**Budapest University of Technology and Economics**
Faculty of Electrical Engineering and Informatics
Department of Control Engineering and Information Technology

# Development of Efficient Control Methods to Support Agile Maneuvering of Autonomous Quadcopters

MASTER'S THESIS

*Author*
Péter Antal

*Advisor*
Dr. Tamás Péni
Dr. Roland Tóth
Dr. habil. István Harmati

December 8, 2022

M Ű E G Y E T E M 1 7 8 2

Diplomaterv feladat

**Antal Péter**

MSc. villamosmérnök hallgató részére

# Hatékony irányítási módszerek kidolgozása autonóm kvadkopterek agilis manőverezésének támogatására

Az autonóm kvadkopterek számos területen alkalmazhatók, a felderítési, mentési feladatoktól a profi filmfelvételek készítésén át a látványos bemutatókig. Az alkalmazási kör szélesedése egyre összetettebb feladatok megoldást igényli. Olyan eljárások kidolgozására van szükség, amelyekkel új, összetett képességek (pl. tanult pilóta számára is bonyolult manőverek végrehajtása) könnyen elsajátíthatók, azaz gyorsan megtervezhetők és precízen végrehajthatók. A diplomaterv célja olyan hatékony szabályozási eljárások, pályatervezési algoritmusok és gépi tanulásra épülő módszerek kidolgozása, amelyek lehetővé teszik az autonóm drónok számára új, agilis manőverek gyors megtanulását. Az eljárásokat egy konkrét, nehéz manőver, a backflip megvalósításán kell bemutatni, először szimulációban majd valós rendszeren, a Bitcraze Crazyflie miniatűr drónjain.

**Megoldandó feladatok:**
***Első félév***

1. Végezzen irodalomkutatást autonóm drónok dinamikus modellezése, pályakövető szabályozása és pályatervezési algoritmusai témakörben, előnyben részesítve olyan irányítási módszereket, amelyeket gyors manőverezésre dolgoztak ki.
2. Hozzon létre Crazyflie drónok szimulációjára alkalmas szoftverkörnyezetet Python programozási nyelven a PyBullet OpenAI Gym szoftverek felhasználásával.
3. Tervezzen nyílt hurokban alkalmas beavatkozójel-sorozatot a bukfenc manőver megvalósítására. Törekedjen az optimalizálási feladat minél egyszerűbb felírására!
4. Oldja meg az előző feladatot zárt köri szabályozással is! Ehhez tervezzen pályakövető szabályozást a Crazyflie kvadkopterhez, majd pontos pályakövetést feltételezve tervezzen spline trajektóriát a bukfenc manőverre!
5. Implementálja az eljárásokat a 2. pontban kidolgozott szimulációs környezetben. Hasonlítsa össze az algoritmusokat robosztusság, performancia (pontosság) és számítási komplexitás szempontjából!

***Második félév***

6. Vizsgálja meg, hogy miként lehet megerősítő tanulást alkalmazni a bukfenc manőver kivitelezésében. Hasonlítsa össze ezt a megközelítést a korábbi módszerekkel.
7. Ismerkedjen meg a Sztaki-ban felépített AIMotionLab hardver- és szoftverkörnyzetével, különös tekintettel a nagypontosságú beltéri pozícionálórendszer és a kommunikációs interfészek használatára.
8. Ismerje meg a Crazyflie kvadkopter működését, a rendelkezésre álló szoftverkomponenseket.
9. Implementálja korábban tervezett eljárásokat a valós drónon is a Crazyswarm keretrendszer felhasználásával!
10. Értékelje a kapott eredményeket a valós alkalmazhatóság szempontjából! Hasonlítsa össze a szimulációban kapott eredményekkel.

| | |
|---|---|
| **Konzulens:** | Dr. habil. Harmati István egyetemi docens |
| **Külső konzulens:** | dr. Péni Tamás, dr. Tóth Roland, Számítástechnikai és Automatizálási Kutatóintézet (SZTAKI) |

Budapest, 2022. március 17.

Dr. habil. Harmati István
témavezető
egyetemi docens

Dr. Kiss Bálint
tanszékvezető
egyetemi docens

# Contents

# HALLGATÓI NYILATKOZAT

Alulírott *Antal Péter*, szigorló hallgató kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2022. december 8.

*Antal Péter*
hallgató

# Abstract

Continuous development of quadcopters in recent years has led to their wide application in industry and in many research areas, including rescue missions, agricultural monitoring, making camera footages or spectacular entertainment shows. As the application fields are continuously growing, more and more capabilities are expected from the drones. Agile maneuvering in a cluttered environment, working in team, cooperating with humans or performing acrobatic maneuvers, just to name a few. These tasks require to perform complex, fast maneuvers that pushes the drones to their physical limits. Classical flight controllers based on a linearized dynamical model can no longer cope with these tasks and more advanced control methods that exploit the entire operating domain are needed. These control algorithms can be developed by mathematically well-grounded nonlinear design techniques (e.g. differential geometry) or machine learning approaches.

In this work, trajectory planning and motion control design to execute a flip maneuver is presented for a nano quadcopter. Such a maneuver is a challenging task even for experienced drone pilots. The complexity and speed of the maneuver is characterized by the fact that it takes less than a second to complete, during which the vehicle is able to make a full turn around one of the horizontal axes.

After presenting the theoretical background and relevant literature, two approaches are proposed to perform the flip maneuver. The first is a simple, open-loop strategy which is designed by optimizing the sequence of specific motion primitives describing the flip maneuver using a digital twin model of the quadcopter. The second approach is based on a nonlinear geometric controller, designed by using a dynamic model of the drone, that is able to track even highly challenging reference trajectories. In order to compensate the effect of modelling uncertainties and external disturbances, we collect measurement data, and design a novel robust adaptive geometric controller based on an augmented dynamic model of the quadcopter. To perform the backflip maneuver, a feasible reference trajectory is designed that describes the intended state evolution. Then, the designed trajectory is precisely tracked by the proposed geometric controller. The advantages and disadvantages of the two control methods are evaluated both in simulations and by real-world experiments using Bitcraze Crazyflie 2.1 quadcopters.

Finally, a reinforcement learning framework is introduced to improve the performance of the proposed control schemes. Gaussian Process based model augmentation is used to account for the unknown dynamics of the real quadcopter using measurement data. Then, policy learning is applied to train an augmented feedback controller and improve the control performance of the geometric method.

# Összefoglaló

A kvadkopterek folyamatos fejlődése az elmúlt évek során ahhoz vezetett, hogy ma már számos területen alkalmazzák őket az iparban és kutatások során egyaránt, mint például felderítés mentőakciók során, kamerafelvételek készítése hírközlés céljából, illetve mezőgazdasági monitorozás. Alkalmazási területeik folyamatos bővülésével a kvadkoptereknek egyre több képességgel kell rendelkeznie, mint agilis manőverezés akadályok közötti szűk helyeken, kooperáció több drónnal, emberekkel való együttműködés, illetve akrobatikus manőverek bemutatása. Ilyen feladatok végrehajtásához gyors, komplex manőverezésre van szükség a drón fizikai határait teljes mértékben kihasználva. Ehhez a linearizált dinamikai modellre tervezett klasszikus szabályozók nem alkalmazhatóak, komplexebb irányítási stratégiára van szükség, ami az egész működési tartományon képes szabályozni a kvadkopter mozgását. Ilyen algoritmusok tervezéséhez felhasználhatóak nemlineáris irányítási módszerek, például differenciálgeometria vagy gépi tanulás alapján.

Dolgozatom során egy bonyolult, tapasztalt emberi pilóták számára is kihívást jelentő manőver, a bukfenc pályatervezését és pályakövető szabályozását mutatom be egy kisméretű kvadkopteren szemléltetve. A manőver komplexitását és gyorsaságát jellemzi, hogy a végrehajtása kevesebb, mint egy másodpercet vesz igénybe, ami alatt a jármű egy teljes fordulatot képes megtenni.

Az elméleti háttér és releváns irodalom bemutatása után a manőver végrehajtására két különböző megközelítést demonstrálok. Az első egy nyílt hurkú irányítási stratégia a bukfencet leíró paraméterezett primitívekből álló szekvenciának a drón digitális ikermodelljén való optimalizációja alapján. A második egy nemlineáris geometriai elvű szabályozás a kvadkopter dinamikai modelljére tervezve, ami igen bonyolult referenciapálya követésére is képes. A modellezési bizonytalanságok és külső zavarások hatásainak kompenzációjaként mérési adatokat gyűjtünk, majd az adatok alapján kiterjesztett dinamikai modellre továbbfejlesztjük a geometriai szabályozót, így elérve egy újfajta robusztus, adaptív irányítási algoritmust. A bukfenc manőver végrehajtásához szükséges egy alkalmas referenciapálya tervezése is specifikálva a kívánt mozgás állapotait, amit optimalizációval oldunk meg, majd ezt a kvadkopter a javasolt pályakövető szabályozással precízen tudja követni. A két irányítási stratégia előnyeinek és hátrányainak elemzését szimulációs és a valós rendszeren végzett mérési eredmények alapján mutatjuk be.

Végül egy megerősítő tanulásra épülő keretrendszer kerül bemutatásra a javasolt szabályozási algoritmusok teljesítményének növelésére. A valódi kvadrokopter és az ideális modell közötti eltérés mérési adatok felhasználásával, Gauss-folyamatok formájában kerül identifikációra. Ezután policy learning algoritmust alkalmazunk a geometriai módszer kiterjesztésére, ezzel a szabályozás teljesítményének növelésére.

# Chapter 1

# Introduction

The aim of this work is to develop and implement trajectory planning and motion control algorithms that allow a nano quadcopter to perform complex maneuvers at high speed. The backflip maneuver has been chosen as an example, because it is a challenging task even for an expert human pilot, and it emphasizes the complex nonlinear behaviour of the drone. The complexity and speed of the maneuver is characterized by the fact that it takes less than a second to complete, during which the vehicle is able to make a full turn around one of the horizontal axes.

The maneuver is performed by miniature quadcopters, more generally micro aerial vehicles (MAVs). MAVs are a type of unmanned aerial vehicles (UAVs) that has small size and are usually autonomous. Due to the small-sized and low-power sensor and computer units, they are now continuously developed and applied both in industry and research. Their most common applications include military purposes (exploration and observation), search and rescue missions, aerial photography, distributed sensing and information processing while working in team, or agile maneuvering in a cluttered environment. In Figure 1.1, three micro aerial vehicles are shown: the first two developed for military applications, and the third for research and education.

Many common tasks of a miniature quadcopter, such as navigating in a cluttered environment or flying in strong wind require to perform complex, fast maneuvers that push the drones to their physical limits. In these cases, classical flight controllers designed for a linearized dynamical model are no longer applicable and more advanced control methods that exploit the entire operating domain are needed. These algorithms can be developed based on nonlinear control techniques, or machine learning approaches.

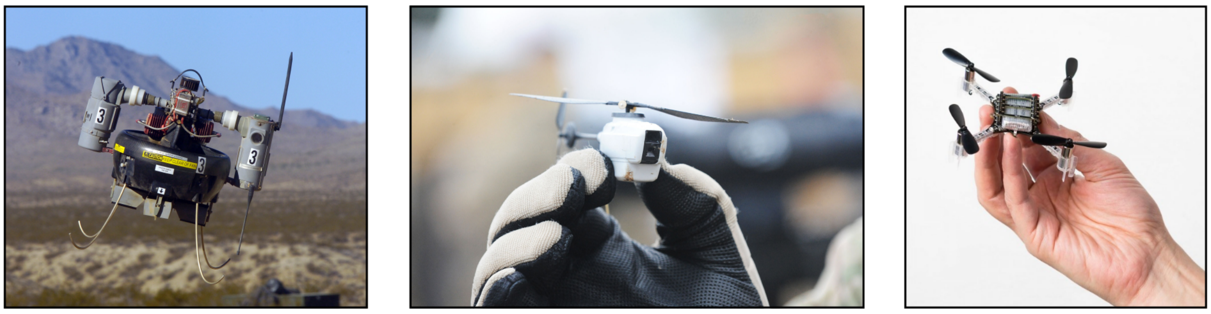


**Figure 1.1:** Micro aerial vehicles: RQ-16 T-Hawk (left), Black Hornet Nano (center), and Bitcrcaze Crazyflie 2.1 (right).

At the AIMotion Lab of ELKH SZTAKI, there is a flying arena with a fleet of Bitcraze Crazyflie 2.1 miniature quadrotors. These drones are designed for research and education purposes, therefore they are open both in terms of hardware and software details, making it possible to implement our own algorithms on-board. In order to achieve precise indoor positioning and provide fly safely, an external OptiTrack motion capture system provides position and orientation information for the drone, the data of which is fed back to the on-board controller via radio communication. With the highly reliable and precise localization, the Crazyflie is able to track complex trajectories with high performance using a suitable flight controller.

Backflipping with a quadcopter is an interesting control problem both from theoretical and application point of view. Flying vehicles are often subject to significant disturbances (e.g. strong wind or collision) which results in being upside-down. The recovery from that state is very complex, but a quadcopter that can perform a backflip has a higher chance to regain stability and follow its mission. From theoretical point of view, the problem is interesting not only because it emphasizes the nonlinearity of the dynamic model, but also requires special considerations regarding the attitude representation and control.

In the literature, there are several different control strategies to perform the flip maneuver. In [15], energy-based control is applied to overcome the uncontrollability of the quadcopter at singular configurations to follow a circular or clothoidal reference trajectory. In [10], Lyapunov-based controller synthesis is used to execute multi-flip maneuvers with quadcopters. Machine learning approaches are utilized in many cases, for example to imitate the maneuver performed by an expert drone pilot with apprenticeship learning [1], or design time-optimal trajectories with deep reinforcement learning [48] and learn acrobatic maneuvers [26, 24].

