Policy Gradient

Moving forward with our goal to devise an effective policy for the quadrotor control problem, we now turn our attention to a crucial tool within reinforcement learning that allows for the direct optimization of policies: Policy Gradient methods. Policy Gradient methods enable an agent to optimize policies directly without needing a value function. In its most general form, a policy map states to actions. The policy gradient provides a smooth and continuous optimization landscape that can greatly aid learning and exploration.
The specific reinforcement learning setting we are considering involves Probabilistic Inference for Learning COntrol (PILCO). In this setting, the policy gradient approach is enriched by a practical consideration of model uncertainty, which leads to improved learning efficiency and robustness of the policy, crucial factors for operating in dynamic environments such as that of a quadrotor.
The critical aspect of *gradient-based policy search* is that the policy is defined by some parameters θ optimized by gradient-descent over a cost function. To incorporate the model uncertainty on the policy optimization, let us consider our quadrotor dynamics given by the discrete-time system described by the unknown function $\boldsymbol{f}(\cdot,\cdot)$:

$$\boldsymbol{x}_{t+1} = \boldsymbol{f}\left(\boldsymbol{x}_t, \boldsymbol{u}_t\right) + \boldsymbol{w}_t. \tag{4.6}$$

In the last equation, $\boldsymbol{x}_t \in \mathbb{R}^{d_x}$ and $\boldsymbol{u}_t \in \mathbb{R}^{d_u}$ are respectively the state and control inputs of the system at each time step t, while $\boldsymbol{w}_t \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma}_w)$ is the model additive noise shaped as an independent Gaussian random variable. The cost function formulated in the previous section $c(\boldsymbol{x}_t)$ returns the immediate cost, or penalty, for lying in state $\boldsymbol{x}_t$. On the other hand, inputs will be chosen from a policy $\pi_\theta : \boldsymbol{x} \to u$ according to the parameters θ. The final goal is to derive a policy that minimizes the expected cumulative cost over a finite number of time steps T, namely,

$$J(\theta) = \sum_{t=0}^{T} \mathbb{E}_{\boldsymbol{x}_t}\left[c(\boldsymbol{x}_t)\right] \tag{4.7}$$

starting from the initial state x_0 selected according a given probability $p(x_0)$. To learn the policy model-based approach typically repeats in succession several trials attempting to resolve the required task. Each trial is built in three main phases:

1. *Model Learning*: the data collected from all the previous interactions are used to build a model of the system dynamics;
2. *Policy Update*: the policy is optimized attempting to minimize the cost $J(\theta)$ according to the current model;
3. *Policy Execution*: the current optimized policy is applied to the system, and the data are stored for model improvement in the next trials.

This method will rely on the learned model to predict the state evolution when the current policy is applied. These predictions will be then used to estimate $J(\theta)$ and its gradient $\nabla_\theta J(\theta)$ to update the policy parameters θ following the gradient-descent approach.

4.2.2 GPR and One-Step-Ahead Prediction

In our exploration of Gaussian Process Regression (GPR)-based methodologies, as previously detailed in section 3.3.3, a frequent tactic entails the individual modeling of each state dimension utilizing a distinct Gaussian Process (GP). Let us consider the i-th state's value at times $t+1$ and t. The differential $\Delta_t^{(i)}$ between these two temporal instances can be defined mathematically as

$$\Delta_t^{(i)} = x_{t+1}^{(i)} - x_t^{(i)}, \quad i \in \{1, \dots, d_x\}, \tag{4.8}$$

Moreover, we can introduce an element of observational noise into our model by defining $y_t^{(i)}$ as

$$y_t^{(i)} = \Delta_t^{(i)} + e^{(i)} \tag{4.9}$$

In this equation, $e^{(i)}$ serves the function of modeling additive noise. By incorporating this factor, we simulate the noise that is inevitably encountered in real-world observational data, adding robustness to our model. Again, we consider as GP input the combination of the system state and its input at time t

$$\tilde{\boldsymbol{x}}_t = \begin{bmatrix} \boldsymbol{x}_t \\ \boldsymbol{u}_t \end{bmatrix}. \tag{4.10}$$

Then, given the dataset

$$\begin{aligned} \mathcal{D} &= \left(\tilde{X}, \boldsymbol{y}^{(i)}\right) \\ \boldsymbol{y}^{(i)} &= \left[y_{t_1}^{(i)}, \dots, y_{t_n}^{(i)}\right]^T, \\ \tilde{X} &= [\tilde{x}_{t_1}, \dots, \tilde{x}t_n] \end{aligned} \tag{4.11}$$

with $\boldsymbol{y}^{(i)}$ a sequence of n noisy measurement, and $\tilde{X}$ is the set of corresponding GP inputs. We can define the probabilistic model assumed by GPR for each state

$$\boldsymbol{y}^{(i)} = \begin{bmatrix} h^{(i)}(\tilde{x}_{t_1}) \\ \vdots \\ h^{(i)}(\tilde{x}_{t_n}) \end{bmatrix} + \begin{bmatrix} e_{t_1}^{(i)} \\ \vdots \\ e_{t_n}^{(i)} \end{bmatrix} = \boldsymbol{h}^{(i)}(\tilde{X}) + \boldsymbol{e}^{(i)} \tag{4.12}$$

where $\boldsymbol{e}^{(i)} \sim \mathcal{N}(0, \sigma_i I)$ is the additive noise modeled as a zero-mean Gaussian and is the unknown function

$$h^{(i)} : \tilde{x} \to \Delta^{(i)}, \quad i \in \{1, \dots, d_x\} \tag{4.13}$$

that describes the system dynamics, modeled as *a priori* zero-mean GP.

According to the formalism described in chapter 3 the GP $h^{(i)}$ can be shaped as

$$h^{(i)} \sim \mathcal{N}\left(\mathbf{0}, K_i(\tilde{X}, \tilde{X})\right) \tag{4.14}$$

where the matrix $K_i(\tilde{X}, \tilde{X})$ is the kernel function whose choice has been discussed in section 3.3.1. From equations (3.20) and (3.21), we can now compute in closed form the posterior distribution of $h^{(i)}$. Given a general GP input $\tilde{\boldsymbol{x}}_t \notin \tilde{X}$ the distribution of $\hat{\Delta}_t^{(i)}$ and the estimate of $\Delta_t^{(i)}$ are Gaussian with mean and variance

$$\begin{aligned} \mathbb{E}[\hat{\Delta}_t^{(i)}] &= \mathbb{E}[\hat{h}_t^{(i)}(\tilde{\boldsymbol{x}}_t)] = k_i(\tilde{\boldsymbol{x}}_t, \tilde{X})\Gamma_i^{-1}\boldsymbol{y}^{(i)} \\ \mathbb{V}[\hat{\Delta}_t^{(i)}] &= \mathbb{V}[\hat{h}_t^{(i)}(\tilde{\boldsymbol{x}}_t)] = k_i(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_t) - k_i(\tilde{\boldsymbol{x}}_t, \tilde{X})\Gamma_i^{-1}k_i(\tilde{\boldsymbol{x}}_t, \tilde{X})^T \end{aligned} \tag{4.15}$$

with Γ_i and $k_i(\tilde{\boldsymbol{x}}_t, \tilde{X})$

$$\begin{aligned} \Gamma_i &= (K_i(\tilde{X}, \tilde{X}) + \sigma_i^2 I) \\ k_i(\tilde{\boldsymbol{x}}_t, \tilde{X}) &= [k_i(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_{t_1}), \dots, k_i(\tilde{\boldsymbol{x}}_t, \tilde{\boldsymbol{x}}_{t_n})]^\cdot \end{aligned} \tag{4.16}$$

In conclusion, modeling each state with a distinct GP, the posterior distribution for the estimated state at time $t+1$ is

$$p(\hat{\boldsymbol{x}}_{t+1}|\tilde{\boldsymbol{x}}_t, \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\mu}_{t+1}, \Sigma_{t+1}) \tag{4.17}$$

where

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \boldsymbol{x}_t + \left[\mathbb{E}[\hat{\Delta}_t^{(1)}], \ldots, \mathbb{E}[\hat{\Delta}_t^{(d_x)}]\right]^T \\ \Sigma_{t+1} &= diag\left(\left[\mathbb{E}[\hat{\Delta}_t^{(1)}], \ldots, \mathbb{E}[\hat{\Delta}_t^{(d_x)}]\right]\right) \end{aligned}. \tag{4.18}$$

4.2.3 Long-Term Prediction with GP Dynamical Models

The improvement of the parameterized policy π_θ is based on the long-term prediction of the evolution $p(\hat{x}_1), \ldots, (\hat{x}_T)$. The exact computation requires applying the one-step-ahead prediction in cascade, propagating uncertainty. This means, starting from an initial distribution $p(\boldsymbol{x}_0)$, at each time steps t obtaining the next state distribution by marginalizing over $p(\hat{\boldsymbol{x}}_t)$, namely

$$p(\hat{\boldsymbol{x}}_{t+1}) = \int p(\hat{\boldsymbol{x}}_{t+1}|\hat{\boldsymbol{x}}_t, \pi_\theta(\hat{\boldsymbol{x}}_t), \mathcal{D}) d\hat{\boldsymbol{x}}_t. \tag{4.19}$$

This exact predicted distribution computation, in practical terms, is not feasible. Thus we will apply a different approach to solve this problem. The original PILCO presentation [10] aims to approximate the solution with *Moment Matching*. In Moment Matching, we assume the GP models use only the SE kernel as a prior covariance and considering a normal initial state distribution $\boldsymbol{x}_0 \sim N(\boldsymbol{\mu}_0, \Sigma_0)$, the first and second moments of $p(\hat{\boldsymbol{x}}_1)$ can be computed in the closed form [9]. Consequently, the distribution $p(\hat{\boldsymbol{x}}_1)$ can be approximated as a Gaussian distribution, with its mean and variance corresponding to the previously computed moments. Subsequently, the probability distributions for the following time steps of the prediction horizon are iteratively computed using the same procedure. For detailed information about the computation of the first and second moments, refer to [9]. Moment matching offers the advantage of providing a closed-form solution for handling uncertainty propagation through the GP dynamics model. Therefore, it is possible to compute the policy gradient based on long-term predictions analytically in this context. However, the Gaussian approximation employed in moment matching also entails two main

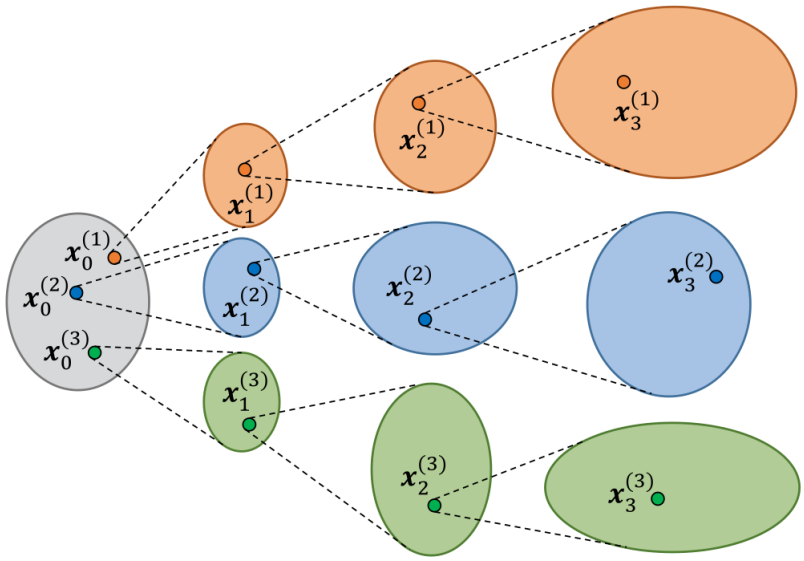


Fig. 4.1: Particles propagating thought the stochastic model [2]

weaknesses:

1. The computation of the two moments assumes the use of SE kernels;
2. This method allows for modeling only unimodal distributions, which might be an overly restrictive approximation of the true system behavior.