A simple learning strategy for adaptive feedforward control is proposed in [34], based on the optimization of a parametric motion primitive sequence. Backflipping pushes the actuators of the quadcopter to their physical limits, as application of near-maximal and minimal control inputs are required. This approach builds on the theory of bang-bang control and first-principles motion primitive design to perform and optimize the flip maneuver. The proposed method is easy to implement and it is well suited for generating a feasible trajectory, however, many trials on the real robot are necessary to optimize the parameters of the motion, and the feedforward control is sensitive to parameter uncertainties.

Geometric control is a nonlinear, model based approach for the feedback control of rigid bodies in 3D space. In [30], it is theoretically proven that geometric control is able to stabilize the orientation of a quadcopter in the whole operating domain based on differential geometric considerations and Lyapunov stability. In [31], the geometric control is augmented by robust terms to guarantee uniformly ultimately bounded tracking errors in the presence of disturbances in the quadcopter dynamics. However, the control design requires a priori knowledge of the magnitude of external disturbances which can be challenging in unknown situations and environments or lead to poor performance. An adaptive augmentation of geometric control is proposed in [21], where the adaptive terms compensate the effects of uncertainties in the quadrotor dynamics, while the stability of the closed loop system is proven mathematically. Artificial neural networks are used in [6] to develop an adaptive geometric control law that renders the quadcopter able to perform complex maneuvers in wind fields. Although neural networks can be implemented efficiently, the training often requires many data points, and it is difficult to estimate the uncertainty of the outputs. Geometric control has been improved and extended by other researchers, as well, and it is the basis of several advanced trajectory design and agile maneuvering control algorithms, e.g. [49, 36].

Recently, the research of reinforcement learning (RL) and its fusion with control systems is also a very popular topic. Reinforcement learning is a type of machine learning, where there are no labeled outputs for the inputs of the algorithm, only rewards, and the aim of the learning scenario is to find the optimal actions, which maximize the cumulative reward of the achieved states. In quadrotor control, RL is used to extend the capabilities of classical, optimization-based trajectory planning and control methods by interacting with the real robot and collecting measurement data. Typical research applications include simulation to reality transfer [39], increasing the agility and speed of racing drones even more [48], and learning agile maneuvers [23, 46].

Motivated by the challenges of the introduced state-of-the art research, our aim is to compare existing trajectory planning and control methods for backflipping, and develop novel algorithms to overcome their observed limitations. The main contributions of our present work are as follows:

1. We improve the convergence and the training time of the feedforward control design of [34] by applying Bayesian optimization with Gaussian Process surrogate function, making it possible to effectively apply the method with real experiments based tuning.
2. By Gaussian Process (GP) based augmentation of a nominal quadcopter model, we achieve adaption to unknown model dynamics and external disturbances together with quantification of the remaining model uncertainty. A robust geometric control scheme is designed that exploits the GP model for improved robust performance compared to previous methods. We also show convergence and robustness of the resulting method.
3. To execute the flip maneuver by the robust geometric approach, we propose an optimization-based trajectory planning method. The algorithm is based on quadratic programming and it is computationally efficient.
4. We compare the proposed methods in simulation and experimentally in performing the backflip maneuver.
5. We discuss possibilities to improve the performance and achieve online adaption capabilities of the proposed control systems using reinforcement learning methods.

The structure of this thesis is depicted in Figure 1.2, and builds up as follows. First, we introduce Gaussian Process regression, an efficient, inference based function approximation method, on which our proposed Bayesian optimization and adaptive control schemes are based on. Second, we give an overview of the modelling of the quadrotor flight dynamics in Chapter 3. In Chapter 4, we propose a Bayesian optimization based design of a motion primitive sequence that provides a feedforward control scheme for the backflipping maneuver. In Chapter 5, we introduce geometric control of quadrotors, we propose a GP model augmentation based nonlinear robust adaptive geometric control approach for the precise tracking of complicated reference trajectories, and an optimization-based trajectory planning method for performing the backflip. In Chapter 6, we evaluate and compare the performance of the introduced feedforward and feedback control approaches via numerical simulations. In Chapter 7, we present the implementation of the proposed algorithms, and demonstrate their performance in experiments. In Chapter 8, we introduce a reinforcement learning framework to improve the performance of the proposed geometric control by model augmentation and policy learning. Finally, the conclusions and possibilities for further improvements are summarized in Chapter 9.

The results of this thesis are partly published in a conference paper [3], while, based on the results beyond [3], a journal article has also been submitted [4]. A video demonstrating the simulations and experiments is available at `https://youtu.be/AhqfXZ-CPqM`.

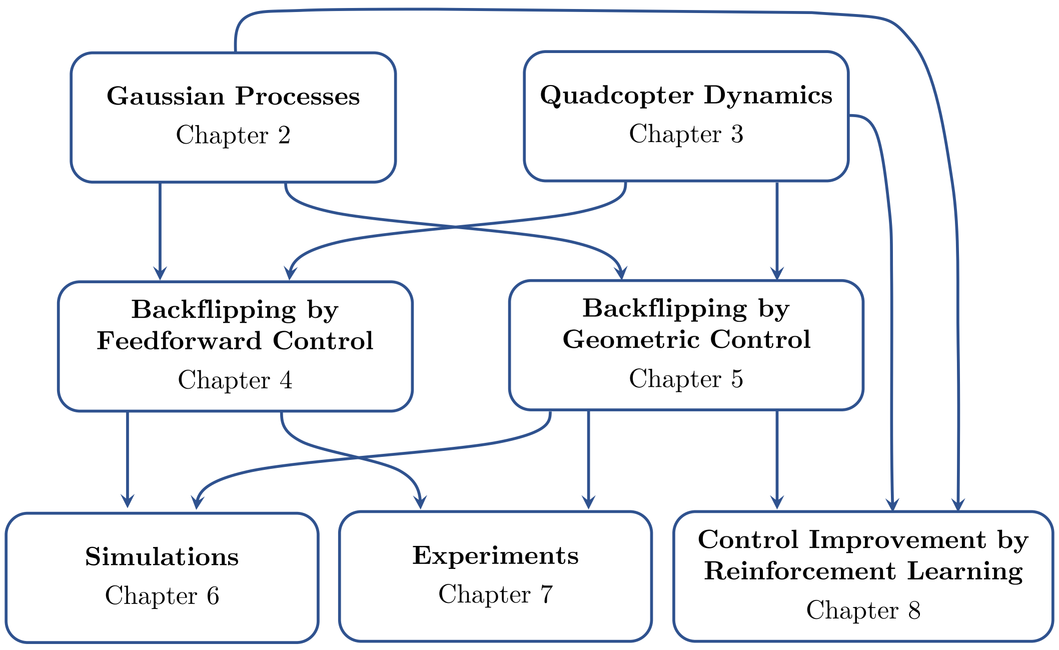


**Figure 1.2:** Illustration of the main topics of the thesis and their relations.

# Chapter 2

# Gaussian Process Regression

The approximation of unknown functions from measurement data is a common problem in the engineering practice. For this purpose, artificial neural networks (ANNs) have gained popularity in recent years due to their efficient training properties, generalization capability, and ability to represent large datasets. The most widely applied type of ANNs is multilayer perceptron. Although these networks have many relevant applications, they usually have large number of parameters, can suffer from overfitting, and are sensitive to hyperparameter tuning. Moreover, the uncertainty of their output is difficult to obtain.

Gaussian Processes (GPs) are universal function approximators representing a single layer neural network [43]. Due to their nonparametric structure, flexibility, and ability to express the uncertainty of the approximation, GPs are widely used in robotics and control systems technology [22, 33, 14]. In this work, first we apply GPs to define a surrogate model for Bayesian optimization of the backflip motion primitive sequence parameters, which is detailed in Chapter 4. Second, we use them to augment geometric control with robust and adaptive terms, which is proposed in Chapter 5. The reinforcement learning based algorithms proposed in Chapter 8 include applying GPs both for modelling and control design.

## 2.1 Definition and main principles

GP regression is used for estimating an unknown, possibly nonlinear relation $f_0 : \mathbb{R}^n \to \mathbb{R}$ between input $x \in \mathbb{R}^n$ and output $y \in \mathbb{R}$ variables based on noisy observations, in the form of

$$y = f_0(x) + \epsilon \tag{2.1}$$

where $\epsilon$ is an independent noise process with $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. In fact, $y$ and $\epsilon$ are random variables, but for the sake of simplicity, we will not use a different notation for their sample realization. Consider that a set of observations $\mathcal{D}_N = \{\bar{x}_i, \bar{y}_i\}_{i=1}^N$ is available from (2.1). The core idea of GP based estimation of $f_0$ is to consider that candidate estimates $f$ belong to a GP, seen as a prior distribution. Then, using $\mathcal{D}_N$ and this prior, a predictive GP distribution of $f$ is computed that provides estimate of $f_0$ in terms of its mean and describes uncertainty of this estimate by its variance.

In terms of definition, a *Gaussian Process* $\mathcal{GP} : \mathbb{R}^n \to \mathbb{R}$ assigns to every point $x \in \mathbb{R}^n$ a random variable $\mathcal{GP}(x) \in \mathbb{R}$ such that for any finite set $x_1 \ldots x_N$, the joint probability distribution of $\mathcal{GP}(x_1), \ldots, \mathcal{GP}(x_N)$ is Gaussian. GPs are fully determined by their mean and covariance functions, hence if $f$ is a GP with mean $m$ and covariance $\kappa$,

$$m(x) = \mathbb{E}\{f(x)\}, \tag{2.2a}$$
$$\kappa(x,\tilde{x}) = \mathbb{E}\{(f(x)-m(x))(f(\tilde{x})-m(\tilde{x}))\}, \tag{2.2b}$$

then the joint Gaussian probability of $\mathcal{GP}(x_1), \dots, \mathcal{GP}(x_N)$ is $\mathcal{N}(M_x, K_{xx})$ with

$$M_x = [\; m(x_1) \;\; \cdots \;\; m(x_N) \;]^\top, \tag{2.3a}$$
$$[K_{xx}]_{i,j} = \kappa(x_i, x_j), \quad i,j \in \{1 \dots N\}. \tag{2.3b}$$

Both $m$ and $\kappa$, where the latter is also often called a *kernel* function due to the relation of GPs to *Reproducing Kernel Hilbert Space* (RKHS) estimators, are often parametrized in terms of *hyperparameters* $\theta \in \mathbb{R}^{n_\theta}$. In fact, taking $f$ with mean $m$ and covariance function $\kappa$ as the prior distribution in the estimation process defines the prior knowledge about $f_0$ in terms of $m$, while the choice of $\kappa$ determines the Hilbert space in which an estimate of the function is searched for. Parametrization of $m$ and $\kappa$ in terms of $\theta$ allows to adjust the prior, i.e., these choices to the estimation problem of $f_0$ using $\mathcal{D}_N$. For the estimation of a smooth $f_0$, a *Squared Exponential* (SE) kernel for $\kappa$ is a common choice. The SE kernel is characterized by

$$\kappa_{\mathrm{SE}}(x,\tilde{x}) = \sigma_{\mathrm{f}}^2 \exp\left(-\frac{1}{2}(x-\tilde{x})^\top \Lambda^{-1}(x-\tilde{x})\right), \tag{2.4}$$

where hyperparameters are the scaling $\sigma_{\mathrm{f}}$, used for numerical conditioning, and the symmetric matrix $\Lambda$, which determines the smoothness of the candidate function class along each $x_i$.

## 2.2 Bayesian inference

Based on the given dataset $\mathcal{D}_N$, and the prior $f$ with mean $m$ and covariance $\kappa$,

$$p(Y \mid X, \theta) = \mathcal{N}(M_x, K_{xx} + \sigma_\epsilon^2 I) \tag{2.5}$$

describes the probability density function of the outputs $Y = [\; y_1 \;\; \cdots \;\; y_N \;]^\top$ seen as random variables conditioned on the observed inputs $X = [\; x_1 \;\; \cdots \;\; x_N \;]^\top$ and hyperparameter values $\theta$. To predict the value of the unknown function $f_0$ at a test point $x_*$, the following joint distribution

$$\begin{bmatrix} Y \\ f(x_*) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} M_x \\ m(x_*) \end{bmatrix}, \begin{bmatrix} K_{xx} + \sigma_\epsilon^2 I & K_x(x_*) \\ K_x^\top(x_*) & \kappa(x_*, x_*) \end{bmatrix}\right)$$

with $[K_x(x_*)]_i = \kappa(x_i, x_*)$ holds based on the previous considerations. Hence, the predictive distribution for $f(x_*)$, based on the observed samples $\{y_i\}_{i=1}^N$ in $\mathcal{D}_N$, is the posteriori

$$p(f(x_*)|\mathcal{D}_N, x_*) = \mathcal{N}(\mu(x_*), \sigma(x_*))$$

characterized by

$$\mu(x_*) = m(x_*) + K_x^\top(x_*)\left(K_{xx} + \sigma_\epsilon^2 I_N\right)^{-1}(Y - M_x), \tag{2.6a}$$
$$\sigma(x_*) = k(x_*, x_*) - K_x^\top(x_*)\left(K_{xx} + \sigma_\epsilon^2 I_N\right)^{-1} K_x(x_*). \tag{2.6b}$$

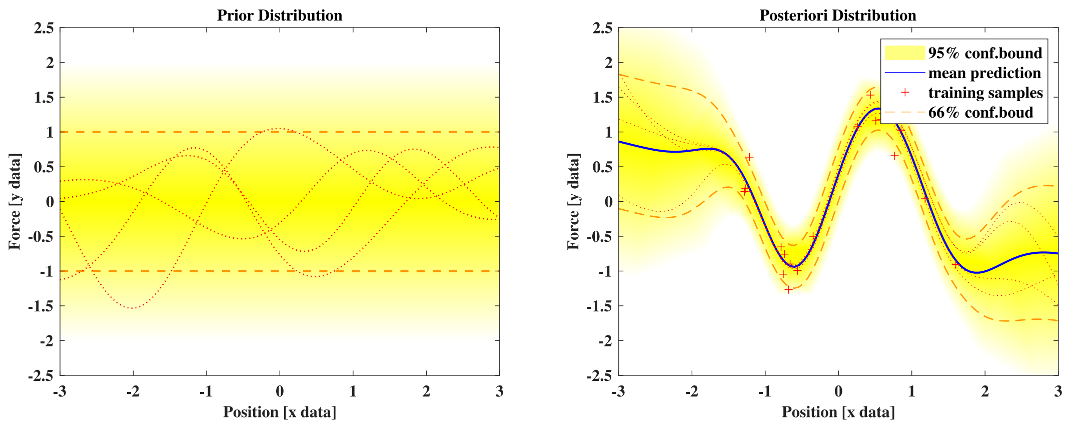


**Figure 2.1:** Example of prior and posterior distributions with zero mean and Squared Exponential covariance function [50].

The mean (2.6a) gives as approximation for $f_0(x_*)$ while the variance (2.6b) gives measure of the uncertainty of this approximation. Computation of (2.6) requires only elementary matrix operations, therefore it is computationally efficient.

The procedure of Bayesian inference is illustrated in Figure 2.1. On the left, a prior distribution is shown with zero mean and Squared Exponential covariance function. The 66% confidence bound is indicated in orange, and sampled functions are drawn in red colour. On the right, the posterior distribution is shown, which is obtained by conditioning the prior on the training samples indicated by red crosses. The mean of the prediction is depicted in blue. The plot shows that the predicted mean is smooth, and precise near the training samples, where the uncertainty is small.

## 2.3 Hyperparameter optimization

To tune the hyperparameters $\theta$ and $\sigma_\epsilon^2$ associated with the prior, a common method is to maximize the likelihood, i.e., probability of the observations of $Y$ for (2.5) marginalized with respect to $\theta$ and $\sigma_\epsilon$:

$$\begin{bmatrix}\theta^* \\ \sigma_\epsilon^*\end{bmatrix} = \arg\max_{\theta,\sigma_\epsilon} \log\left(p(Y|X,\theta,\sigma_\epsilon)\right), \tag{2.7}$$

where without loss of generality, the log of the probability density function is taken to simplify the optimization problem and

$$\log\left(p(Y|X,\theta,\sigma_\epsilon)\right) = -\frac{1}{2}\left(Y^\top(K_{xx}^{-1}+\sigma_\epsilon^2 I_N)Y + \log\det\Big(K_{xx}^{-1}+\sigma_\epsilon^2 I_N\Big) + N\log(2\pi)\Big)\right). \tag{2.8}$$

For alternative methods, see [43]. Here, we considered scalar valued GPs, however, GP regression can be applied under multidimensional outputs, by estimating a predictive distribution of each output dimension independently.

Chapter 3

# Mathematical Model of a Quadcopter

In this chapter, the mathematical model of a quadcopter is presented, mainly based on [35]. Firstly, the coordinate frames are characterized to describe the position and orientation of the drone. The dynamic equations are then written in these frames for translational and rotational motion using multiple attitude representations. Finally, the input equations are described which define a mapping between the control inputs and propeller angular velocities.

## 3.1 Coordinate frames

In order to derive the dynamical model or assign the coordinate frames, first the configuration of the quadcopter has to be fixed. The configuration defines the placement and the direction of rotation of the rotors thus basically determines the structure of the dynamical model and the control strategy. The most commonly used configurations are denoted by '+' and '×', which differ only in the assignment of the front of the vehicle (see Figure 3.1). Due to practical considerations (e.g. camera placement) the earlier is more common, therefore we use '×' configuration in this work.

After the configuration is fixed, the coordinate frames can be clearly defined. Three main frames are introduced: the inertial frame $\mathcal{F}^i$ interpreted as NED (north-east-down) coordinates, the vehicle frame $\mathcal{F}^v$, and the body frame $\mathcal{F}^b$, which is fixed to the vehicle.

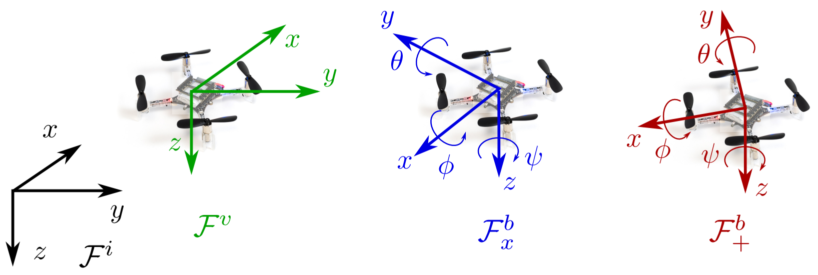


**Figure 3.1:** Inertial, vehicle, and body frames describing the geometric relations of the vehicle and the environment. The body frame is depicted both in '×' and '+' configurations.

The three frames are displayed in Figure 3.1, with the body frame both in '$\times$' and '+' configuration. The transformation from $\mathcal{F}^i$ to $\mathcal{F}^v$ is a translation denoted by $r \in \mathbb{R}^3$, and from $\mathcal{F}^v$ to $\mathcal{F}^b$ a rotation denoted by $R_v^b \in \mathrm{SO}(3)$. The notation SO(3) corresponds to the *special orthogonal group*, which we introduce here briefly, the proper mathematical background of group theory and Lie-groups for rigid body orientation can be found for example in [25, 29].

The *orthogonal group* $\mathrm{O}(n)$ is the group of $n \times n$ orthogonal matrices, with the following property:

$$A \in \mathrm{O}(n) \Leftrightarrow AA^\top = I, \tag{3.1}$$

where $A$ is an $n \times n$ matrix. The column vectors of these matrices are orthogonal, and their determinant is always either 1 or $-1$. The *special orthogonal group* $\mathrm{SO}(n)$ is the subgroup of orthogonal matrices, in which every matrix has determinant 1. These matrices can represent rotations in $n$-dimensional space, therefore they are named rotation matrices.

In order to represent both translations and rotations, $SO(n)$ needs to be extended. Consider the set of all $(n+1) \times (n+1)$ transformation matrices of the form

$$\left\{ \begin{bmatrix} R & r \\ 0 & 1 \end{bmatrix} \;\middle|\; R \in \mathrm{SO}(n) \text{ and } r \in \mathbb{R}^n \right\}. \tag{3.2}$$

The matrix $R$ achieves the rotation, and the vector $r$ the translation. The result is the *special Euclidean group* $\mathrm{SE}(n)$, which is homeomorphic to $\mathbb{R}^n \times \mathrm{SO}(n)$, because the rotation matrix and translation vectors may be chosen independently. Rigid body translation and rotation are described in the 3D space, therefore the representation is in the special Euclidean group SE(3), and the rotation matrix is in the special orthogonal group SO(3).

In Figure 3.1, the rotation is described by the roll, pitch, and yaw (RPY) Euler angles, denoted by $\phi, \theta, \psi \in \mathbb{R}$, respectively. The rotation matrix of the transformation between the vehicle and body frames can be written in terms of the RPY angles as follows:

$$\begin{aligned} R_v^b &= Rot(x,\phi) Rot(z,\theta) Rot(z,\psi) = \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_\phi & S_\phi \\ 0 & -S_\phi & C_\phi \end{bmatrix} \begin{bmatrix} C_\theta & 0 & -S_\theta \\ 0 & 1 & 0 \\ S_\theta & 0 & C_\theta \end{bmatrix} \begin{bmatrix} C_\psi & S_\psi & 0 \\ -S_\psi & C_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} C_\psi C_\theta & C_\theta S_\psi & -S_\theta \\ C_\psi S_\phi S_\theta - C_\phi S_\psi & C_\phi C_\psi + S_\phi S_\theta S_\psi & S_\phi C_\theta \\ S_\phi S_\psi + C_\phi C_\psi S_\theta & C_\phi S_\theta S_\psi - C_\psi S_\phi & C_\phi C_\theta \end{bmatrix}, \end{aligned} \tag{3.3}$$

where $C. = \cos(\cdot)$, $S. = \sin(\cdot)$. The coloumn vectors of the rotation matrix formulate an orthonormal basis of the 3D space, therefore

$$R_b^v = \left(R_v^b\right)^{-1} = \left(R_v^b\right)^\top. \tag{3.4}$$

Using (3.4), the rotation matrix of the transformation from the body to the vehicle frame can be formulated simply by taking the transpose of (3.3).

## 3.2 Translational dynamics

The vehicle can be modelled as a rigid body with mass and inertia in the 3D space, on which gravity field acts. The active movement of the drone is generated by four electric motors and the propellers on them. They produce a collective thrust applied in the center of mass of the vehicle, and three moments around the three coordinate axes.

The translational dynamics of the quadcopter are characterized by Newton's equations of motion, generally expressed as

$$f = m\frac{\mathrm{d}V}{\mathrm{d}t_i}, \tag{3.5}$$

where $f$ is the vector of forces acting on the mass $m$, and $V$ is the velocity of the mass. The notation $t_i$ expresses that the derivative is calculated with respect to the inertial frame.

In our model, there are two forces acting on the vehicle: the collective thrust generated by the motors, and the force of the gravitational field. The collective thrust always points in the negative $z$ direction of the body frame, and gravity always points in the positive $z$ direction of the inertial frame. The force vector in (3.5) results from these terms, as

$$f = R_b^v \begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}, \tag{3.6}$$

where $F$ is the collective thrust of the propellers, and $g$ is the gravitational acceleration.

## 3.3 Rotational dynamics

There are multiple ways to describe the rotational motion of a rigid body in 3D space, three of which are presented in this work. The most common and simple way is to use local coordinates to describe the attitude with three axes, and three rotation angles around them. An example is the RPY angle representation mentioned in Section 3.1, where the three axes are the body $z, y, x$ axes, and the three angles are the yaw ($\psi$), pitch ($\theta$) and roll ($\phi$), respectively. Although this approach is intuitive and commonly used, it has two major drawbacks. The first is that it is only valid in a certain region of the attitude, for example $-\pi/2 \leq \phi \leq \pi/2,\ -\pi/2 \leq \theta \leq \pi/2,\ 0 \leq \psi \leq 2\pi$. The second is the so called *gimbal lock*, meaning that a degree of freedom is lost if two axes become parallel out of the three. In case of the flip maneuver, when the pitch is 90 degrees, the yaw axis is parallel to the roll axis, and the compensation for the yaw changes is not possible. Due to these drawbacks, RPY (or Euler) angles are used only for slower maneuvering and trajectory tracking, but for aggressive maneuvers other, more suitable modelling approaches are needed.

Another possible representation of rigid body rotations is using quaternions. In this case, there is no need to deal with the gimbal lock, and a continuous trajectory can be described by a continuous function of the quaternion elements. However, unit quaternions are ambiguous, as they double-cover the rotation group SO(3), meaning that quaternions $q$ and $-q$ represent the same rotation which causes difficulties in control design. Therefore we use this representation only for reference trajectory design.

The third representation of orientation we describe is the rotation matrix. Any rigid body rotation can be characterized by a $3 \times 3$ orthogonal matrix with determinant 1. These matrices form the configuration space of the orientation of a non-symmetrical object in 3D space, named the special orthogonal group, SO(3). Advantages of this approach are that the product of two rotations is the composition of the rotations, therefore the orientation can be given as the rotation from an initial frame to a current body frame, and it avoids singularities and complexities arising when using local coordinates. The disadvantages of using orientation matrices are that the use of 9 variables (elements of a $3 \times 3$ matrix) is computationally less efficient than 3 angles or 4 quaternion elements, and they are less illustrative.

In this section, the rotational dynamics of the quadrotor is derived using all three mentioned representations, in order to be able to use them for trajectory planning and control design. The translational dynamics from Section 3.2 and the rotational dynamics with one of the three attitude representations together characterize the dynamic model of the quadcopter.