Due to this limitation for this work, we apply a different method. The integral in (4.19) can be approximated by relying on Monte Carlo, particularly particle-based methods. Specifically, M particles are sampled from the initial state distribution $p(\boldsymbol{x}_0)$. Each one of the M particles is propagated using the one-step-ahead GP models (4.17). Let $x_t^{(m)}$ be the state of the mth particle at time t, with $m = 1, ..., M$. At time step t, the actual policy π_θ is evaluated to compute the associated control. The GP model provides the Gaussian distribution $p(\boldsymbol{x}_{t+1}^{(m)}|\boldsymbol{x}_t^{(m)}, \pi_\theta(\boldsymbol{x}_t^{(m)}), D)$ from which $\boldsymbol{x}_{t+1}^{(m)}$, the state of the particle at the next time step, is sampled. This process is iterated until each particle's trajectory of length T is generated. The overall process is shown in Figure 4.1. The long-term distribution at each time step is approximated with the distribution of the particles.

Note that this approach does not impose any constraint on the choice of the kernel function and the initial state distribution. Moreover, there are no restrictions on the distribution of $p(\hat{\boldsymbol{x}}_t)$. Therefore, particle-based methods do not suffer

from the problems seen in moment matching at the cost of being more computationally heavy. Specifically, the computation of (4.17) entails the computation of (4.15), which are the mean and the variance of the delta states, respectively.
Regarding the computational complexity, it can be noted that $\Gamma_i^{-1}\boldsymbol{y}^{(i)}$ computed a single-time offline during the training of the GP model (the same computation is needed at the moment matching case), and the number of operations required to compute (4.15).a is linear w.r.t. the number of samples n. The computational bottleneck is the computation of (4.15).b, which is $O(n^2)$. Then, the cost of a single state prediction is $O(d_x n^2)$, leading to a total computational cost of $O(d_x MTn^2)$. Depending on the complexity of the system dynamics, the number of particles necessary to obtain a good approximation might be high, determining a considerable computational burden.
Nevertheless, the computational burden can be substantially mitigated via graphics processing units (GPU) parallel computing due to the possibility of computing the evolution of each particle in parallel. Notice that we are estimating the full state assuming the model predicts each component with distinct and independent GP. However, from section 3.3.3, we have seen that our model, like many physical systems, is described by its pose and respective velocities. This allows us to employ the speed integration model, limiting estimating only the velocity GP and integrating the result to obtain the pose components. This will decrease the cost of a state prediction to $O(\frac{d_x}{2}MTn^2)$. Nevertheless, this approach is based on a constant acceleration assumption and works appropriately only considering small enough sampling times.

4.3 Policy Optimization

Monte Carlo Particle-Based Method and speed-integration GPR offer a reliable strategy for achieving credible long-term state predictions. Such long-term predictions allow for policy optimization that takes into account the entire behavior horizon. This perspective of policy optimization fosters a more insightful policy capable of mitigating issues related to inherent noise. The following sections will provide a detailed exposition of this policy optimization approach.

4.3.1 Policy Structure

In MC-PILCO, we can optimize any differentiable policy function. Neural Networks are an example of a function. However, we will resort to a more interpretable function for our purpose. The function is the so-called *squashed RBF network* policy, defined as

$$\pi_\theta(\boldsymbol{x}) = u_{max} \tanh\left(\frac{1}{u_{max}} \sum_{i=1}^{n_b} w_i e^{||\boldsymbol{a}_i - \boldsymbol{x}||^2_{\Sigma_\pi}}\right). \tag{4.20}$$

Clearly, the policy parameters are $\boldsymbol{\theta} = \{\boldsymbol{w}, \boldsymbol{A}, \Sigma_\pi\}$, where $w = \left[w_1, \ldots, w_{n_b}\right]$ are the weights of the Gaussian basis function, while $A = \left[a_1, \ldots, a_{n_b}\right]$ and Σ_π, assumed diagonal, determines the center and the shape of theses functions respectively. Again the elements of the Gaussian shape are called lengthscales. In the function, the hyperbolic tangent limits the basis function between $[-1, 1]$, and the returned values are then rescaled of a factor u_{max}. This means that our policy output will be in the interval $[-u_{max}, u_{max}]$ that, in our context, can be seen as the maximum motor speed offset allowed.

4.3.2 Policy Gradient Computation

The success of policy optimization is contingent upon the cumulative cost function of long-term prediction, denoted as $\mathbb{E}_{\boldsymbol{x}_t}$. However, accurately estimating the trajectory of state evolution necessitates the utilization of the Monte Carlo particle method. With reference to the procedures detailed in Section 4.2.3, we define $\boldsymbol{x}_t^{(m)}$ as the state of the m-th particle, with m ranging from 1 to M and t between 0 and T. In this setting, the cumulative cost is estimated through the Monte Carlo method as follows:

$$\hat{J}(\theta) = \sum_{t=0}^{T} \left(\frac{1}{M} \sum_{m=1}^{M} c(\boldsymbol{x}_t^{(m)})\right), \tag{4.21}$$

The state evolution of each particle, $\boldsymbol{x}t^{(m)}$, is a sample drawn from the normal distribution $p(\boldsymbol{x}t + 1^{(m)} | \boldsymbol{x}t^{(m)}, \pi\theta(\boldsymbol{x}t^{(m)}), \mathcal{D}) \sim \mathcal{N}(\boldsymbol{\mu}t + 1, \Sigma_{t+1})$, as given by equation (4.18).

The intrinsic stochastic components of our model complicate the direct computation of gradients with respect to the policy parameters. However, applying

the so-called "reparameterization trick" [2] offers an elegant solution, providing a pathway to differentiate through the stochastic processes.

In this approach, rather than sampling directly from a normally distributed variable $N(\boldsymbol{\mu}t+1, \Sigma t+1)$ during the gradient computation, we sample a point ξ from a standard normal distributioncharacterized by a zero mean and unit varianceand of dimensions identical to $\boldsymbol{\mu}t+1$. This sample point is subsequently mapped into our distribution of interest, yielding $x^{(m)}t+1 = \boldsymbol{\mu}t+1+Lt+1\xi$, where L_{t+1} signifies the Cholesky decomposition of Σ_{t+1}, i.e., $\Sigma_{t+1} = L_{t+1}L_{t+1}^T$. By applying this reparameterization trick, we can create a deterministic relationship between $x^{(m)}t+1$ and θ, allowing the computation of $\nabla\theta\hat{J}$ using backpropagation. The policy parameters θ are then updated via the Adam optimizer, with α_{lr} serving as the step size within the framework of Adam [17].

Dropout The gradient-based optimization procedure we employ is not without its challenges, with the occurrence of local minima standing out as a significant hurdle. These cost function nadirs can trap the optimization process in sub-optimal configurations. Optimizers such as Adam introduce noise reduction techniques to provide more accurate gradient estimations, which aid in escaping these sub-optimal traps. However, even these stochastic gradient descent methods cannot guarantee arrival at the global minimum.

This work adopts the "dropout" technique presented in [2]. Dropout, applied to the policy (4.20), involves the random elimination of weights $\boldsymbol{w}$ with probability p_d. It is accomplished by scaling all weights w_i by a random variable $r_i \sim Bernoulli(1-p_d)$. This operation, in essence, constructs a distribution over the weights, thus creating a parameterized stochastic policy π_θ. Following this process, the stochastic policy employed during policy optimization elevates the entropy of the particle distribution, aiding the system in exploring areas associated with lower costs and thereby sidestepping local minima. The utility of dropout also extends to mitigating issues related to exploding gradients, possibly because the gradient is evaluated across a variety of $\boldsymbol{w}$ values. Each different $\boldsymbol{w}$ value yields a different policy, and by averaging the gradients over slightly diverging policies, a regularization effect is achieved.
However, it's important to note that while dropout introduces beneficial entropy into the policy, it may, in doing so, negatively affect the precision of the resulting solution. Consequently, it's crucial to manage dropout in such a way that the policy optimization phase yields a deterministic policy, thereby ensuring the

accuracy of the solution.
Following this reasoning, a heuristic scaling procedure has been proposed [2] that progressively diminishes the dropout rate, or p_d, to zero during the policy update iterations.