### 3.3.1 Euler angles

The orientation is described with the vector of roll, pitch and yaw angles, $[\phi, \theta, \psi]^\top$. However, since the three rotations are in different frames, the vector of derivatives of the three angles is not equal to the derivative of the vector of the angles. We denote the former $[\dot{\phi}, \dot{\theta}, \dot{\psi}]^\top$, and the latter $[p, q, r]^\top$. The relation of these vectors is described by a matrix transformation, as

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = W \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}, \tag{3.7}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = W^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{3.8}$$

The rotational dynamics can be described by Euler's equations, as

$$\tau = \frac{\mathrm{d}J\omega}{\mathrm{d}t_i}, \tag{3.9}$$

where $\tau$ is the torque acting on the body, $J$ is the inertia, and $\omega$ is the angular velocity. The rotation is easier to express in the body frame, as the $J^b$ inertia is constant, and the angular velocity is $\omega^b = [p, q, r]^\top$. However, as the derivatives in (3.9) are calculated with respect to the inertial frame, the formulae of derivation in a moving frame have to be applied, resulting in

$$\frac{\mathrm{d}J^b\omega^b}{\mathrm{d}t_i} = \frac{\mathrm{d}J^b\omega^b}{\mathrm{d}t_b} + \left(\omega^b \times J^b\omega^b\right) = \underbrace{\frac{\mathrm{d}J^b}{\mathrm{d}t_i}}_{=0}\omega^b + J^b\frac{\mathrm{d}\omega^b}{\mathrm{d}t_i} + \left(\omega^b \times J^b\omega^b\right) = \tau, \tag{3.10}$$

$$\Downarrow$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \left(J^b\right)^{-1}\left(\tau - \omega^b \times J^b\omega^b\right). \tag{3.11}$$

Quadcopters are almost perfectly symmetric, therefore the off-diagonal terms of the inertia matrix are usually neglected. Thus the equation for the rotational acceleration results in

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{J_y - J_z}{J_x} \cdot q \cdot r + \frac{\tau_x}{J_x} \\ \frac{J_z - J_z}{J_x} \cdot p \cdot r + \frac{\tau_y}{J_y} \\ \frac{J_x - J_y}{J_z} \cdot p \cdot q + \frac{\tau_z}{J_z} \end{bmatrix}. \tag{3.12}$$

The drone as a rigid body in 3D space has 12 states: the position, velocity, rotation and rotational velocity. The equations for the velocity are from (3.3) and (3.6), for the rotation from (3.8), and for the rotational velocity from (3.12). Gathering these to a matrix form, the following state space representation is formulated:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{V}_x \\ \dot{V}_y \\ \dot{V}_z \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} V_x \\ V_y \\ V_z \\ -(\sin\phi \cdot \sin\psi + \cos\phi \cdot \cos\psi \cdot \sin\theta)\frac{F}{m} \\ -(\cos\phi \cdot \sin\psi \cdot \sin\theta - \cos\psi \cdot \sin\phi)\frac{F}{m} \\ -(\cos\phi \cdot \cos\theta)\frac{F}{m} + g \\ p + \sin\phi \cdot \tan\theta \cdot q + \cos\phi \cdot \tan\theta \cdot r \\ \cos\phi \cdot q - \sin\phi \cdot r \\ \frac{\sin\phi}{\cos\theta} \cdot q + \frac{\cos\phi}{\cos\theta} \cdot r \\ \frac{J_y - J_z}{J_x} \cdot q \cdot r + \frac{\tau_x}{J_x} \\ \frac{J_z - J_x}{J_y} \cdot p \cdot r + \frac{\tau_y}{J_y} \\ \frac{J_x - J_y}{J_z} \cdot p \cdot q + \frac{\tau_z}{J_z} \end{bmatrix}. \tag{3.13}$$

### 3.3.2 Quaternion based model

The second presented approach for rigid body attitude representation is using quaternions. A quaternion is a hyper complex number of rank 4, which can be represented in multiple forms, see for example equation (3.14). The quaternion elements from $q_1$ to $q_3$ are called the vector part of the quaternion, while $q_0$ is the scalar part.

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^\top = \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix} \tag{3.14}$$

Quaternions have a special multiplication operator, denoted by $\otimes$. If $p$ and $q$ represent rotations, $p \otimes q$ represents the combined rotation. Just as the rotation, quaternion product is not commutative. Two possible (equivalent) methods for calculating this product are the following.

$$p \otimes q = \underbrace{p_0 q_0 - \mathbf{p} \cdot \mathbf{q}}_{\text{scalar part}} + \underbrace{p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p} \times \mathbf{q}}_{\text{vector part}}, \tag{3.15}$$

$$p \otimes q = Q(p)q = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} p_0 q_0 - p_1 q_1 - p_2 q_2 - p_3 q_3 \\ p_0 q_1 + p_1 q_0 + p_2 q_3 - p_3 q_2 \\ p_0 q_2 - p_1 q_3 + p_2 q_0 + p_3 q_1 \\ p_0 q_3 + p_1 q_2 - p_2 q_1 + p_3 q_0 \end{bmatrix}. \tag{3.16}$$

Rotations are defined by unit quaternion products, i.e. the definition of a norm is necessary, as

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}. \tag{3.17}$$

The complex conjugate of a quaternion will also be needed, defined as

$$q^* = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{bmatrix}^\top . \tag{3.18}$$

Given that the angular velocity is in the body frame, the derivative of the quaternion rotation in the dynamic equations has the form [19]

$$\dot{q} = -\frac{1}{2}\omega^b \otimes q. \tag{3.19}$$

It is worth noting that if we apply the quaternion product to a $\mathbb{R}^3$ vector and a quaternion, the scalar part of the vector is always considered zero.

The rotation in (3.6) can be characterized by two quaternion products, namely

$$f = q \otimes \begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} \otimes q^* + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}, \tag{3.20}$$

where $q$ is the rotation quaternion between the inertial and body frame.

The state space representation using (3.13) and the above expressions has the following form:

$$\begin{bmatrix} \dot{r} \\ \dot{V} \\ \dot{q} \\ \dot{\omega}^b \end{bmatrix} = \begin{bmatrix} V \\ q \otimes \frac{1}{m} \begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} \otimes q^* + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \\ -\frac{1}{2}\omega^b \otimes q \\ \left(J^b\right)^{-1} \left(\tau - \omega^b \times J^b \omega^b\right) \end{bmatrix}. \tag{3.21}$$

### 3.3.3 Rotational dynamics on SO(3)

Although it is possible to describe any rotation with quaternions, ambiguities arise when using them to represent the attitude. The third orientation representation method described is the quadcopter model on the special orthogonal group, SO(3).

The attitude dynamics have a simple form using rotation matrices, namely

$$\dot{R}_v^b = R_v^b \hat{\omega}^b, \tag{3.22a}$$

$$\dot{\omega} = \left(J^b\right)^{-1} \left(\tau - \omega^b \times J^b \omega^b\right), \tag{3.22b}$$

where $R_v^b \in \mathrm{SO}(3)$ is the rotation matrix between the vehicle and body frames, and the *hat map* $\hat{\cdot} : \mathbb{R}^3 \to \mathrm{SO}(3)$ is defined by the condition that $\hat{x}y = x \times y$ for all $x, y \in \mathbb{R}^3$. From here on the indices corresponding to the body and vehicle frames from (3.6) and (3.11) are omitted for clarity, the inertia and angular velocity are always defined in the body frame.

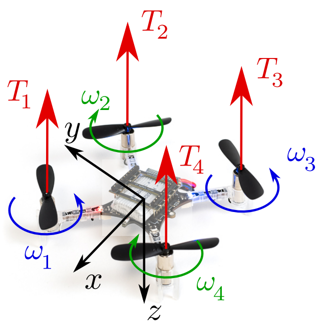


**Figure 3.2:** Thrusts and angular velocities of the rotors with the body frame.

## 3.4 Input equations

The dynamic model has four inputs: the collective thrust, and the torques around the three axes of the body frame. These inputs can be calculated from the individual thrusts and angular speeds of the actuators, the equations of which are described in this section.

The thrust generated by each motor ($T_i$) is proportional to the square of the corresponding angular velocity ($\omega_i$), resulting in the equation

$$T_i = k\omega_i^2, \tag{3.23}$$

where $k$ is the *thrust constant*.

The direction of rotor thrust, angular velocity and the numbering of propellers are displayed in Figure 3.2. The torques around the axes $x$ and $y$ can be calculated as the product of the thrusts $T_i$, and the distance of the motors and the center of mass of the vehicle $l$, as

$$\tau_x = \frac{l}{\sqrt{2}}(T_3 + T_4 - T_1 - T_2) = \frac{l}{\sqrt{2}}k\left(\omega_3^2 + \omega_4^2 - \omega_1^2 - \omega_2^2\right), \tag{3.24}$$

$$\tau_y = \frac{l}{\sqrt{2}}(T_1 + T_4 - T_2 - T_3) = \frac{l}{\sqrt{2}}k\left(\omega_1^2 + \omega_4^2 - \omega_2^2 - \omega_3^2\right). \tag{3.25}$$

The torque around axis $z$ is calculated based on the effect, that every propeller rotates the vehicle in the opposite direction, and this rotating torque is proportional to the square of the angular velocity. Therefore rotors 1 and 3 create positive torque, and the others negative, resulting in the equation

$$\tau_z = b\left(\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2\right), \tag{3.26}$$

where $b$ is the *drag constant*. We can write the input equations (3.23)–(3.26) in a matrix form, resulting in

$$\begin{bmatrix} F \\ \tau \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -\frac{l}{\sqrt{2}} & -\frac{l}{\sqrt{2}} & \frac{l}{\sqrt{2}} & \frac{l}{\sqrt{2}} \\ \frac{l}{\sqrt{2}} & -\frac{l}{\sqrt{2}} & -\frac{l}{\sqrt{2}} & \frac{l}{\sqrt{2}} \\ \frac{b}{k} & -\frac{b}{k} & \frac{b}{k} & -\frac{b}{k} \end{bmatrix} \begin{bmatrix} k\omega_1^2 \\ k\omega_2^2 \\ k\omega_3^2 \\ k\omega_4^2 \end{bmatrix}. \tag{3.27}$$

Since there is a constant mapping between the motor angular velocity and the vector of collective thrust and moments, we consider the $\begin{bmatrix} F, & \tau^\top \end{bmatrix}^\top$ vector the input of the system.

The identification of the input parameters $k$ and $b$ are important for accurate simulation-to-reality transfer, however, it is often difficult due to the complex aerodynamic effects, such as blade flapping and downwash. These parameters are usually determined experimentally. It is important to note, that precise modelling of the actuation mechanism often requires more complex dynamical models than (3.23) and (3.26) [20]. In the methods presented we use only the simplified model, although we point out in Chapter 6 that increasing the accuracy of the model improves the performance of the control algorithms.

## 3.5 Conclusions

In this chapter, we presented the mathematical model of a standard quadrotor vehicle. The translational and rotational dynamics are characterized by (3.6) and (3.11), respectively, which were derived based on the Newton-Euler equations of motion. Using the presented formulae, we are to conduct model-based simulation, trajectory planning and controller design detailed later in Chapters 4 and 5.

For the 3D rotation of the vehicle, three representations were introduced, each with its own advantages and disadvantages: Euler angles, unit quaternions, and rotation matrices. In Chapter 4, we use the Euler angle representation to construct the 2D model of the drone. Moreover, in Chapter 5, we apply the rotation matrix representation for model-based control design and the quaternion representation for trajectory planning of the backflip maneuver.

Finally, we described the input equations characterized by (3.27) to provide a mapping between the control inputs of the dynamic model and the angular velocities of the four rotors which can be directly controlled using an electronic speed controller (ESC). We use the input mapping for simulations and also implementation of low-level control algorithms on the real quadcopter, which are presented in Chapters 6 and 7, respectively.

# Chapter 4

# Feedforward Control by Bayesian Optimization

## 4.1 Overview

The flip maneuver can be executed as a 360 degree rotation around the $y$ axis of the quadcopter's body frame, displayed in Figure 3.1. If the corresponding ideal actuation sequence of the individual motors in terms of $T_i$ is computed based on the nominal quadcopter model to execute this maneuver, then the actuation sequence can be implemented on the real quadcopter in terms of feedforward control. However, computation of such an actuation profile is difficult, due to (i) the complexity of the involved optimization problem to find a feasible motion trajectory under the given actuation constraints and (ii) due to unmodelled aerodynamical effects on the quadcopter that can significantly influence the system response.

The feedforward control proposed in [34] solves the problem above by tuning a fixed sequence of parameterized motion primitives via experiments. However, the approximation of the Jacobian matrix in the optimization loop requires many trials on the real drone and careful selection of the measurement data is needed to ensure the numerical convergence. We have modified the original algorithm at several points in order to improve its performance and adapt it to our specific design configuration. First, we tune the parameters of the motion primitives in simulation by using a high fidelity nonlinear model of the drone. Second, the optimization is solved by a Bayesian optimization method, using Gaussian Process surrogate function introduced in Chapter 2. The main advantages of the proposed method are that Bayesian optimization is numerically better conditioned and requires significantly less function evaluations than Jacobi approximation, as the evaluation points are systematically selected. This makes the proposed method computationally more favourable.

Unlike the maneuver demonstrated in [34], we perform the backflip in '$\times$' configuration as two rotors can produce a larger torque than one. The desired trajectory of the flip motion is within the $x - z$ plane of the body frame (illustrated in Figure 4.1), therefore the equations of motion (5.1) can be simplified utilizing that the translation along the $y$ axis and rotation around the $x, z$ axes are constant zero. In this section, we use Euler angle representation: the backflip is executed as a 360 degree rotation of the pitch angle ($\theta$), while the roll ($\phi$) and yaw ($\psi$) remain zero.

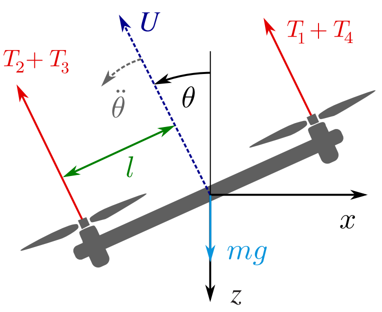


**Figure 4.1:** The vehicle frame, the orientation and forces acting on the quadcopter in two dimensions.

The simplified equations of motion are as follows:

$$m\ddot{x} = -(T_1 + T_2 + T_3 + T_4)\sin\theta, \tag{4.1a}$$
$$m\ddot{z} = -(T_1 + T_2 + T_3 + T_4)\cos\theta + mg, \tag{4.1b}$$
$$J_{\mathrm{yy}}\ddot{\theta} = l(T_1 + T_4 - T_2 - T_3), \tag{4.1c}$$

where $g$ is the gravitational acceleration, $m$ is the mass of the drone, $l$ is the distance of each propeller from the center of mass of the vehicle, $J_{\mathrm{yy}}$ is the moment of inertia about the body $y$ axis, and $T_1, T_2, T_3, T_4$ are the thrusts of each rotor. The direction of the thrusts and pitch are illustrated in Figure 4.1. The collective acceleration $U$ is the sum of all rotor thrusts divided by the mass of the drone:

$$U = \frac{T_1 + T_2 + T_3 + T_4}{m}. \tag{4.2}$$

As the small DC motors of the Crazyflie 2.1 have very small time constants the actuator dynamics are omitted.

## 4.2 Parametrized primitive of the maneuver

The goal of the maneuver is to perform a flip, therefore the states at the start $t_0$ and end $t_{\mathrm{f}}$ should be the same, except for the pitch angle which is shifted by $2\pi$. From these considerations, the initial and final state conditions for the maneuver can be formulated as follows:

$$x(t_0) = x(t_{\mathrm{f}}) = 0, \tag{4.3a}$$
$$z(t_0) = z(t_{\mathrm{f}}) = 0, \tag{4.3b}$$
$$\dot{x}(t_0) = \dot{x}(t_{\mathrm{f}}) = \dot{z}(t_0) = \dot{z}(t_{\mathrm{f}}) = 0, \tag{4.3c}$$
$$\theta(t_{\mathrm{f}}) = \theta(t_0) + 2\pi = 0. \tag{4.3d}$$

We use a simple control strategy without explicit optimization for execution time. The motion consists of five sections, all of which has two constant control inputs, the desired collective acceleration $U_{\mathrm{des}}$, and desired angular acceleration $\ddot{\theta}_{\mathrm{des}}$. These five steps are illustrated in Figure 4.2, and defined as follows.

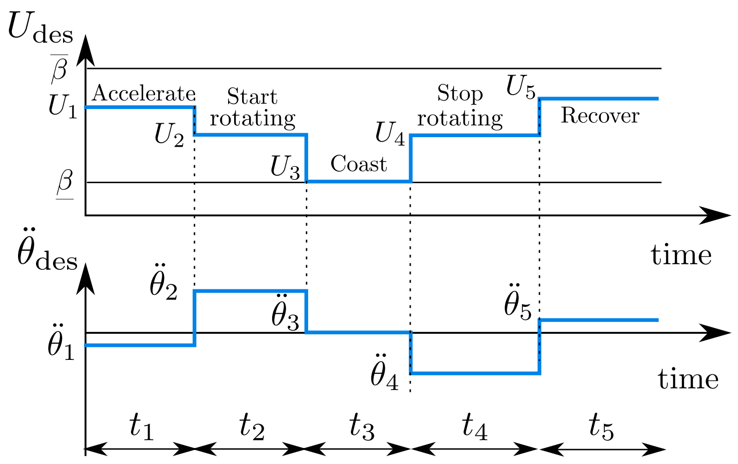


**Figure 4.2:** Parametrized flip primitive with the five sections.

1. **Accelerate:** Gain elevation and kinetic energy with near-maximal collective acceleration, while rotating slowly to the negative direction.
2. **Start Rotate:** Increase angular velocity with torque, i.e. maximal differential thrusts.
3. **Coast:** With low and uniform thrusts hold the angular velocity, wait for the drone to rotate.
4. **Stop Rotate:** Use maximal differential thrust to decrease angular velocity and stop the rotation.
5. **Recover:** Prevent the drone from falling to the ground by applying near-maximal collective thrust, and try to get back to hover mode.

Each of the five sections has three parameters, the collective acceleration $U_i$, duration $t_i$, and angular acceleration $\ddot{\theta}_i$, resulting in 15 parameters altogether. However, based on [34], we can reduce the number of parameters by applying bang-bang type control on a restricted control envelope. The bang-bang approach means that the control actions $U_i$ and $\ddot{\theta}_i$ are either zero or near-maximal at all sections. Following the equations detailed in [34], 10 parameters out of 15 can be expressed using the properties of bang-bang control. The remaining 5 independent parameters are optimized to perform the desired backflip motion. The vector of these parameters is characterized as follows:

$$\eta = \begin{bmatrix} U_1 & t_1 & t_3 & U_5 & t_5 \end{bmatrix}^\top \in \mathbb{R}^5_+. \tag{4.4}$$

From the elements of $\eta$ and the corresponding equations detailed in [34], the control input vector can be calculated as

$$\begin{bmatrix} F_i \\ \tau_i \end{bmatrix} = \begin{bmatrix} mU_i \\ 0 \\ J_{\text{yy}}\ddot{\theta}_i \\ 0 \end{bmatrix} \qquad i \in \{1, 2, 3, 4, 5\}, \tag{4.5}$$

where $m$ is the mass of the quadcopter, and the index $i$ is obtained from the current time and the duration of the sections.

The five parameters are tuned in order to minimize the norm of the final state error $e(\eta) \in \mathbb{R}^5_+$. Formally, the optimization problem is characterized as follows:

$$\begin{aligned}
\underset{\eta \in \mathbb{R}^5_+}{\text{minimize}} \quad & \|e(\eta)\|_2 \\
\text{s.t.} \quad & e(\eta) = [\ x(t_\text{f})\ z(t_\text{f})\ \dot{x}(t_\text{f})\ \dot{z}(t_\text{f})\ \theta(t_\text{f})\ ]^\top, \\
& x(t_0) = z(t_0) = \dot{x}(t_0) = \dot{z}(t_0) = 0, \\
& U_\text{min} \le U_i \le U_\text{max} \quad i \in \{1, 5\}, \\
& t_\text{min} \le t_j \le t_\text{max} \quad j \in \{1, 3, 5\},
\end{aligned} \tag{4.6}$$

where the bounds $U_\text{min}$, $U_\text{max}$, $t_\text{min}$ and $t_\text{max}$ are determined based on the parameters of the quadcopter.

## 4.3 Bayesian parameter optimization

In [34], the numerical optimization of $\eta$ in (4.4) is based on iterative optimization, using the approximate Jacobian matrix of the final state error w.r.t the parameter vector. However, the numerical gradient approximation has vast computational cost, because the whole maneuver needs to be simulated in every approximation step and it suffers from convergence problems. Here, we apply a Bayesian optimization approach to find the global optimum of (4.6). This approach does not require the calculation of derivatives and it is suitable for global optimization of cost functions that are expensive to evaluate, see [18, 7]. To apply this approach, the optimization problem (4.6) is written as

$$\underset{x \in \mathcal{X}}{\text{maximize}}\, f(x), \tag{4.7}$$

where $\eta = x \in \mathbb{R}^n$ is the $n = 5$ dimensional vector of optimization variables, $\mathcal{X}$ is the feasible parameter set (bounded interval (hyper rectangle) of the search space for the parameters $\eta$), and $f$ is the objective function, i.e., $f(x) = -\|e(\eta = x)\|_2$.

The core concept of Bayesian optimization is to evaluate the unknown objective function at limited number of points, giving $\mathcal{D}_N = \{y_i = f(x_i), x_i\}_{i=1}^N$, fit a surrogate parametric model based on GP regression on the data and optimize this surrogate model of the original objective function [43]. The optimization is performed iteratively, where in each step the next evaluation point is determined by minimizing a so called *acquisition function*, the objective function is evaluated at this point and the surrogate model is updated. The optimization stops if the minimum is reached with high confidence or the iteration reaches a certain number of evaluations.

The acquisition function blends the approximated objective (the mean of the GP) and the approximation uncertainty (the variance of the GP) in a scalar valued function that can be optimized by standard gradient-based procedure. A common acquisition function is *expected improvement*, defined as follows. After $f$ is evaluated in $N$ points giving the observation data set $\mathcal{D}_N$, a GP predictive distribution $\hat{f}_N \sim \mathcal{GP}(\mu_N, \sigma_N)$ is obtained w.r.t. $\mathcal{D}_N$. Let $\bar{f}_N^+$ be the value of the best sample so far and $x_N^+ = \arg\max_{x \in X_N} \mu^{(N)}(x)$ be the location of that sample based on $X_N = \{x_i\}_{i=1}^N$, i.e., $\bar{f}_N^+ = \mu^{(N)}(x_N^+)$. Now we would like to choose the next test point $x_{N+1}$, such that the expected improvement predicted by the GP model is the best w.r.t. $x_N^+$:

$$\text{EI}_N(x) := \mathbb{E}\left\{ \lfloor \hat{f}_N(x) - \bar{f}_N^+ \rfloor \right\}, \tag{4.8}$$

where $\lfloor y \rfloor = \max(y, 0)$. The right hand side of (4.8) has an analytical form and the next test point is obtained via $x_{N+1} = \arg\max_{x\in\mathcal{X}} \mathrm{EI}_N(x)$, the point with highest expected improvement. In the literature, there are other common acquisition functions, e.g. upper confidence bound or knowledge gradient [18]. We summarize the steps of Bayesian optimization using a GP surrogate model in Algorithm 1, based on [18].

**Algorithm 1** High-level steps of Bayesian optimization

```
$f$ is evaluted at $N > 0$ initial points, providing $\mathcal{D}_N$
Set a GP prior on $f$ in terms of $\hat{f} \sim \mathcal{GP}(m, \kappa)$
while $N \leq N_{\max}$ do
    Determine $\hat{f}_N \sim \mathcal{GP}(\mu_N, \sigma_N)$ via (2.6) and (2.7) w.r.t. $\mathcal{D}_N$
    Let $x_{N+1} = \arg\max_{x\in\mathcal{X}} \mathrm{EI}_N(x)$,
    Observe $y_{N+1} = f(x_{N+1})$
    $\mathcal{D}_{N+1} = \mathcal{D}_N \cup \{y_{N+1}, x_{N+1}\}$
    $N \leftarrow N + 1$
end while
return $x_* = \arg\max_{x\in\mathcal{X}} \mu^{(N)}(x)$,
```

## 4.4 Conclusions

In this section, we introduced a Bayesian optimization based feedforward control approach to perform a backflip maneuver with quadcopters. Using the nonlinear model of the drone detailed in Chapter 3, it is possible to simulate the maneuver with different parameter sets, evaluate the objective function of the optimization problem characterized by (4.6), and obtain a solution. Then, the optimized parameter set can be directly used to execute the motion primitive sequence illustrated in Figure 4.2, and perform the backflip maneuver.

For the implementation of the backflip, a stabilizing feedback controller is also required to balance the quadcopter at the beginning and after the end of the maneuver. Set point stabilization is a common task in quadcopter control, hence a classical LTI feedback controller, designed for the linearized dynamics around hovering, for example PID or LQR, is suitable for this purpose [41].

It is important to note that parameter uncertainties are the bottleneck of open-loop control strategies, it is essential to have an accurate simulation model which can be adapted to the real system in case the dynamic behaviour of the vehicle changes. In the next section, a nonlinear geometric control approach is proposed that is more robust to parameter uncertainties due to feedback, therefore does not require such complex and accurate modelling of the quadcopter dynamics.

# Chapter 5

# Trajectory Planning and Geometric Control

## 5.1 Overview

The second proposed approach for quadcopter backflipping is based on closed-loop control. For this, first a novel robust adaptive reference tracking controller is developed in order to provide reliable tracking even in the case of modelling uncertainties and external disturbances. Second, a feasible reference trajectory is designed for the backflip maneuver that describes the intended state evolution. An important advantage of this approach compared to the feedforward control is generality, i.e. the proposed trajectory design method can be applied to a variety of different maneuvers, while the presented feedforward approach is designed specifically to the backflip maneuver.

The low-level motion control of quadrotors is a widely researched area. The most simple and common approach is linear time invariant (LTI) control, e.g. PID or LQR [5]. A more sophisticated algorithm is feedback linearization, which can be applied to quadrotors due to the differential flatness of the system dynamics [28]. However, geometric control proposed in [30] is designed directly on the configuration manifold of quadcopters (SE(3)), resulting in exponential convergence of the attitude and position tracking errors, hence provides the performance required for aggressive, agile maneuvers. In this chapter, first we introduce geometric tracking control of quadrotors. Next, we propose a robust adaptive geometric control law for trajectory tracking of aggressive maneuvers, and then an optimization-based trajectory planning method.

## 5.2 Geometric tracking control for agile maneuvering

We introduce a control method that is able to track reference position $r_\mathrm{d}(t) = [x_\mathrm{d}(t), y_\mathrm{d}(t), z_\mathrm{d}(t)]^\top$, and reference attitude $R_\mathrm{d}(t) \in \mathrm{SO}(3)$, represented by rotation matrices. To synthesize the control law, (3.6) and (3.22) are used, describing the dynamics of the quadcopter on SE(3), gathered here for clarity:

$$m\ddot{r} = (-FR + mg)e_3, \tag{5.1a}$$

$$J\dot{\omega} = \tau - \omega \times J\omega, \tag{5.1b}$$

$$\dot{R} = R\hat{\omega}. \tag{5.1c}$$

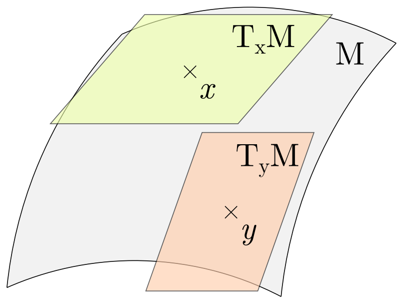


**Figure 5.1:** Visual representation of a manifold M with two tangent spaces: $T_xM$ at point $x$ and $T_yM$ at point $y$. It is clear that a vector in $T_xM$ and another in $T_yM$ can not be compared unless we transform them to the same space.