Dropout Scaling The scaling procedure is guided by a monitoring signal s. This signal is derived from the statistics of past iterations of $\hat{J}$. Specifically, the cost change $\Delta\hat{J}j = \hat{J}(\theta_j) - \hat{J}(\theta j - 1)$ at the j-th optimization step is defined. The monitoring signal s is then computed as a filtered version of the ratio between the mean and the standard deviation of $\Delta\hat{J}_j$, calculated using the Exponential Moving Average (EMA) filter. At the j-th optimization step, the following conditions are met:

$$\mathcal{E}[\Delta\hat{J}j] = \alpha_s\mathcal{E}[\Delta\hat{J}j - 1] + (1 - \alpha_s)\Delta\hat{J}j, \tag{4.22}$$

$$\mathscr{V}[\Delta\hat{J}j] = \alpha_s((\mathscr{V}[\Delta\hat{J}j - 1]) + (1 - \alpha_s)(\Delta\hat{J}j - \mathcal{E}[\Delta\hat{J}j - 1])^2), \tag{4.23}$$

$$s_j = \alpha_s sj - 1 + (1 - \alpha_s)\frac{\mathcal{E}[\Delta\hat{J}_j]}{\sqrt{\mathscr{V}[\Delta\hat{J}_j]}}, \tag{4.24}$$

where α_s is the coefficient of the EMA filter, which governs its memory capacity. For each iteration of the optimization procedure, the algorithm examines whether the absolute value of the monitoring signal s over the last n_s iterations is beneath the threshold σ_s. Specifically, the algorithm checks the condition:

$$[|s_{j-n_s}| \cdots |s_j|] < \sigma_s \tag{4.25}$$

If this condition is satisfied, it suggests that the solution may fall into a local minimum, thus triggering a decrease in the dropout rate. The new value of p_d is then calculated as follows:

$$\begin{aligned} p_d &= \max(p_d - \Delta p_d, 0), \\ \alpha_{lr} &= \max(\lambda_s\alpha_{lr}, \alpha_{lr_{min}}) \\ \sigma_s &= \lambda_s\sigma_s \end{aligned} \tag{4.26}$$

where α_{p_d} is the learning rate for the dropout rate, and $p_{d_{min}}$ is the minimum achievable dropout rate. The algorithm continues to iterate, reducing p_d as long

as

$$p_d \geq 0 \text{and} \alpha_{lr} \geq \alpha_{lr_{min}} \tag{4.27}$$

In the above way, our proposed method is guided by intelligent dropout scaling, which helps prevent the solution from getting trapped in suboptimal local minima. The intelligent dropout scaling also leads to a regularization effect, reducing the chances of overfitting while improving the model's generalizability. The algorithm 2 reports the pseudo-code for the MC-PILCO with dropout.

Algorithm 2 PID Algorithm for z altitude

Require: Policy $\pi_\theta(\cdot)$, cost $c(\cdot)$, kernel $k(\cdot,\cdot)$, max optimization steps N_{opt}, particles number M, learning rate α_{lr}, min learning rate $\alpha_{lr_{min}}$, dropout probability p_d, dropout probability reduction Δp_d and monitoring parameters: σ_s, λ_s, n_s.

```
Apply initial control policy to the system and collect data
while task not successfully learned do
  (1) Model Learning:
  GP models are learned from sampled data - section 3.3.3
  (2) Policy Update:
  s_0 ← 0
  for j = 1 to Nopt do
    Simulate M particles rollouts with current GP models and π_θ
    Compute J(θ_j) from eq. (4.21)
    Compute ∇J(θ)
    π_{θ_{j+1}} = gradient-based update
    Update s_j with eq. (1.45)
    if (1.46) then
      Update pd, α_lr and σ_s with eq. (4.22), (4.24)
    end if
    if not (1.48) then
      Exit for loop
    end if
  end for
  (3) Policy Execution:
  Apply the updated policy to the system and collect data
end while
return trained policy, learned GP model
```

In this work, we adapted the MC-PILCO implementation developed by [2] in Python and publicly available at `https://www.merl.com/research/license/MC-PILCO`.

4.3.3 Cost Function

The definition of the cost function $c(\mathbf{x}_t)$, representing the penalty for occupying a particular state, is a crucial element in our reinforcement learning (RL) framework. The reason is that this function encapsulates our objective - penalizing the quadrotor for deviating from the desired state. In the current study, we define the cost function as a saturating function expressed as the negative exponential of the squared norm of the difference between the current and desired state, i.e.,

$$c(\mathbf{x}_t) = 1 - \exp\left(-(\boldsymbol{x}_t - \boldsymbol{x}^{des})^T L(\boldsymbol{x}_t - \boldsymbol{x}^{des})\right). \tag{4.28}$$

Here, $\mathbf{x}^{des}$ represents the desired state. The matrix L is a diagonal matrix that allows for different weights for different components of $\boldsymbol{x}_t - \boldsymbol{x}^{des}$, depending on the variations in their ranges or their relative importance. Our primary objective is to find a policy capable of controlling the propeller motor's rotational speed, thus reducing the problem's complexity by focusing on the desired position in the xyz space. This is based on the understanding that stability at a specific location requires a null velocity and no inclination w.r.t. the xy plane. This simplification translates into a cost function based on the first three state parameters:

$$c(\mathbf{x}_t) = 1 - \exp\left(-\left(\frac{x_t - x^*}{l_x}\right)^2 - \left(\frac{y_t - y^*}{l_y}\right)^2 - \left(\frac{z_t - z^*}{l_z}\right)^2\right). \tag{4.29}$$

Here, l_x, l_y, and l_z represent characteristic lengthscales in the x, y, and z dimensions, respectively, which provide a normalized measure of the deviation from the desired position.

5

MC-PILCO Application Results

In Chapter 4, we introduced our Reinforcement Learning approach, enlightening how it works and all the needs to reach our quadrotor position tracking objective. This section will finally apply this framework to our problem and observe its results. We initially want to observe if the learned policy can move the quadrotor to the desired position. We will then focus on seeing if it is capable of reaching that point sable and how the system behaves under the policy control with respect to a traditional PID controller.
Policy optimization has been performed on a simulation moved by our nominal differential orders equation. The second step was to test this performance in a more realistic environment. This has been done by exploiting the PyBullet physics simulator for Python. Combined with the Gym package, we used a quadrotor simulation that added to the dynamics of external disturbances such as ground effects, drag forces, and downwash.

5.1 Nominal ODE Simulation

We started by developing a model of the quadrotor that describes its state evolution by ODEs, as indicated in equation (2.54). In Section 2.6, we determined the base velocity that allows the quadrotor to maintain a certain altitude and, ideally, position. For simplicity, we assume the quadrotor starts at an altitude of $2m$ from the ground, with an initial position of $\boldsymbol{x}_0 = [0,\ 0,\ 2,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0]^T$. This higher altitude helps us avoid problems related to take-off and landing, and also reduces the risk of collisions.

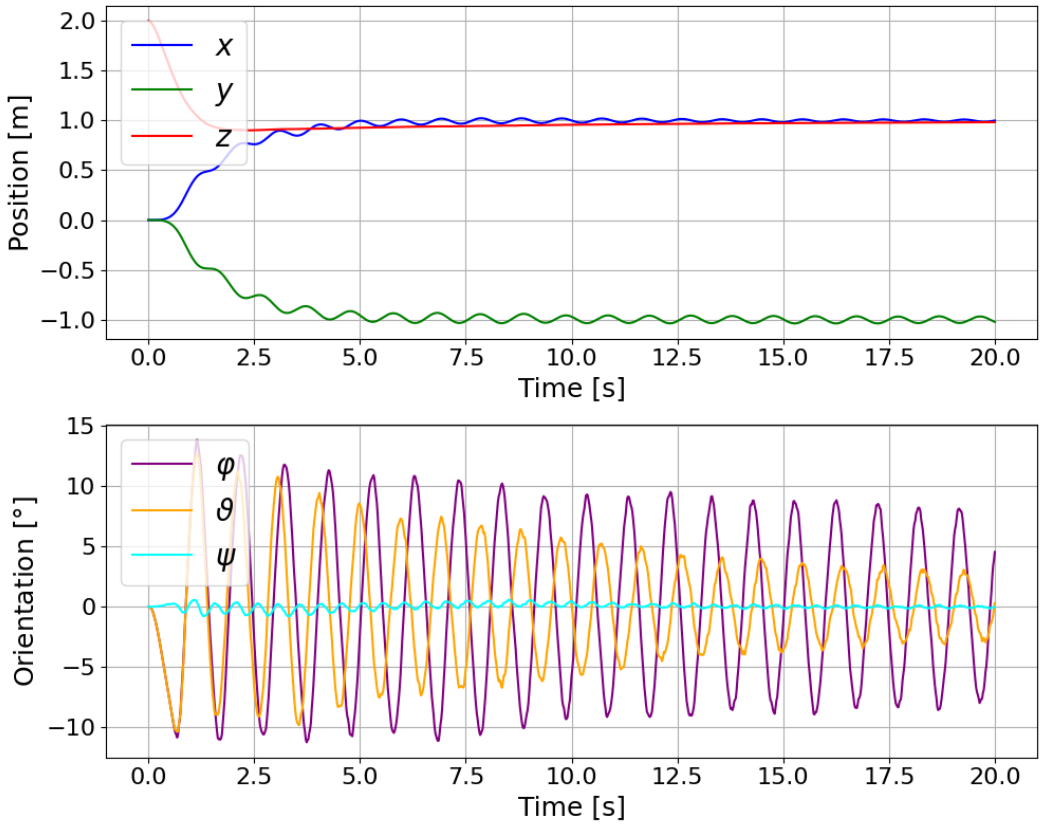


Fig. 5.1: PID Position ODE Control for Target Position $\boldsymbol{x}^{des}$

Our goal now is to move the quadrotor from the initial position $\boldsymbol{x}_0$ to the desired position $\boldsymbol{x}^{des} = [1, -1, 1]^T$, which represents a $1m$ movement in all three directions. Because of the quadrotor's unique structure, we don't need to specify the components of the other states. Indeed, maintaining a desired position means the quadrotor's orientation must remain parallel to the xy plane, and, similarly, all velocities should be zero. In other words, imposing a target position introduces restrictions on other dimensions. The only thing that can change freely is the orientation ψ, but for now, we are ignoring this as it doesn't affect the final goal.

From the PID for position tracking described by (2.60), we have developed a controller that enables the quadrotor to move to the target position. However, reaching the goal results in a slow and nearly unstable operation. Figure 5.1 reports the controller's behavior.

The control of the z altitude exhibits good performance, with the quadrotor getting close to the target within about $2s$. On the other hand, reaching the desired positions on x and y axes involves a critical state evolution. The primary

issue is the oscillatory behavior, which requires several seconds to fade away, consequently delaying the rise time - it takes about $5s$ to reach the target positions, with oscillations still present after $20s$. In our nominal environment, we observe that the quadrotor can handle this maneuver, where the only disturbance is the white additive noise considered in the states' observations. However, in a real-world context, additional external disturbances may introduce numerous other instabilities and uncertainties, potentially leading to disastrous behavior.

5.1.1 Cost Shape

In the last section of the previous chapter, we introduced the cost function, which will be responsible for defining our final goal. According to the target desired, our function is

$$c(\mathbf{x}_t) = 1 - \exp\left(-\left(\frac{x_t - (1)}{l_x}\right)^2 - \left(\frac{y_t - (-1)}{l_y}\right)^2 - \left(\frac{z_t - (1)}{l_z}\right)^2\right). \tag{5.1}$$

The lengthscales l_x, l_y, and l_z are crucial parameters in our cost function. Their role is to provide a normalized measure of the deviation from the desired position in each dimension, effectively scaling the contribution of each state component to the cost. The choice of these lengthscales is not trivial and can significantly influence the controller's performance. In this work, we have chosen all three lengthscales to be equal to 3. The reason for this choice is twofold. First, this value provides a sufficiently smooth curve around the desired point, meaning that minor deviations from the target position will not result in significant changes in the cost. This is desirable as it allows the quadrotor to adjust its position gradually without being heavily penalized for minor errors. Second, this choice helps avoid the risk of infinite gradients, which could lead to unstable or unpredictable behavior of the controller.