The tracking errors for the position and velocity are given by

$$e_r = r - r_d, \tag{5.2a}$$
$$e_v = v - v_d. \tag{5.2b}$$

The attitude and angular velocity tracking errors are chosen such that they evolve on the tangent bundle of the configuration manifold SO(3). For this, we introduce the attitude error function proposed in [30], as follows:

$$\Psi(R, R_d) = \frac{1}{2}\mathrm{tr}\left(I - R_d^\top R\right), \tag{5.3}$$

where $\mathrm{tr}(\cdot)$ is the trace operator, and $I$ is the $3 \times 3$ identity matrix. The error function (5.3) is locally positive definite about $R_d^\top R = I$ within the region where the angle of the current and reference attitudes is less than $\pi$. Using the identity $-\frac{1}{2}\mathrm{tr}(\hat{x}\hat{y}) = x^\top y$, the derivative of the error function is given by

$$\mathrm{D_R}\Psi(R, R_d) \cdot R\hat{\eta} = \frac{1}{2}\left(R_d^\top R - R^\top R_d\right)^\vee \cdot \eta, \tag{5.4}$$

where the variation of the rotation matrix is expressed as $\delta R = R\hat{\eta}$ for $\eta \in \mathbb{R}^3$, and the *vee operator* $(\cdot)^\vee : \mathrm{SO}(3) \to \mathbb{R}^3$ is the inverse of the hat operator $\hat{(\cdot)}$ introduced in Section 3.3.3. From this, the attitude tracking error is defined as

$$e_R = \frac{1}{2}\left(R_d^\top R - R^\top R_d\right)^\vee. \tag{5.5}$$

The tangent vector of the current and reference rotations $\dot{R} \in \mathrm{T_R SO(3)}$ and $\dot{R}_d \in \mathrm{T_{R_d} SO(3)}$ lie in different tangent spaces (see the illustration in Figure 5.1), therefore they cannot be directly compared to calculate the angular velocity error. This is addressed by transforming $\dot{R}_d$ into a vector in $\mathrm{T_R SO(3)}$, and comparing it with $\dot{R}$, as follows [30]:

$$\dot{R} - \dot{R}_d(R_d^\top R) = R\hat{\omega} - R_d\hat{\omega}_d R_d^\top R = R(\omega - R^\top R_d\omega_d)^\wedge. \tag{5.6}$$

The angular velocity tracking error can be now defined as

$$e_\omega = \omega - R^\top R_d\omega_d, \tag{5.7}$$

which is the angular velocity of the rotation matrix $R_d^\top R$ represented in the body-fixed frame.

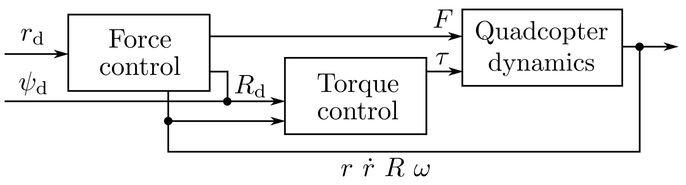


**Figure 5.2:** Block diagram of the geometric feedback control.[31]

The control laws for the force and torque inputs are defined as

$$F = (-k_\mathrm{r} e_\mathrm{r} - k_\mathrm{v} e_\mathrm{v} - mge_3 + m\ddot{r}_\mathrm{d})^\top R e_3, \tag{5.8a}$$

$$\tau = -k_\mathrm{R} e_\mathrm{R} - k_\omega e_\omega + \omega \times J\omega - J\left(\hat{\omega} R^\top R_\mathrm{d} \omega_\mathrm{d} - R^\top R_\mathrm{d} \dot{\omega}_\mathrm{d}\right), \tag{5.8b}$$

with controller gains $k_\mathrm{r}, k_\mathrm{v}, k_\mathrm{R}, k_\omega \in \mathbb{R}$. In [30], it is proven that using the control law (5.8), the zero equilibrium of the tracking errors of the compete dynamics is exponentially stable, given that the desired trajectory satisfies

$$\| -mge_3 + m\ddot{r}_\mathrm{d}\| < B \tag{5.9}$$

for a given positive constant $B$, and the initial conditions satisfy

$$\begin{aligned} &\Psi(R(0), R_\mathrm{d}(0)) \le \psi_1 < 1, \\ &\|e_\omega(0)\|^2 < \frac{2}{\lambda_\mathrm{min}(J)} k_\mathrm{R} \left(1 - \Psi\left(R(0), R_\mathrm{d}(0)\right)\right) \end{aligned} \tag{5.10}$$

for a fixed constant $\psi_1$, where $\lambda_\mathrm{min}(J)$ denotes the smallest eigenvalue of the inertia matrix. The stability is proven by constructing positive definite Lyapunov functions for the attitude and position dynamics separately, and ensuring that the time derivative of the Lyapunov candidates are negative within the stable region of the control gains.

The geometric control has been originally developed and is mainly used for tracking a prescribed 3D path. The associated reference trajectory is usually defined by utilizing the *differential flatness* property of quadcopter dynamics [16]. The dynamic model presented in Chapter 3 is underactuated as it has six degrees of freedom and only four control inputs. However, the model is differentially flat, i.e., all state variables and control inputs can be expressed from a finite number of time derivatives of four flat outputs, namely the position $x, y, z$ and yaw angle $\psi$. In a classical tracking problem, reference trajectories are constructed for the flat outputs, as it is illustrated in Figure 5.2. From this, the attitude reference $R_\mathrm{d} = [r_1, r_2, r_3]$ can be obtained as follows [49]:

$$r_1 = r_2 \times r_3, \tag{5.11a}$$

$$r_2 = \frac{r_3 \times [\cos\psi_\mathrm{d}, \sin\psi_\mathrm{d}, 0]}{\|r_3 \times [\cos\psi_\mathrm{d}, \sin\psi_\mathrm{d}, 0]\|}, \tag{5.11b}$$

$$r_3 = \frac{-k_\mathrm{r} e_\mathrm{r} - k_\mathrm{v} e_\mathrm{v} + mge_3 + m\ddot{r}_\mathrm{d}}{\|-k_\mathrm{r} e_\mathrm{r} - k_\mathrm{v} e_\mathrm{v} + mge_3 + m\ddot{r}_\mathrm{d}\|}. \tag{5.11c}$$

Using the orientation reference in (5.11), the third coloumn vector $r_3$ always points in the direction of the desired collective thrust, $r_2$ is always perpendicular to both $r_3$ and the desired heading direction given by $[\cos\psi_\mathrm{d}, \sin\psi_\mathrm{d}, 0]$, and $r_1$ is the desired heading direction, as it is perpendicular to both $r_2$ and $r_3$. Furthermore, the three coloumn vectors are of unit length at all time instants, ensuring the orthogonality of $R_\mathrm{d}$ altogether. However, during the flip maneuver we intend to specify $R_\mathrm{d}$ from the pitch reference, therefore we do not use (5.11), but design the trajectory such that the reference attitude is consistent with the reference position, as they are coupled in the dynamical model.

## 5.3 Robust adaptive geometric control by Gaussian Processes

The geometric control introduced in Section 5.2 is based on an ideal model of the quadcopter dynamics, therefore can lose performance or even stability in case of model uncertainties and external disturbances. To address this problem, the nominal control law is augmented with additional robust control terms in [31] to guarantee uniformly ultimately bounded tracking errors in the presence of bounded disturbances. However, the control design requires a priori knowledge of the magnitude of external disturbances which can be challenging in unknown situations and environments or lead to poor performance.

In Chapter 2, we have introduced Gaussian Processes as universal function approximators from measurement data. In this section, we propose a novel robust adaptive geometric controller, based on the identification of modelling uncertainties in terms of Gaussian Processes.

To synthesize the control law, we use (5.1) describing the dynamics of the quadcopter and augment it by additive state-dependent disturbance, resulting in

$$m\ddot{r} = mge_3 - FRe_3 + \Delta_{\mathrm{r}}(\chi), \tag{5.12a}$$

$$J\dot{\omega} = \tau - \omega \times J\omega + \Delta_{\mathrm{R}}(\chi), \tag{5.12b}$$

where $\chi = [r,\ \dot{r},\ R,\ \omega]^\top$ is the quadcopter state, and $\Delta_{\mathrm{r}}(\chi), \Delta_{\mathrm{R}}(\chi) \in \mathbb{R}^3$ are the unknown disturbances.

We augment the control law proposed in [31] with adaptive state-dependent terms $\eta_{\mathrm{r}}, \eta_{\mathrm{R}}$, as follows:

$$F = (-k_{\mathrm{r}}e_{\mathrm{r}} - k_{\mathrm{v}}e_{\mathrm{v}} - mge_3 + m\ddot{r}_{\mathrm{d}} - \eta_{\mathrm{r}} + \mu_{\mathrm{r}})^\top Re_3, \tag{5.13a}$$

$$\begin{aligned}\tau =& - k_{\mathrm{R}}e_{\mathrm{R}} - k_{\Omega}e_{\Omega} + \omega \times J\omega - \\ & J\left(\hat{\Omega}R^\top R_d\Omega_{\mathrm{d}} - R^\top R_{\mathrm{d}}\dot{\Omega}_{\mathrm{d}}\right) - \eta_{\mathrm{R}} + \mu_{\mathrm{R}},\end{aligned} \tag{5.13b}$$

with controller gains $k_{\mathrm{r}}, k_{\mathrm{v}}, k_{\mathrm{R}}, k_{\omega} \in \mathbb{R}$, and error terms $e_{\mathrm{r}}, e_{\mathrm{v}}, e_{\mathrm{R}}, e_{\omega} \in \mathbb{R}^3$ defined in Section 5.2, gathered here for clarity:

$$e_{\mathrm{r}} = r - r_{\mathrm{d}}, \tag{5.14a}$$

$$e_{\mathrm{v}} = \dot{r} - \dot{r}_{\mathrm{d}}, \tag{5.14b}$$

$$e_{\mathrm{R}} = \frac{1}{2}\left(R_{\mathrm{d}}^\top R - R^\top R_{\mathrm{d}}\right)^\vee, \tag{5.14c}$$

$$e_{\omega} = \omega - R^\top R_{\mathrm{d}}\omega_{\mathrm{d}}. \tag{5.14d}$$

We identify the external disturbances from noisy observations using Gaussian Processes, more specifically in the form

$$\hat{\Delta}_{\mathrm{r}} = \mathcal{GP}_{\mathrm{r}}(\chi) \sim \mathcal{N}(\eta_{\mathrm{r}}(\chi), \Sigma_{\mathrm{r}}(\chi)), \tag{5.15a}$$

$$\hat{\Delta}_{\mathrm{R}} = \mathcal{GP}_{\mathrm{R}}(\chi) \sim \mathcal{N}(\eta_{\mathrm{R}}(\chi), \Sigma_{\mathrm{R}}(\chi)). \tag{5.15b}$$

We assume that the GPs are trained and the true disturbances $\Delta_{\mathrm{r}}(\chi), \Delta_{\mathrm{R}}(\chi)$ are inside the 95% confidence interval of the GPs.

Subsequently, we formulate the robust terms of the control law similarly to [31], as

$$\mu_{\mathrm{r}} = -\frac{\delta_{\mathrm{r}}^{\tau+2} e_{\mathrm{B}} \left\| e_{\mathrm{B}} \right\|^{\tau}}{\delta_{\mathrm{r}}^{\tau+1} \left\| e_{\mathrm{B}} \right\|^{\tau+1} + \epsilon_{\mathrm{r}}^{\tau+1}}, \tag{5.16a}$$

$$e_{\mathrm{B}} = e_{\mathrm{v}} + \frac{c_1}{m} e_{\mathrm{r}}, \tag{5.16b}$$

$$\mu_{\mathrm{R}} = -\frac{\delta_{\mathrm{R}}^{2} e_{\mathrm{A}}}{\delta_{\mathrm{R}} \left\| e_{\mathrm{A}} \right\| + \epsilon_{\mathrm{R}}}, \tag{5.16c}$$

$$e_{\mathrm{A}} = e_{\Omega} + c_2 J^{-1} e_{\mathrm{R}}, \tag{5.16d}$$

where $c_1, c_2, \epsilon_{\mathrm{r}}, \epsilon_{\mathrm{R}}, \tau$ are positive constants, $\tau > 2$, and $\|\cdot\|$ is the Euclidean vector norm. However, instead of estimating the uncertainty bounds $\delta_{\mathrm{r}}, \delta_{\mathrm{R}}$ as in [31], we utilize the uncertainty of the corresponding trained Gaussian Process. We calculate the standard deviation of a GP at an evaluation point as the radius of the bounding sphere of the ellipsoid defined by the square root of the covariance matrix, formulated as

$$\sigma_{\mathrm{r}}(\chi) = \|L_{\mathrm{r}}(\chi)\|_2 = \sqrt{\|\Sigma_{\mathrm{r}}(\chi)\|_2}, \quad L_{\mathrm{r}} L_{\mathrm{r}}^{\top} = \Sigma_{\mathrm{r}}, \tag{5.17a}$$

$$\sigma_{\mathrm{R}}(\chi) = \|L_{\mathrm{R}}(\chi)\|_2 = \sqrt{\|\Sigma_{\mathrm{R}}(\chi)\|_2}, \quad L_{\mathrm{R}} L_{\mathrm{R}}^{\top} = \Sigma_{\mathrm{R}}, \tag{5.17b}$$

where $\|\cdot\|_2$ is the $\mathcal{L}_2$-norm (also called spectral norm), computed as the largest singular value of the matrix. We define the uncertainty bounds using the 95% confidence interval of the normal distribution, namely

$$\hat{\delta}_{\mathrm{r}}(\chi) = 2\sigma_{\mathrm{r}}(\chi), \quad \hat{\delta}_{\mathrm{R}}(\chi) = 2\sigma_{\mathrm{R}}(\chi). \tag{5.18}$$

From the confidence interval, we calculate an ultimate bound for the difference between the disturbances and the adaptive terms over the operating domain, as follows:

$$\delta_{\mathrm{r}} = \max_{\chi} \hat{\delta}_{\mathrm{r}}(\chi), \quad \delta_{\mathrm{R}} = \max_{\chi} \hat{\delta}_{\mathrm{R}}(\chi). \tag{5.19}$$

**Theorem 1 (Uniform ultimate boundedness of tracking errors).** *Consider the control force $F$ and torque $\tau$ defined at* (5.13)*. Suppose that the initial condition satisfies*

$$\Psi(R(0), R_{\mathrm{d}}(0)) < \psi_1 < 1, \tag{5.20}$$

$$\|e_{\mathrm{r}}(0)\| < e_{\mathrm{r}_{\max}}. \tag{5.21}$$

*For the fixed constant $\psi_1$, there exist controller parameters such that all tracking errors are uniformly ultimately bounded.* ▪

**Proof**: Throughout the proof, we adapt the steps of Proposition 3 in [31] to the GP based uncertainty bounds. First, we derive the dynamics of the rotational and translational tracking error and construct Lyapunov functions for them.