5.1.2 Policy Optimization Trials and Results

Exploration Policy In the previous section, we discussed the alternation of exploration and exploitation. The latter corresponds to the target policy, the rule set our agent adheres to achieve the given objective. In contrast, the exploration policy serves a different purpose altogether.
Typically, in reinforcement learning (RL) algorithms, the agent refines its policy

by leveraging observations from the environment. However, there can be instances where it might be more advantageous for the agent to diverge from its learned policy. The goal behind this deviation is to foster an environment for exploration, encouraging the agent to test diverse actions in a range of states, thereby expanding its understanding of the environment.
This exploration phase can be likened to a stage of data collection or information gathering. Through exploration, the agent can uncover the various rewards linked with different state-action pairs, facilitating a more profound comprehension of the inherent cost structure of the environment. Not only does exploration aid in reward discovery, but it also stimulates the agent to venture into uncharted states and transitions. This could potentially lead to lower costs, thereby contributing to the development of a more sophisticated policy.
In our Model-Based RL setting, exploration takes on an even more pivotal role. During the initial iteration, the agent has neither knowledge nor data about the environment, rendering it incapable of learning a model for its long-term prediction. Hence, it is necessary to execute some exploration actions to gather preliminary data. Given our objective to evaluate whether our algorithm can enhance the performance of existing controllers, we employ the PID position controller we designed as our exploration policy. The evolution of this exploration policy is depicted in Figure 5.1.

Parameter	**Description**	**Value**
Δ_{p_d}	pd reduction term	0.125
α_{lr}	optimizer's learning rate	0.01
$\alpha_{lr_{min}}$	minimum learning rate	0.0025
α_s	EMA filter memory	0.99
σ_s	threshold of monitoring signal	0.08
n_s	number of monitored iterations	200
Λ_s	reduction coefficient of σ_s and α_{lr}	0.5
p_d	dropout rate	0.25

Table 5.1: Summary of parameters

Optimization Trials and Results The parameters listed in Table 5.1 are those used during the policy optimization process. The entire process is carried out over a total of 5 trials, in each of which we expect an increase in performance or, more generally, an improvement in the result.
In this context, the time horizon for the exploration trajectory is set to $20s$, while

the horizon for policy optimization is $5s$. The reasoning behind this choice lies in the desire to provide a large volume of data for the initial learning phase, hence a longer exploration time horizon.
Our primary objective is to verify the algorithm's ability to accurately learn the model. In light of the results set out in Section 3.4, it is reasonable to expect an initial non-optimal estimate. However, it is anticipated that as the trials progress, there will be an improvement in performance, hopefully culminating in satisfactory results. This trend is corroborated by the data from the various trials. To better illustrate this point, Figure 5.3 shows the performance of the first Gaussian Process (GP) in reference to $\Delta\dot{x}$. As can be seen, the model demonstrates the ability to learn the dynamics from the first trial, albeit with a high level of uncertainty. Over time, this uncertainty tends to decrease. In fact, the latest trials show that the model not only provides precise estimates but is also capable of effectively generalizing the additive error introduced in the observations.

The importance of a well-trained model lies in its ability to maintain a strong correlation between the particles, which represent the long-term state estimates, and the actual evolution of the system. Figure 5.4 illustrates the state estimates obtained through the particles for each trial.
Another indicator of the effectiveness of policy optimization can be observed by analyzing the progression of cost. Figure 5.2 shows encouraging trends in this regard. In the first trial, the total cost decreases only to a certain point. However, a higher plateau in this phase of improvement can be explained by the fact that we are only at the first trial. At this stage, the model is not yet capable of making accurate predictions, and the agent has not had the opportunity to accumulate enough experience from interactions with the environment to learn an optimal policy. This circumstance is also evident in the evolution of the particles, where it is clear that the agent, in the first trial, is not yet able to exert effective control. Consequently, the cost of the particles will go to 1, the highest possible value.

With the advancement to the second trial, where the model has had the opportunity to incorporate data from the first application of the policy to the system, tangible progress is manifested. It is possible to note, in fact, a decrease in the uncertainty associated with the Gaussian Process (GP) and a simultaneous convergence of the control of particles toward the predefined objective. This positive evolution finds further confirmation in the cost progression, which, at this stage, begins to converge toward optimal values.

However, the most significant evaluation metric is the behavior of the policy when applied to the system described by ordinary differential equations (ODE). This information is illustrated in Figure 5.5, where the pose state and cost behavior are shown. The first trajectory presented is the one generated by the PID controller for exploration. As can be easily observed, the trajectory is almost unstable, and the controller seems unable to reach the pre-set target within a wide time horizon of $20s$. The following images show the results obtained from the learned policy. From the first trial, the response appears much smoother, thus eliminating all the oscillatory issues previously highlighted, although it cannot reach the pre-set target. This may be due to the model's inability, up to this point in the process, to converge with the evolution of the particles.
However, starting from the second trial, the results begin to become significant. With the increase in the precision of the model and good convergence of the particles, the actual system rollout shows that the optimized policy can reach the target in a stable manner, with optimal movements and significantly reduced response times, approximately $2s$ rise time.
In the subsequent trials, the refinement of the policy may not appear as evident. However, observing the rollout on the real system, it will be possible to notice a gradual optimization of the rise times, stability (with reduced overshoots), and precision in reaching the target.

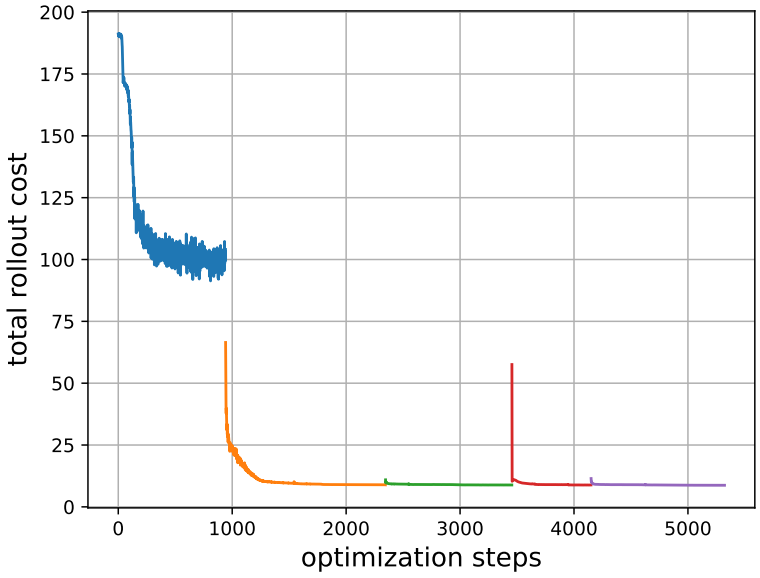
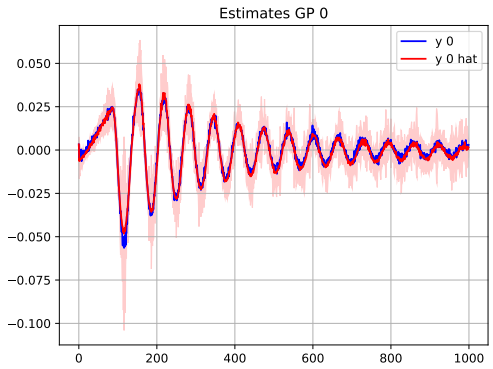
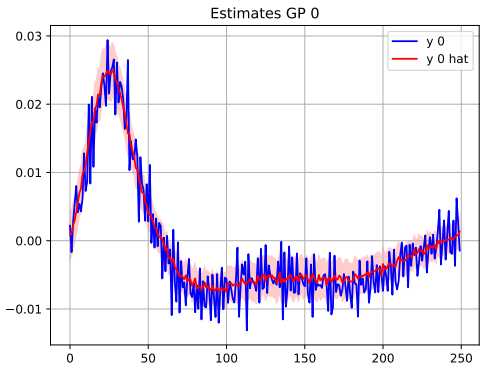


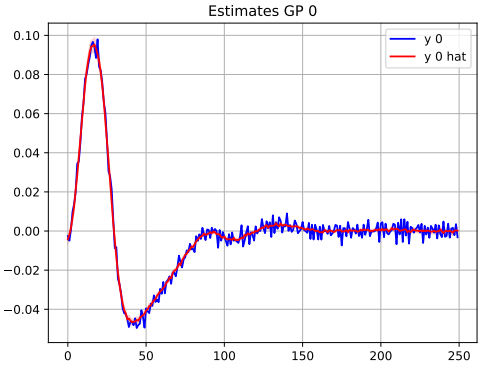
Fig. 5.2: Learning Plot



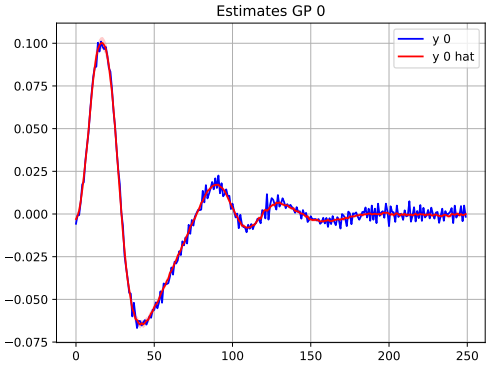
(a) Exploration



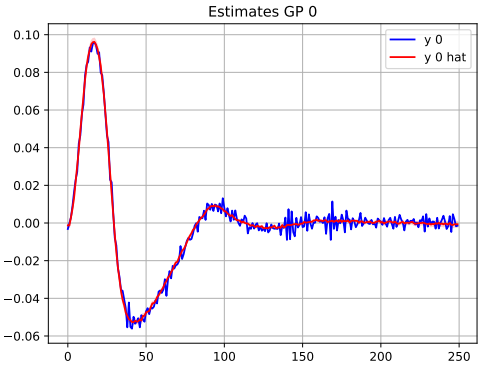
(b) Trial 1



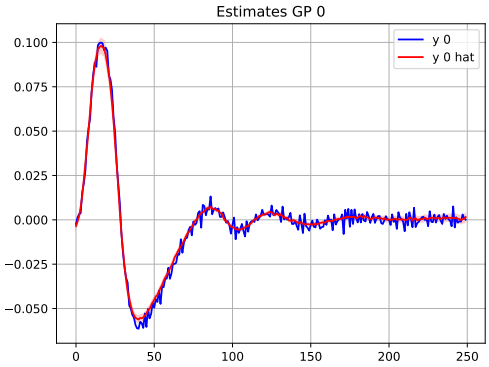
(c) Trial 2



(d) Trial 3

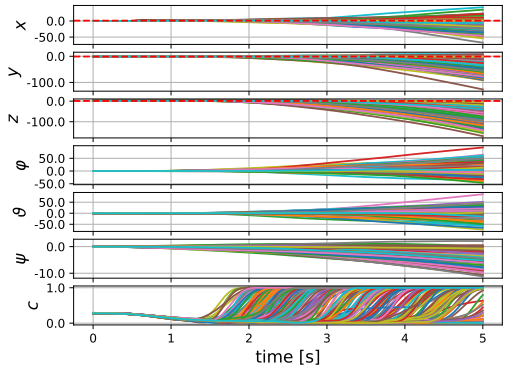


(e) Trial 4

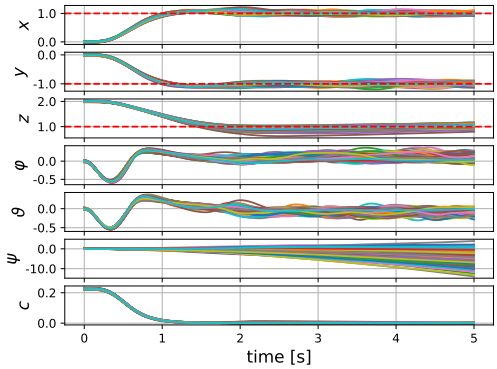


(f) Trial 5

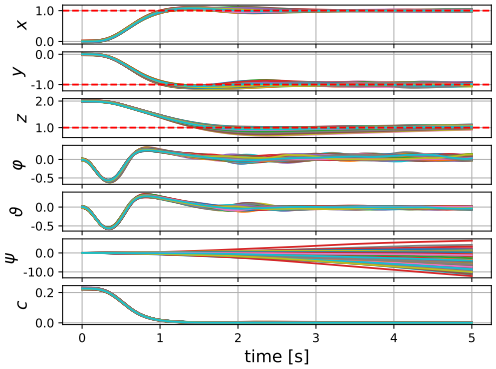
Fig. 5.3: GP Estimation Trials Improvement



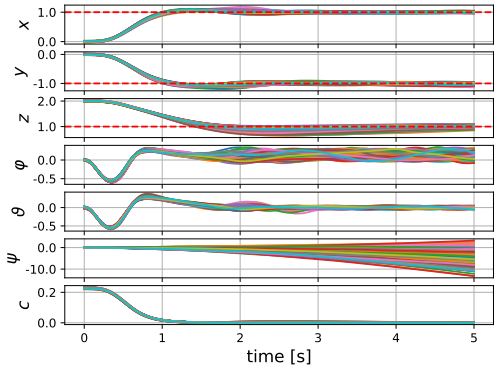
(a) Trial 1



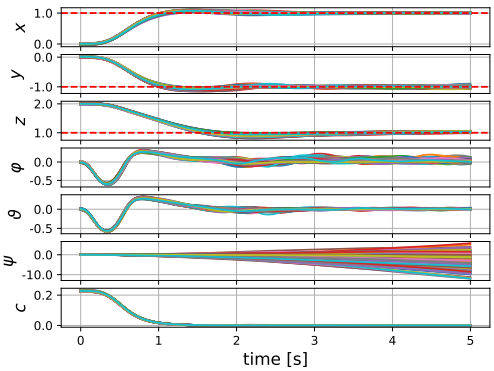
(b) Trial 2



(c) Trial 3

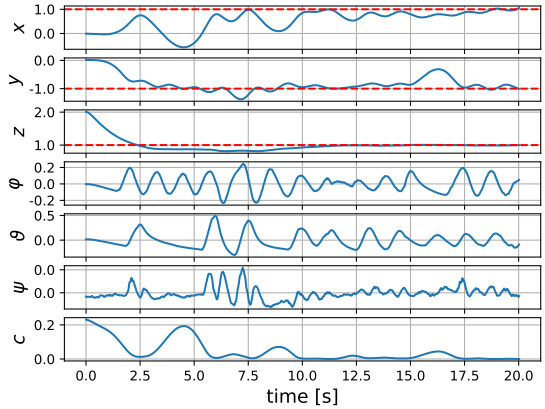


(d) Trial 4

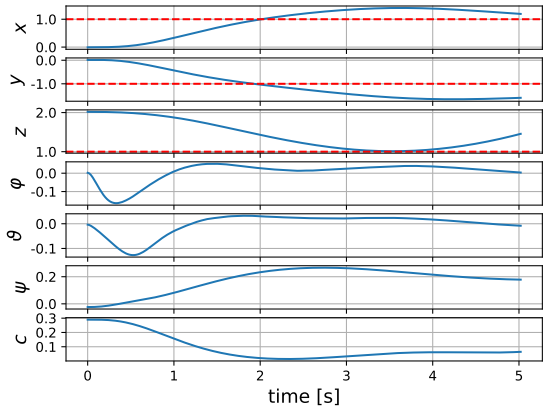


(e) Trial 5

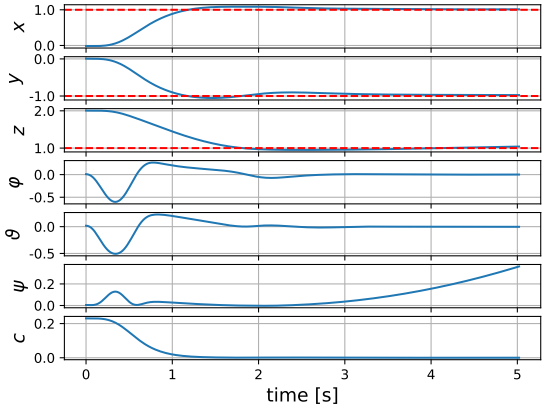
Fig. 5.4: Particles Long-Terms Predictions



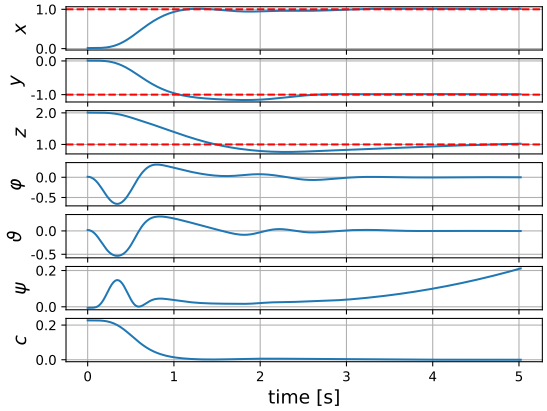
(a) Exploration



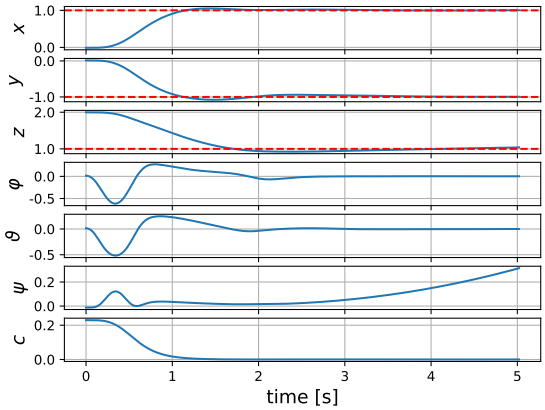
(b) Trial 1



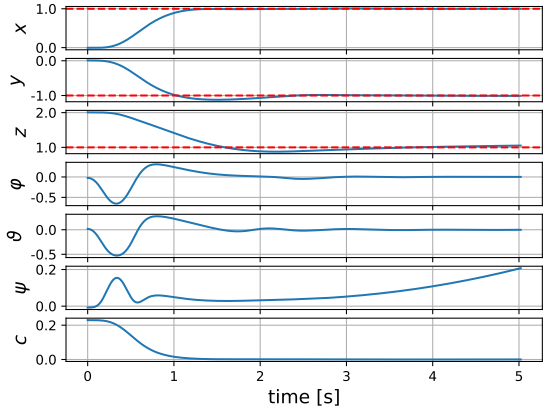
(c) Trial 2



(d) Trial 3



(e) Trial 4



(f) Trial 5

Fig. 5.5: True Rollout Evolution

5.2 Simulation with PyBullet

In Section 5.1, we introduced the nominal model simulated using ODE. As shown, the algorithm was fully capable of learning a policy that could optimally control the drone's position, significantly improving the performance of a traditional PID controller. In particular, all oscillatory trends are entirely suppressed, the rise time is improved, and stability for the desired goal is achieved.
Since the environment used so far is a nominal one, despite the immense improvements, verifying its performance in more realistic environments is necessary, which consequently presents external disturbances and greater uncertainties.
Unfortunately, it was impossible to test the algorithm with a real drone, so we relied on the PyBullet library, which can simulate realistic physics dynamics with a high degree of reliability.

5.2.1 PyBullet Simulator

PyBullet is an open-source, cross-platform physics library often used for 3D simulations in the field of robotics. PyBullet can simulate the laws of physics, including effects such as gravity, friction, and collision. Robot simulation is easily supported through the Universal Robot Description Format (URDF) file format. This allows users to simulate complex robots with many moving parts, which can interact with the environment in realistic ways.
Furthermore, PyBullet is easily integrable with OpenAI Gym, a toolkit for the development and comparison of reinforcement learning algorithms. This integration allows the creation of complex and realistic simulation environments in which reinforcement learning agents can be trained.

In our case, PyBullet will be used in combination with OpenAIGym for the simulation of a quadrotor. The environment is publicly available at the GitHub repository `https://github.com/utiasDSL/gym-pybullet-drones`.
This modeling has been done in great detail, allowing for the consideration of various physical aspects. These effects are:

1. *Ground Effect*: the phenomenon where a drone flying close to the ground experiences an increase in lift due to air flow interference between the drone and the ground. This is modeled in PyBullet by taking into account

the distance between the drone and the ground and adjusting the forces of the thrusters based on this distance.

2. *Drag Forces*: forces opposite to the motion of the drone through the air. These forces are modeled in PyBullet by calculating the drag force based on the drone's speed and a drag coefficient specified in the URDF file.
3. *Downwash*: the airflow that is pushed down by a drone's thrusters. This can influence the behavior of the drone, especially in a quadrotor where the downwash from one thruster can affect the other thrusters. This can be modeled in PyBullet using a physical model that takes into account the interactions between the airflows of the different thrusters.

Although accurate, it should be noted that PyBullet provides an approximate model. However, it can perfectly capture the main and most "problematic" physical details for our purpose.