Based on (5.12), (5.13), (5.14), the error dynamics can be expressed in the following form:

$$\begin{aligned}
m\dot{e}_{\mathrm{v}} &= mge_3 - m\ddot{r}_{\mathrm{d}} - \frac{F}{e_3^\top R_{\mathrm{d}}^\top R e_3} R_{\mathrm{d}} e_3 - X + \Delta_{\mathrm{r}} \\
&= -k_{\mathrm{r}} e_{\mathrm{r}} - k_{\mathrm{v}} e_{\mathrm{v}} - X + \Delta_{\mathrm{r}} - \eta_{\mathrm{r}} + \mu_{\mathrm{r}}, \qquad (5.22\mathrm{a})\\
X &= \frac{F}{e_3^\top R_{\mathrm{d}}^\top R e_3}((e_3^\top R_{\mathrm{d}}^\top R e_3) R e_3 - R_{\mathrm{d}} e_3), \qquad (5.22\mathrm{b})\\
J\dot{e}_\omega &= \tau + \Delta_R - \omega \times J\omega + J\left(\hat{\omega} R^\top R_{\mathrm{d}} \omega_{\mathrm{d}} - R^\top R_{\mathrm{d}} \dot{\omega}_{\mathrm{d}}\right) \\
&= -k_{\mathrm{R}} e_{\mathrm{R}} - k_\omega e_\omega + \Delta_{\mathrm{R}} - \eta_{\mathrm{R}} + \mu_{\mathrm{R}}. \qquad (5.22\mathrm{c})
\end{aligned}$$

Similar to [31], consider the following Lyapunov function candidates:

$$\mathcal{V}_1 = \frac{1}{2} k_{\mathrm{r}} \|e_{\mathrm{r}}\|^2 + \frac{1}{2} m \|e_{\mathrm{v}}\|^2 + c_1 e_{\mathrm{r}}^\top e_{\mathrm{v}}, \qquad (5.23\mathrm{a})$$

$$\mathcal{V}_2 = \frac{1}{2} e_\omega^\top J e_\omega + k_{\mathrm{R}} \Psi(R, R_{\mathrm{d}}) + c_2 e_{\mathrm{R}}^\top e_\omega. \qquad (5.23\mathrm{b})$$

First, let us focus on $\mathcal{V}_2$. For the attitude error function $\Psi$ a lower and upper bound can be given in terms of the attitude error $e_{\mathrm{r}}$ as follows:

$$\frac{1}{2} \|e_{\mathrm{R}}\|^2 \leq \Psi\left(R, R_{\mathrm{d}}\right) \leq \frac{1}{2 - \psi_{1,\max}} \|e_{\mathrm{R}}\|^2 . \qquad (5.24)$$

The detailed derivation of these bounds can be found in [17]. Note that, (5.24) implies that $\Psi$ is positive definite and decrescent for all $t \in \mathbb{R}_+$. By using (5.24), the following upper bound can be derived for the time derivative of $\mathcal{V}_2$:

$$\dot{\mathcal{V}}_2 \leq -z_2^\top W_2 z_2 + e_{\mathrm{A}}^\top (\Delta_{\mathrm{R}} - \eta_{\mathrm{R}} + \mu_{\mathrm{R}}), \qquad (5.25)$$

$$z_2 = \begin{bmatrix} \|e_{\mathrm{R}}\| \\ \|e_\omega\| \end{bmatrix} \in (\mathbb{R}^2)^{\mathbb{R}_+}, \quad W_2 = \begin{bmatrix} \frac{c_2 k_{\mathrm{R}}}{\lambda_{\mathrm{M}}} & -\frac{c_2 k_\omega}{2\lambda_{\mathrm{m}}} \\ -\frac{c_2 k_\omega}{2\lambda_{\mathrm{m}}} & k_\omega - c_2 \end{bmatrix},$$

where $\lambda_{\mathrm{m}}, \lambda_{\mathrm{M}}$ denote the smallest and largest eigenvalue of the inertia matrix $J$. If the controller parameters are chosen such that $W_2$ is positive definite, then $z_2^\top W_2 z_2$ is non-negative. Now we examine the second term on the r.h.s of (5.25) and construct an upper bound for this term as well. First, note that $\|\Delta_{\mathrm{R}} - \eta_{\mathrm{R}}\| \leq \delta_{\mathrm{R}}$ holds by construction of $\delta_{\mathrm{R}}$, under the assumption that $\Delta_{\mathrm{r}}, \Delta_{\mathrm{R}}$ are inside the 95% confidence interval of the GP distribution. By using this inequality and the definition of the robust control law (5.16), the following upper bound can be obtained:

$$\begin{aligned}
e_{\mathrm{A}}^\top (\Delta_{\mathrm{R}} - \eta_{\mathrm{R}} + \mu_{\mathrm{R}}) &\leq \delta_{\mathrm{R}} \|e_{\mathrm{A}}\| - \frac{\delta_{\mathrm{R}}^2 \|e_{\mathrm{A}}\|^2}{\delta_{\mathrm{R}} \|e_{\mathrm{A}}\| + \epsilon_{\mathrm{R}}} \\
&= \frac{\delta_{\mathrm{R}} \|e_{\mathrm{A}}\|}{\delta_{\mathrm{R}} \|e_{\mathrm{A}}\| + \epsilon_{\mathrm{R}}} \epsilon_{\mathrm{R}} \leq \epsilon_{\mathrm{R}}.
\end{aligned} \qquad (5.26)$$

Therefore

$$\dot{\mathcal{V}}_2 \leq -z_2^\top W_2 z_2 + \epsilon_{\mathrm{R}}, \qquad (5.27)$$

where $\epsilon_{\mathrm{R}}$ is chosen to be a sufficiently small positive constant. According to the proof of Proposition 3 in [31], inequality (5.27) implies that the tracking errors $e_{\mathrm{R}}$ and $e_\omega$ are uniformly ultimately bounded.

Consider now Lyapunov function candidate $\mathcal{V}_1$. Its time derivative can be given as follows:

$$\begin{aligned}\dot{\mathcal{V}}_1 = &-\left(k_{\mathrm{v}}-c_1\right)\|e_{\mathrm{v}}\|^2-\frac{c_1 k_{\mathrm{r}}}{m}\|e_{\mathrm{r}}\|^2-\frac{c_1 k_{\mathrm{v}}}{m} e_{\mathrm{r}}^{\top} e_{\mathrm{v}} \\ &+\left(X+\Delta_{\mathrm{r}}-\eta_{\mathrm{r}}+\mu_{\mathrm{r}}\right)^{\top}\left(\frac{c_1}{m} e_{\mathrm{r}}+e_{\mathrm{v}}\right).\end{aligned} \tag{5.28}$$

Similarly to the case of $\mathcal{V}_2$, the first three terms can be made negative by suitably choosing the parameters of the controller. The last term can be divided into two parts: the first one is $e_{\mathrm{B}}^{\top}(\Delta_{\mathrm{r}}-\eta_{\mathrm{r}}+\mu_{\mathrm{r}})$ and $e_{\mathrm{B}}^{\top}X$ is the second. By using (5.19) and (5.16), an upper bound can be derived for the first term:

$$\begin{aligned}e_{\mathrm{B}}^{\top}(\Delta_{\mathrm{r}}-\eta_{\mathrm{r}}+\mu_{\mathrm{r}}) &\leq \delta_{\mathrm{r}}\|e_{\mathrm{B}}\|-\frac{\delta_{\mathrm{r}}^{\tau+2}\|e_{\mathrm{B}}\|^{\tau+2}}{\delta_{\mathrm{r}}^{\tau+1}\|e_{\mathrm{B}}\|^{\tau+1}+\epsilon_{\mathrm{r}}^{\tau+1}} \\ &=\frac{\delta_{\mathrm{r}}\|e_{\mathrm{B}}\|}{\delta_{\mathrm{r}}^{\tau+1}\|e_{\mathrm{B}}\|^{\tau+1}+\epsilon_{\mathrm{r}}^{\tau+1}}\epsilon_{\mathrm{r}}^{\tau+1} \leq \epsilon_{\mathrm{r}}.\end{aligned} \tag{5.29}$$

To obtain an upper bound for $e_{\mathrm{B}}^{\top}X$, first we construct an upper bound for $X$ by using (5.22b) :

$$\begin{aligned}\|X\| &\leq \|A\|\|(e_3^{\top}R_{\mathrm{d}}^{\top}Re_3)Re_3-R_{\mathrm{d}}e_3\| \qquad (5.30)\\ &\leq (k_{\mathrm{r}}\|e_{\mathrm{r}}\|+k_{\mathrm{v}}\|e_{\mathrm{v}}\|+B+\delta_{\mathrm{r}})\Big\|(e_3^{\top}R_{\mathrm{d}}^{\top}Re_3)Re_3-R_{\mathrm{d}}e_3\Big\|,\end{aligned}$$

where

$$A=-k_{\mathrm{r}}e_{\mathrm{r}}-k_{\mathrm{v}}e_{\mathrm{v}}-mge_3+m\ddot{r}_{\mathrm{d}}-\eta_{\mathrm{r}}+\mu_{\mathrm{r}}. \tag{5.31}$$

Moreover, we assume that the reference trajectory $r_{\mathrm{d}}$ has been designed such that the following condition holds:

$$\|-mge_3+m\ddot{r}_{\mathrm{d}}-\eta_{\mathrm{r}}\|<B, \quad B\in\mathbb{R}_+. \tag{5.32}$$

By using the following relation (see [31] for details),

$$\begin{aligned}&\left\|\left(e_3^{\top}R_{\mathrm{d}}^{\top}Re_3\right)Re_3-R_{\mathrm{d}}e_3\right\| \leq \|e_{\mathrm{R}}\|=\sqrt{\Psi(2-\Psi)} \\ &\leq \left\{\sqrt{\psi_{1,\max}\left(2-\psi_{1,\max}\right)} \triangleq \alpha\right\}<1,\end{aligned} \tag{5.33}$$

the upper bound for $X$ can be expressed as follows:

$$\|X\|\leq (k_{\mathrm{r}}\|e_{\mathrm{r}}\|+k_{\mathrm{v}}\|e_{\mathrm{v}}\|+B+\delta_{\mathrm{r}})\alpha. \tag{5.34}$$

By substituting (5.29) and (5.34) into (5.28), we obtain

$$\begin{aligned}\dot{\mathcal{V}}_1 \leq &-\left(k_{\mathrm{v}}(1-\alpha)-c_1\right)\|e_{\mathrm{v}}\|^2-\frac{c_1 k_{\mathrm{r}}}{m}(1-\alpha)\|e_{\mathrm{r}}\|^2 \\ &+\|e_{\mathrm{R}}\|\left\{(B+\delta_{\mathrm{r}})\left(\frac{c_1}{m}\|e_{\mathrm{r}}\|+\|e_{\mathrm{v}}\|\right)+k_{\mathrm{r}}\|e_{\mathrm{r}}\|\|e_{\mathrm{v}}\|\right\} \\ &+\frac{c_1 k_{\mathrm{v}}}{m}(1+\alpha)\|e_{\mathrm{r}}\|\|e_{\mathrm{v}}\|+\epsilon_{\mathrm{r}}.\end{aligned} \tag{5.35}$$

According to the proof of Proposition 3 in [31], inequality (5.35) implies that the tracking errors $e_{\mathrm{r}}$, $e_{\mathrm{v}}$ are uniformly ultimately bounded as well. This completes the proof. □

As a summary, we augment the control law introduced in [31] with adaptive terms, and calculate the uncertainties systematically, leading to improved control performance with robust stability guarantees.

## 5.4 Trajectory planning for the backflip maneuver

In order to use the geometric tracking controller to perform the flip maneuver, a suitable reference trajectory is needed. Aligned with the controller concept, first an attitude reference trajectory $R_{\mathrm{d}}$ is constructed and then it is completed with a position reference $r_{\mathrm{d}}$. However, since backflipping is a more complex task than a usual path following, the flat outputs are not suitable for describing the reference trajectory. Therefore we will construct the attitude reference $R_{\mathrm{d}}$ directly. Similarly to the feedforward approach, the objective for trajectory planning is that the quadcopter should arrive as close to the starting point as possible, while keeping the control inputs within the allowed range during the maneuver.