The configuration of a drone proposed by the repository was used for this experiment. Using a pre-configured drone in PyBullet allows us to emulate the perspective of the MC-PILCO algorithm, which aims to control the drone starting from zero experience, providing a faithful simulation environment for the training and evaluation of this learning approach.

5.2.2 Policy Optimization

Exploration In the previous experiment, we used a PID position controller specifically designed for our nominal model. Therefore, to generate the first exploration, we will need to use a differently calibrated PID.
In the Gym-PyBullet repository, a PID position controller for the available quadrotor is provided. Similar to the one we developed for the nominal model, this one also aims to derive the forces $\boldsymbol{u} = \left[f_z, \tau_x, \tau_y, \tau_z\right]$ necessary for the correct displacement of the quadrotor, based on position errors. These forces will then be converted into the required speeds according to equations (2.56).
Having defined the new target position $\boldsymbol{x}^{des} = [0.75, -0.5, 1]$, the controller's response is shown in Figure 5.6.
From the image, we immediately notice that this controller has definitely been calibrated better and more precisely than the one designed for the model simulated with the ODE. However, in this case, as well, there are some significant issues. Firstly, we notice again a strongly oscillatory trend, with the achieve-

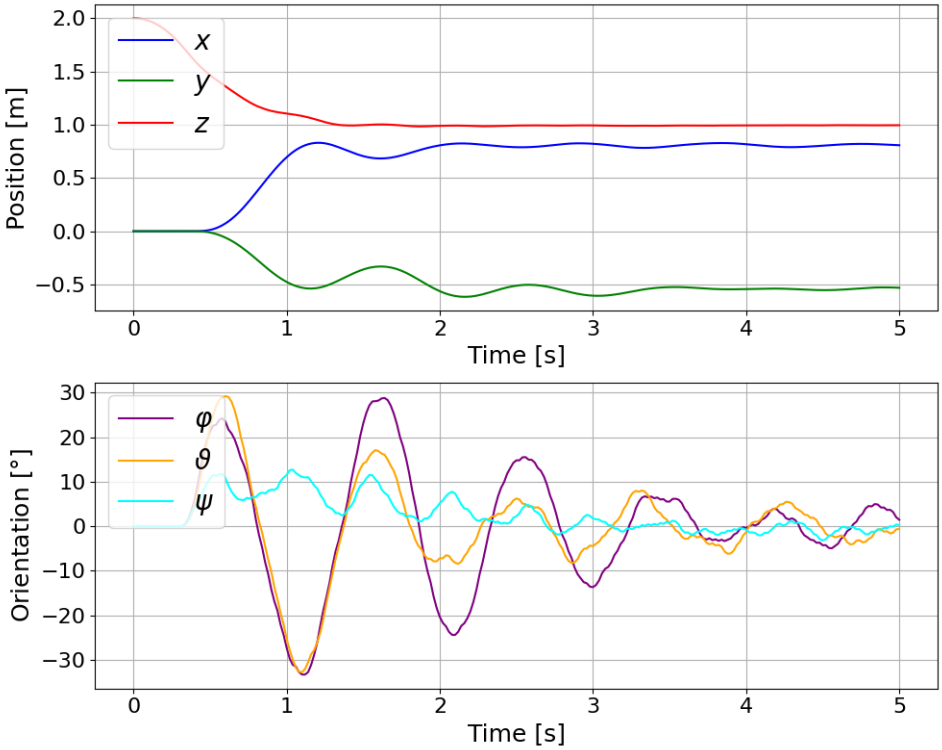


Fig. 5.6: PID Position PyBullet Control for Target Position $\boldsymbol{x}^{des}$

ment of significant inclinations in short periods of time. Moreover, even if only slightly, we notice that the position along y tends to stabilize with a slight offset from the desired point. We, therefore, want to start from this trajectory, provided in the exploration phase, and see what improvements MC-PILCO can make.

Results The parameters used for MC-PILCO are the same as before, reported in Table 5.1. The main difference, due to the difference between the models, lies in the exploration trajectory. In the nominal case, a high frequency in oscillations and inclinations, always less than 15°, was noticed. Despite the long time horizon, this dynamic leads to redundancy in the collected points.
In the environment developed with Gym-PyBullet, on the other hand, the controller is able to execute wider maneuvers and, therefore, more significant for the training phase of the model. In a way, it can be assumed that the optimization of the policy begins with a greater knowledge of the environment.
This consideration is supported by the obtained results, shown in Figure 5.8, where from the first trial, it is noted how the dynamics of the particles tend to converge. From this, the total cost in Figure 5.7 will also converge more quickly.

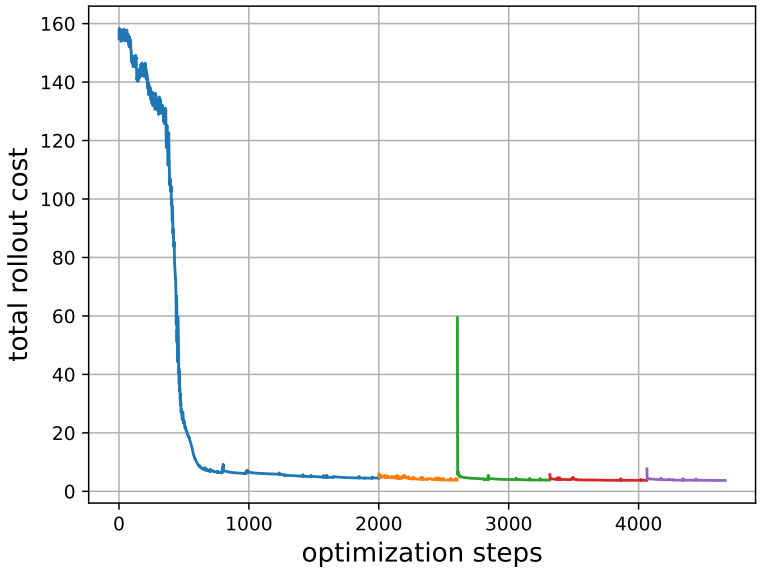
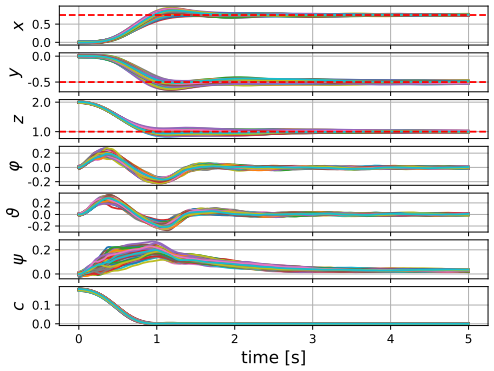


Fig. 5.7: Learning Plot

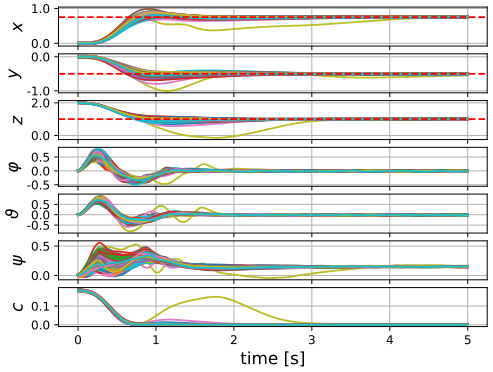
The results of better precision in estimating the GPR model are also noticed in applying the optimized policy to the real system. In fact, something that did not happen in the previous simulation, it is noted from the first trial that the model is able to stabilize, even if not yet precisely to the given target. In the following trials, however, the policy becomes increasingly precise and, in a completely overlapping way with the previous simulation, tends to improve its rise time and overshoot performance.

One last consideration that deserves to be mentioned is the input provided by the PID position controller and the policy optimized with MC-PILCO. In the first case, obtained from the exploratory phase, we notice that the controller gives a very "noisy" input. The PID indeed has a structure that allows it to react only based on the error relative to a specific target. Providing a more regular input becomes a difficult task in such a complex environment and with many external factors. This difficulty can be clearly seen in the input during the exploration phase in Figure 5.10.
On the contrary, MC-PILCO exploits the model learned from the previous experience to predict future evolution, thus allowing it to develop the best control strategy for the task to be performed. Already from the first trial, and more so in the subsequent ones, it is noticeable how the input provided to the system becomes much more regular, requiring smaller variations compared to the tra-

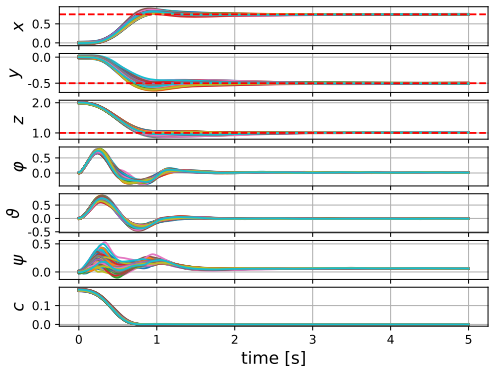
ditional controller. This definitely provides an advantage in real contexts as the motors also have their inertia, and inevitably too sudden speed changes lead to delays with consequent problems for more sophisticated controls.



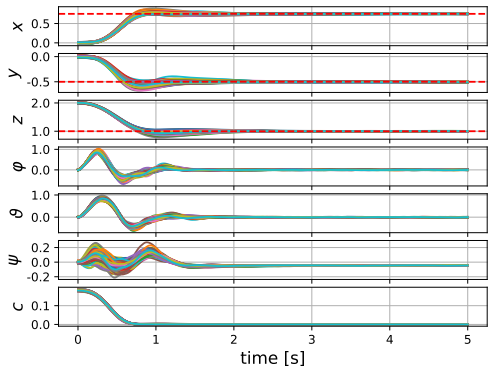
(a) Trial 1



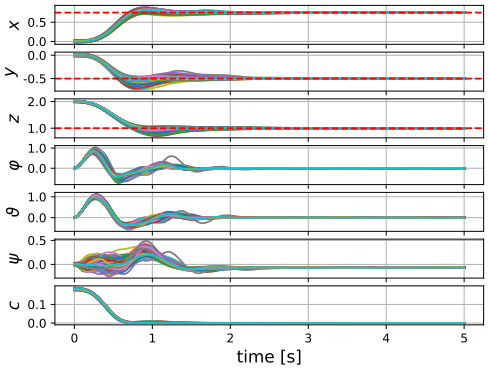
(b) Trial 2



(c) Trial 3

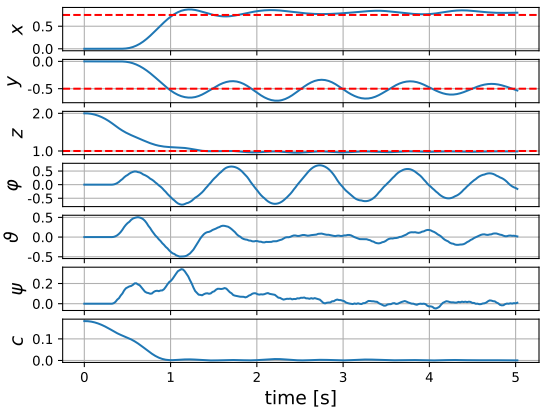


(d) Trial 4

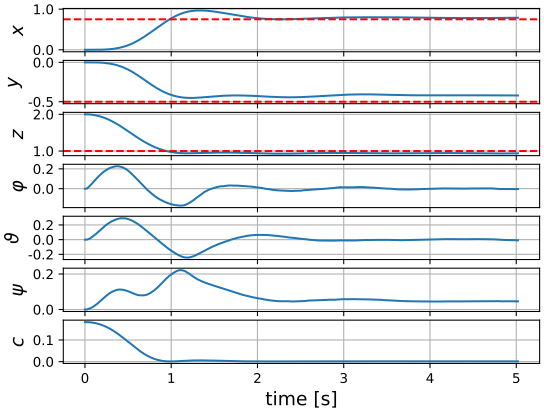


(e) Trial 5

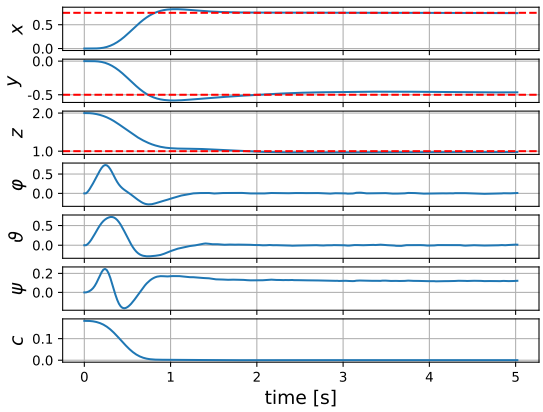
Fig. 5.8: Particles Long-Terms Predictions



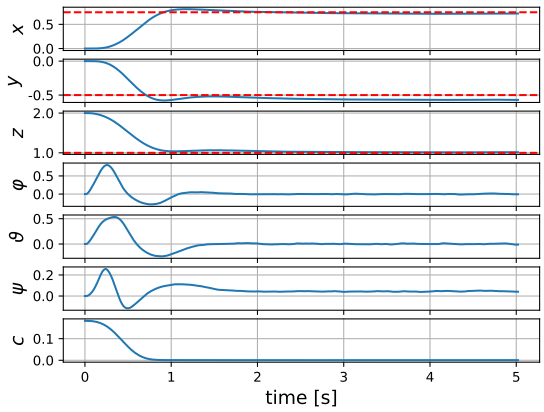
(a) Exploration



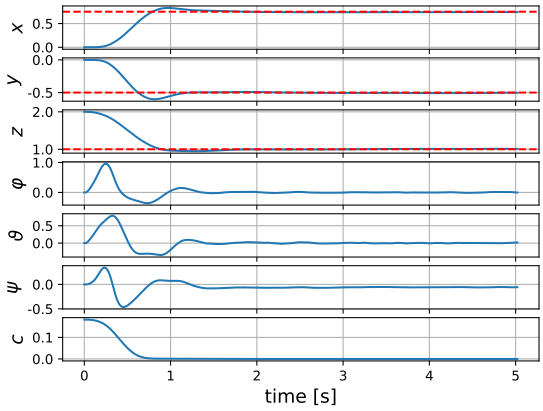
(b) Trial 1



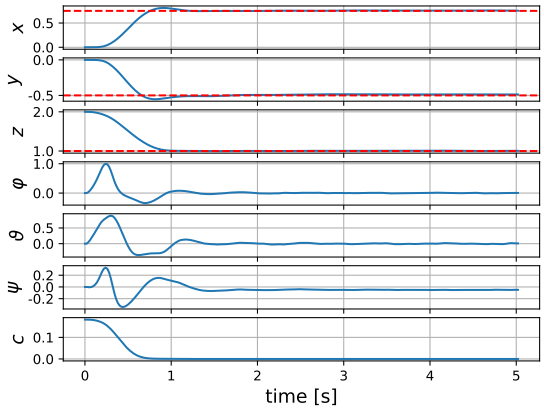
(c) Trial 2



(d) Trial 3

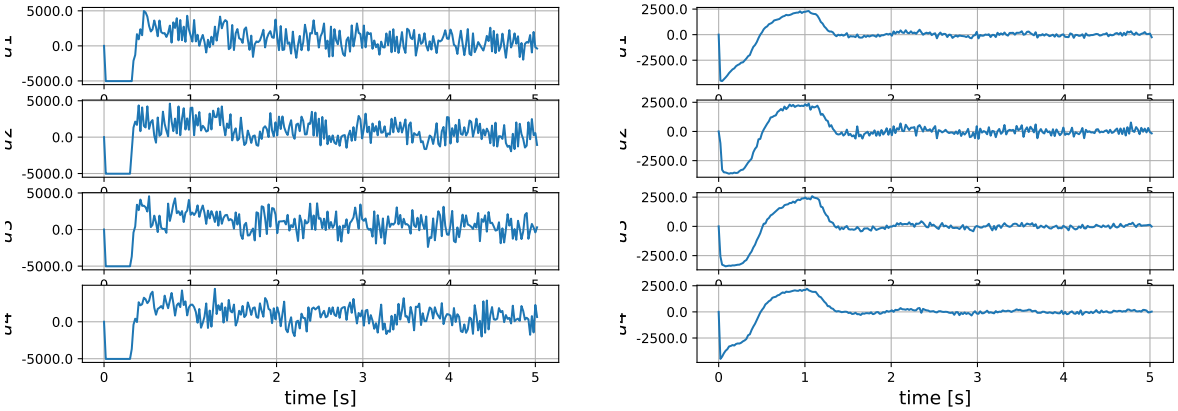


(e) Trial 4



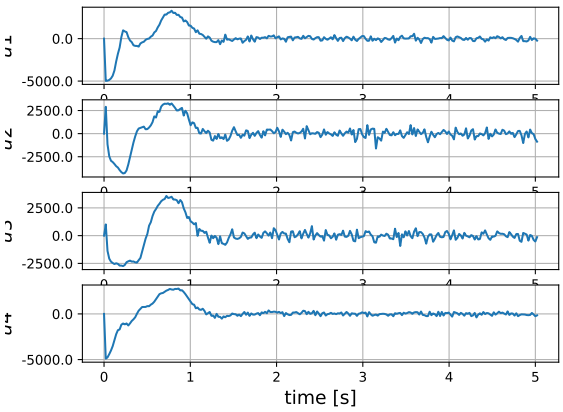
(f) Trial 5

Fig. 5.9: True Rollout Evolution

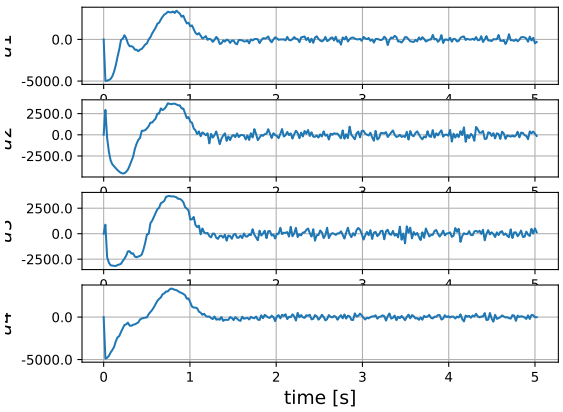


(a) Exploration (PID)

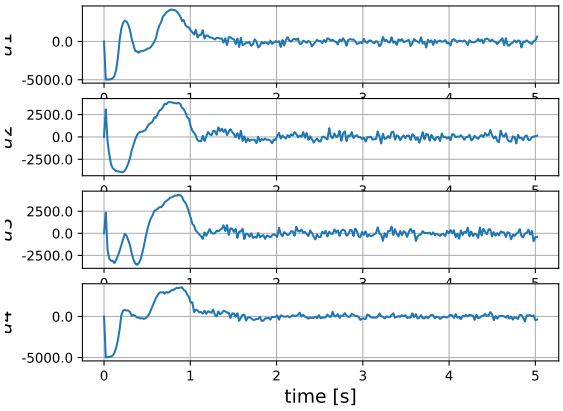
(b) Trial 1 (policy)



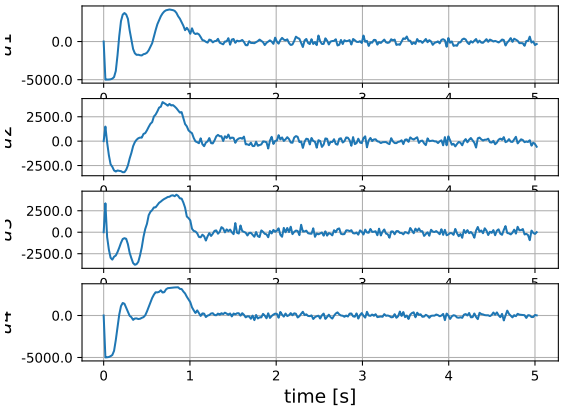
(c) Trial 2 (policy)



(d) Trial 3 (policy)



(e) Trial 4 (policy)



(f) Trial 5 (policy)

Fig. 5.10: Quadrotor Input

6

Conclusions and Future Works

6.1 Review of Previous Steps

In this study, our objective was to introduce and test a novel and alternative methodology for controlling a quadrotor, a type of unmanned aerial vehicle.

Our starting point involved an in-depth examination of the quadrotor system. We pursued this by deriving a comprehensive model of its dynamics. This process was grounded in Newton-Euler laws, which provide a robust theoretical basis for understanding rigid bodies' rotational and translational dynamics. In essence, we used these laws as a framework to understand how external forces and torques influence the motion of the quadrotor.

This dynamic model was initially derived with respect to two distinct frames of reference: an inertial frame (INF) and a body-fixed frame (BFF) attached to the quadrotor. These two reference frames provide different perspectives on the motion of the quadrotor, each offering unique insights.

Following the derivation of the model, we transformed it entirely into the inertial reference frame (INF). This transformation simplifies the subsequent control design by considering the quadrotor's dynamics from a single, fixed reference frame.

To validate our derived models, we implemented a PID controller, a well-known control strategy that adjusts the control output based on the present and past error values. The PID controllers were applied to test the response of our models. We aimed to verify that the dynamic evolutions under these controllers were equivalent for the INF and BFF-derived models. This was a

critical step to confirm the accuracy and reliability of our derived models. Upon successful verification, we proceeded to conduct an in-depth analysis of our novel quadrotor control methodology.

For this thesis, we sought to apply the MC-PILCO (Model-based Probabilistic Inference Learning Control) algorithm in a new context. From the outset, to ascertain its feasibility, we had to evaluate the ability of the chosen GPR model to accurately learn the dynamics of the drone.
This phase presented several challenges. Firstly, the representation of the model in a single reference frame played a crucial role. This allowed us to resort to simpler models that could be managed via speed integration. The process of transforming complex dynamics into a single frame simplifies the system's overall management by integrating velocity, we could calculate the pose of the quadrotor over time, a crucial requirement for our next control strategies.
Another hurdle we encountered pertained to the length of the required time horizons. For very high values, the cumulative error tended to grow exponentially. This error accumulation can be problematic, as it could lead to significant deviations from the real quadrotor trajectory over time. However, we noticed that obtaining a reliable predictive model was possible with a sufficient and significant amount of data.

Subsequently, we had to conduct a study related to approximation methods. Due to computational constraints, working with fewer data points was necessary. We resorted to a particular approximation method known as "Subset of Data", which allowed us to select only certain points based on an error criterion. This method involves the selection of a subset of data points that are most representative or informative in the context of the problem at hand.
Our findings showed that the model provided reliable estimates even with only tens of data points. This is noteworthy as it demonstrates that we can obtain a robust representation of the drone's dynamics with a manageable amount of data. This is advantageous as it reduces the computational burden, which can be a significant concern, especially for real-time applications.
With the successful implementation of the approximation method, we were able to proceed with the actual implementation of the MC-PILCO algorithm. As a model-based method, MC-PILCO relies on having a reliable system dynamics model. Our successful application of the SoD approximation method allowed us to build such a model.

By balancing these challenges, we were able to implement the MC-PILCO

algorithm. Based on probabilistic Gaussian process models, this algorithm efficiently optimizes control policies by actively learning from data. In our context, it allowed us to optimize the drone's control policy to evaluate its performance improvement.

6.2 Conclusions

The MC-PILCO algorithm implemented for this task displayed impressive results when compared to traditional control methods. In a simulated environment, all control performance indicators were significantly improved, illustrating the effectiveness of this approach.
One of the most conspicuous improvements was the optimization of the maneuvers. The flight paths generated by the MC-PILCO algorithm were much smoother and more regular than those obtained with the PID controller. Consequently, other performance measures, such as overshoot and rise time, were also notably improved. Improvements in these measures show that the MC-PILCO algorithm enabled the quadrotor to reach its target position more quickly and accurately than the PID controller.
Another important factor was the robust stability achieved by the quadrotor. It was able to reach the predefined goal with significant consistency and resilience. Robust stability is crucial in real-world applications as it ensures the system's performance is not overly affected by changes in the environment or minor deviations from the model assumptions.

As reliable and faithful to reality, as it may be, this algorithm has always been developed for control in simulated environments. However, based on the experience gained during the execution of this project, we can draw important conclusions.
Firstly, the reliability of the learned model for estimating the dynamics of the quadrotor plays a crucial role during the optimization phase. The ability to predict future evolutions with low uncertainty in response to specific inputs is necessary for algorithms that rely on long-term predictions. MC-PILCO, strongly based on this concept, requires the model to approximate the system's behavior with an extremely high level of precision.
In simulated contexts, we found that a model based on Gaussian Process Regression (GPR) could perfectly fulfill this task. However, this might not be as straightforward in real-world environments, where external disturbances and

uncertainties increase exponentially.
This condition could inevitably lead to a need for more data or, alternatively, the exploration of new, more effective approximation models. One such promising avenue could be the use of neural networks.

The results obtained with the MC-PILCO algorithm demonstrate that only a single significant exploration trajectory for the required task was necessary to learn the optimal policy. This is a significant finding as it indicates the potential efficiency of this approach in learning complex control policies. It reduces the need for extensive exploratory data, thus potentially accelerating the learning process and making the approach more feasible for real-world applications.
In conclusion, the results obtained are very promising for achieving short-term objectives. They show that the MC-PILCO algorithm could be a viable and advantageous alternative for controlling complex nonlinear systems. This opens up wide-ranging perspectives for quadrotor control.

The optimized policy's final outcome in the gym environment was thoroughly tested using a graphical user interface (GUI). In addition to showcasing the final result, the video recording also includes a comparison with the PID controller. The link to the video is `https://youtu.be/Eu_x050-rwA`.

6.3 Future Works

The results obtained indeed open up the prospect of significant advancements for this approach. First and foremost, it would be interesting to see how MC-PILCO is able to interface with the real world. The simulations we conducted demonstrated that, with an effective model and an appropriate exploration trajectory, the quadrotor could perform a point-to-point task effectively. Given the results obtained, it's reasonable to think that, with the right precautions, MC-PILCO could find an optimal policy, improving upon the performance of existing controllers, even in real-world environments.

Subsequently, it would also be interesting to see how this approach can handle different tasks. An intriguing case would be to provide a different goal point from the one reached in the exploration phase. To further complicate the task, it would be a good idea to no longer require the task to reach just one point but a series of points in sequence. This would necessitate further analysis of the problem and potentially require modifications to the initial system representation, perhaps needing to include the target in the state representation.

Further research needs to be undertaken to address these challenges and continue the advancement in the field of autonomous quadrotor control, assuming these endeavors are successful. In this regard, the future of MC-PILCO seems promising, offering a potent tool for the effective control of complex systems.

References

[1] A.Y.Elruby et al. "Dynamic Modeling and Control of Quadrotor Vehicle". In: *AMME Conference* 15th (29-31 May, 2012).

[2] Fabio Amadio et al. "Model-Based Policy Search Using Monte Carlo Gradient Estimation With Real Systems Application". In: *IEEE TRANSACTIONS ON ROBOTICS* 38 (6, December 2022). DOI: 10.1109/TRO.2022.3184837.

[3] BaraJ.Emran and Homayoun Najjaran. "A review of quadrotor: An underactuated mechanical system". In: *Annual Reviews in Control* (19 October 2018). DOI: 10.1016/j.arcontrol.2018.10.009.

[4] Richard Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[5] Davide Bicego et al. "Nonlinear Model Predictive Control with Enhanced Actuator Model for Multi-Rotor Aerial Vehicles with Generic Designs". In: *Journal of Intelligent Robotic Systems* (2020). DOI: 10.1007/s10846-020-01250-9.

[6] Samir Bouabdallah and Roland Siegwart. "Full Control of a Quadrotor". In: *Proceedings of the 2007 IEEE/RSJ International, Conference on Intelligent Robots and Systems* (Oct 29 - Nov 2, 2007). DOI: 10.1109/IROS.2007.4399042.

[7] Christopher K.I. Williams Carl E. Rasmussen. *Gaussian Processes for Machine Learning*. London, UK: The MIT Press, 2005.

[8] Anezka Chovancová et al. "Mathematical Modelling and Parameter Identification of Quadrotor". In: *Procedia Engineering* 96 (2014), pp. 172–181. DOI: 10.1016/j.proeng.2014.12.139.

[9] Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. "Gaussian Processes for Data-Efficient Learning in Robotics and Control". In: *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE 37* (2015).

[10] Marc Peter Deisenroth and Carl Edward Rasmussen. "PILCO: A Model-Based and Data-Efficient Approach to Policy Search". In: *Proceedings of the 28th International Conference on Machine Learning* (2011).

[11] Thor I. Fossen. *Marine Control System: Guidance, Navigation, and Control of Ships, Rigs and Underwater Vehicles*. Trondheim, Norway: MARINE CYBERNETICS, 2002.

[12] Emil Fresk and George Nikolakopoulos. "Full Quaternion Based Attitude Control for a Quadrotor". In: *2013 European Control Conference (ECC)* (July 17-19, 2013, Zürich, Switzerland).

[13] Emil Fresk and George Nikolakopoulos. "Full Quaternion Based Attitude Control for a Quadrotor". In: *European Control Conference (ECC)* (July 17-19, 2013).

[14] Osama Mohammed Hamdy and Hassan Talal Hassan. "Modeling, Simulation and Control of Quadcopter using PID Controller". In: *4th IUGRC International Undergraduate Research Conference, Military Technical College* (2019).

[15] Moad Idrissi, Mohammad Salami, and Fawaz Annaz. "A Review of Quadrotor Unmanned Aerial Vehicles: Applications, Architectural Design and Control Algorithms". In: *Journal of Intelligent Robotic Systems* (2022). DOI: 10.1007/s12555-018-0860-9.

[16] Maidul Islam, Mohamed Okasha, and Erwin Sulaeman. "A Model Predictive Control (MPC) Approach on Unit Quaternion Orientation Based Quadrotor for Trajectory Tracking". In: *International Journal of Control, Automation and Systems* (2019). DOI: 10.1007/s12555-018-0860-9.

[17] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[18] J. Quiñonero-Candela, CE. Rasmussen, and CKI. Williams. *Approximation Methods for Gaussian Process Regression*. Neural Information Processing. Cambridge, MA, USA: MIT Press, Sept. 2007, pp. 203–223.

[19] Filippo Rinaldi, Sergio Chiesa, and Fulvia Quagliotti. "Linear Quadratic Control for Quadrotors UAVs Dynamics and Formation Flight". In: *Journal of Intelligent Robotic Systems* (2013). DOI: 10.1007/s10846-012-9708-3.

[20] Francesco Sabatino. *Quadrotor control: modeling, nonlinear control design, and simulation*. Stockholm, Sweden: Masters Degree Project, 2015.

[21] Bruno Siciliano et al. *Robotics: Modelling, Planning and Control*. Stockholm, Sweden: Springer, 2009.

[22] Matej Smid and Jinrich Duniki. *Online Learning and Control for Data-Augmented Quadrotor Model*. 2 Apr 2023. arXiv: 2304.00503v1 [cs.RO].

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Adaptive Computation and Machine Learning. Cambridge, MA: MIT Press, 2020.

[24] Dewei Zhang et al. "The Quadrotor Dynamic Modeling and Indoor Target Tracking Control Method". In: *Mathematical Problems in Engineering* (2014). DOI: 10.1155/2014/637034.

[25] Xiadong Zhang et al. "A Survey of Modelling and Identification of Quadrotor Robot". In: *Abstract and Applied Analysis* (2014). DOI: 10.1155/2014/320526.

Acknowledgments

A thank you is due to Professor Ruggero Carli for initially directing this work and setting the foundation for the development of the thesis topic.
Appreciation is extended to Dr. Alberto Dalla Libera, whose assistance during the execution of the work has been crucial in completing this project.
Thanks go to my parents, aunt, and uncle for their constant support throughout this journey.
Finally, thanks to my friends for sharing this experience and for the moral support provided.
To all, my gratitude for contributing to this academic journey.