The attitude reference is specified in unit quaternions, because it is not possible to design a continuous trajectory for the flip in Euler angles, and rotation matrices are less illustrative. The conversion between Euler angles and unit quaternions is characterized by [19]

$$
\begin{aligned}
q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} &= \begin{bmatrix} \cos(\psi/2) \\ 0 \\ 0 \\ \sin(\psi/2) \end{bmatrix} \begin{bmatrix} \cos(\theta/2) \\ 0 \\ \sin(\theta/2) \\ 0 \end{bmatrix} \begin{bmatrix} \cos(\phi/2) \\ \sin(\phi/2) \\ 0 \\ 0 \end{bmatrix} = \\
&= \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix}.
\end{aligned}
\tag{5.36}
$$

The reference quaternion is $q_{\mathrm{d}} = [q_{0\mathrm{d}}, q_{1\mathrm{d}}, q_{2\mathrm{d}}, q_{3\mathrm{d}}]^\top$, where $q_{0\mathrm{d}}$ is the scalar part of the quaternion, and $q_{2\mathrm{d}}$ corresponds to the pitch angle, as $q_{1\mathrm{d}} = q_{3\mathrm{d}} = 0$, because both the roll and yaw angles are zero during the flip. Utilizing that $q_{\mathrm{d}}$ is a unit quaternion, we can express the third element of it as follows:

$$
q_{2d} = \sqrt{1 - q_{0d}^2}. \tag{5.37}
$$

Hence, it is sufficient to design a trajectory only for $q_{0\mathrm{d}} = \cos(\theta/2)$.

A 360 degree rotation around the $y$ axis means that the scalar part of the attitude quaternion goes from 1 to -1. In the trajectory design it is important to stay within the $q_{0\mathrm{d}} \in [-1, 1]$ range, because only unit quaternions describe rotation. We have chosen a smooth sigmoid function

$$
q_{0\mathrm{d}} = \frac{2}{1 + e^{-v_{\mathrm{m}}\left(t - \frac{t_{\mathrm{m}}}{2}\right)}} - 1 \tag{5.38}
$$

to describe the scalar part of the reference attitude, where the parameters are the speed of the maneuver $v_{\mathrm{m}}$ and the execution time $t_{\mathrm{m}}$. The attitude quaternion reference trajectory is displayed in Figure 5.3. Assuming that $\phi \equiv \psi \equiv 0$ during the flip, the conversion to Euler angles yields $\theta = 2\arccos(q_{0\mathrm{d}})$, where $\theta \in [-\pi, \pi]$. Hence the pitch angle goes smoothly from zero to $\pi$, jumps to $-\pi$, and goes smoothly to zero. Besides of rotation, the maneuver also requires translational motion, because without proper lifting at the beginning of the backflip, the quadcopter would fall to the ground due to gravity. The

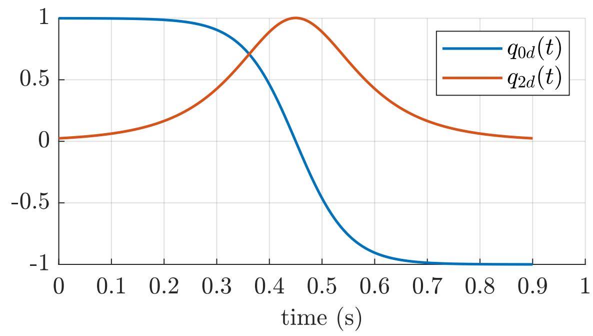


**Figure 5.3:** Attitude quaternion reference trajectory for the backflip maneuver with $v_\mathrm{m} = 20$ 1/s, $t_\mathrm{m} = 0.9$ s.

position reference is designed considering that the rotational and translational equations of the dynamical model are coupled. The translational motion of the flip maneuver is within the $x - z$ plane, therefore $y_\mathrm{d}(t) = 0$. The other two equations of the translational dynamics in (5.1) are

$$m\ddot{x} = -FR_{13}, \tag{5.39a}$$

$$m\ddot{z} = -FR_{33} + mg, \tag{5.39b}$$

where $R_{ij}$ denotes the $(i, j)$-th entry of the rotation matrix $R$. However, assuming that the attitude tracking converges fast enough to the reference, we can substitute the reference rotation matrix in (5.39), resulting in the translational state space representation

$$\dot{\zeta} = A\zeta + Bu, \tag{5.40}$$

$$\zeta = \begin{bmatrix} x \\ \dot{x} \\ \tilde{z} \\ \dot{\tilde{z}} \end{bmatrix}, \quad A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ R_{\mathrm{d},13}/m \\ 0 \\ R_{\mathrm{d},33}/m \end{bmatrix},$$

where $\zeta$ is the state vector, $R_{\mathrm{d},ij}$ are the corresponding elements of the reference rotation matrix $R_\mathrm{d}$ (converted from the reference quaternion $q_\mathrm{d}$), and $A, B$ are the state space matrices. As the equations are decoupled, the effect of gravity can be added to the $z$ position after a simulation, thus in the equation $\tilde{z}$ denotes the modified state. Notice that (5.40) is a linear state space representation with the thrust force $F = u$ as the only control input. By discretizing the system, a quadratic optimization problem can be formulated over a finite horizon, similarly to model predictive control [9]. We calculate the discrete time state space model using complete, zero-order hold discretization, resulting in the form

$$\zeta_{k+1} = A_k\zeta_k + B_k u_k,$$

$$A_k = \begin{bmatrix} 1 & T_\mathrm{s} & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T_\mathrm{s} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad B_k = \begin{bmatrix} R_{\mathrm{d},13}T_\mathrm{s}^2/(2m) \\ R_{\mathrm{d},13}T_\mathrm{s}/m \\ R_{\mathrm{d},33}T_\mathrm{s}^2/(2m) \\ R_{\mathrm{d},33}T_\mathrm{s}/m \end{bmatrix}, \tag{5.41}$$

where $A_k, B_k$ are the discrete state space matrices. The input of the model is the collective thrust of the propellers, $u_k = F_k$. For a fixed duration of the maneuver with $N$ discrete time steps, the following quadratic optimization problem is formulated:

$$\begin{aligned} \underset{u}{\text{minimize}} \quad & \sum_{k=1}^{N} \left[ (\zeta_k - \zeta_{\mathrm{d},k})^\top Q_k \, (\zeta_k - \zeta_{\mathrm{d},k}) + u_k^\top W_k u_k \right] \\ \text{subject to} \quad & \zeta_{k+1} = A_k \zeta_k + B_k u_k, \\ & \{\zeta_k\}_{k=1}^{N} \in \mathcal{X}, \\ & \{u_k\}_{k=0}^{N} \in \mathcal{U}, \end{aligned} \tag{5.42}$$

where $Q_k \in \mathbb{R}^{4\times4}$ and $W_k \in \mathbb{R}$ are weight matrices, and $\mathcal{X}, \mathcal{U}$ are the sets of constraints for the states and the control input, respectively. The only objective of the trajectory design is to minimize the final position error of the quadcopter and keep the position within a specified range, therefore the weight matrices are $W_k = 0, Q_k = 0$ for $k = 1 \dots N$, except for the weight of the final state that is

$$Q_N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{5.43}$$

As all the other weights are zero, it is only required to define a final state position reference that is defined as

$$\zeta_{\mathrm{d},N} = \begin{bmatrix} x_{\mathrm{d},N} \\ \dot{x}_{\mathrm{d},N} \\ \tilde{z}_{\mathrm{d},N} \\ \dot{\tilde{z}}_{\mathrm{d},N} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} g (T_s N)^2 \\ 0 \end{bmatrix}, \tag{5.44}$$

where $\zeta_{\mathrm{d},N}$ contains the effect of gravity during the time span of the maneuver.

We specify linear constraints for the states: $x \in [x_-, x_+]$, $z \in [z_-, z_+]$ to model the available space for the maneuver avoiding collisions with other objects or walls. We also define linear constraints for the control input, namely

$$\frac{\|\tau_k\|}{l} \leq u_k = F_k \leq F_{\max} - \frac{\|\tau_k\|}{l}, \tag{5.45}$$

where $\tau_k$ is the vector of the three torques around the three body axes, out of which $\tau_x = \tau_z = 0$ normally during the flip, $l$ is the distance of the quadcopter center of mass and the propellers projected to the $x - z$ plane, and $F_{\max}$ is the maximal collective thrust of the rotors. The torque control input $\tau_k$ is calculated from the reference attitude $R_\mathrm{d}$ based on (5.8b) assuming that the orientation and angular velocity errors are zero.

The numerical solution of the optimization problem (5.42) can be obtained easily by using an off-the-shelf QP solver, e.g. by `quadprog` in Matlab. Finally, we fit cubic splines on the discrete points of the optimized reference trajectory, which the quadcopter follows with geometric tracking control.

## 5.5 Conclusions

In this chapter, first we introduced geometric control, which is an efficient nonlinear control method for the trajectory tracking of quadrotors. The main advantage of this method compared to classical controllers (PID, LQR) is that the error terms are formulated directly on the configuration space of the quadrotor, therefore almost global convergence to the reference trajectory can be proven mathematically by assuming perfect knowledge of the dynamic model.

Second, we proposed a nonlinear adaptive robust trajectory tracking control method based on the introduced geometric approach. For this, we augmented the dynamic model of the quadrotor by Gaussian Processes to account for uncertainties and unmodelled dynamics. The model augmentation and the control law are characterized by (5.12) and (5.13), respectively. We provided Theorem 1, which described the rigorous mathematical foundations for the boundedness of trajectory tracking errors in the presence of uncertainties represented by GPs that are conditioned on measurement data. Although in this work we only demonstrate the example of the backflip maneuver, the proposed controller can be applied to follow a large set of aggressive quadrotor trajectories.

Finally, we proposed a trajectory planning method by constrained model-based optimization designed directly for the backflip maneuver of quadrotors. The proposed optimization method characterized by (5.42) contains only quadratic programming, therefore it is efficient to solve numerically. The trajectory planning and tracking control design together provide a second approach to perform the backflip maneuver with miniature quadrotors, after the first presented in Chapter 4. The advantage of the method presented in this chapter compared to the optimization-based feedforward control is that closed-loop control is generally more robust to modelling uncertainties, moreover, the proposed robust adaptive controller is able to incorporate measurement data to provide reliable operation even in case of unexpected external disturbances.

# Chapter 6

# Simulation Study

## 6.1 Simulation environment

The simulations are based on the dynamic model of a Bitcraze Crazyflie 2.1 miniature quadcopter which we use for demonstrating the experimental results in Chapter 7, as well. The nonlinear equations of motion are defined in Chapter 3, and the physical parameters of the drone are shown in Table 6.1, obtained from [20].

After some literature survey about drone simulators such as [44] and [47], we decided to choose an OpenAI Gym environment based on PyBullet [41]. This framework is written in Python language, contains the physical model and parameters of the Bitcraze Crazyflie 2.1, has built-in multi-agent control, and tuned PID controller for the drone. There are also some simple examples for trajectory tracking, and reinforcement learning with a swarm of drones.

All of the simulation code used in this work is available at GitHub[1], the Python code is in the `simulations/crazyflie-demo-simulation` repository, and the Matlab code is in the `trajectory-design` repository. A video illustrating the simulation results is available at `https://youtu.be/AhqfXZ-CPqM`. In this section and the oncoming sections, we plot the measurement results with the $z$ axis pointing upwards (in contrast to the NED convention discussed in Chapter 3), because the backflip maneuver is more illustrative this way.

[1] `https://github.com/antalpeter1999/TDK2021`

**Table 6.1:** Physical parameters of the Crazyflie 2.1 quadcopter.

| | | |
|---|---|---|
| Mass | $m$ | 0.028 g |
| Propeller-to-propeller length | $l$ | 92 mm |
| Diagonal inertia elements | $J_{\text{xx}}$ | $1.4 \cdot 10^{-5}$ kgm$^2$ |
| | $J_{\text{yy}}$ | $1.4 \cdot 10^{-5}$ kgm$^2$ |
| | $J_{\text{zz}}$ | $2.17 \cdot 10^{-5}$ kgm$^2$ |
| Thrust coefficient | $k$ | $2.88 \cdot 10^{-8}$ Ns$^2$ |
| Drag coefficient | $b$ | $7.24 \cdot 10^{-10}$ Nms$^2$ |

## 6.2 Optimized feedforward control

Based on the control approach presented in Chapter 4, we implemented the open-loop flip controller which evaluates the five sections of the parametrized motion primitive. For the optimization of the parameters defined in (4.4), we utilize the Bayesian Optimization Python module [40].

The simulation starts with hovering for about 0.1 s, followed by executing the open-loop maneuver, and then switching back to PID control to stabilize the drone and bring back to the initial setpoint. The objective function of the optimizer uses the same simulation, and returns the norm of the final state error, as it is characterized in (4.6). The optimal parameters were calculated with Bayesian optimization as the solution of (4.6), using 250 random initial function evaluations and 1000 iterations. The numerical values of these are

$$p^* = \begin{bmatrix} U_1^* & t_1^* & t_3^* & U_5^* & t_5^* \end{bmatrix}^\top = \begin{bmatrix} 17.5 & 0.12 & 0.16 & 17.95 & 0.075 \end{bmatrix}^\top, \tag{6.1}$$

where the unit of the collective accelerations $U_i^*$ is m/s$^2$, and the time is in seconds. It is simple to convert the collective acceleration to collective thrust,

$$F = mU, \tag{6.2}$$

where $m$ is the mass of the quadcopter. Simulation results are displayed in Figure 6.1, using the optimal parameter vector and a set of near-optimal parameters, as well. On the left plot, the position of the quadcopter during the flip is shown, with snapshots from the simulation. The end of the optimal maneuver is around the coordinate $(x, z) = (-0.4, 0)$ m with near-zero pitch angle, thus the final state error is only significant in the $x$ position. From that point, a PID controller [41] stabilizes the drone and controls to the origin. On the right, the trajectory of the pitch angle in Euler representation, the angular velocity, and the collective thrust as a control input are shown, where the five sections defined in

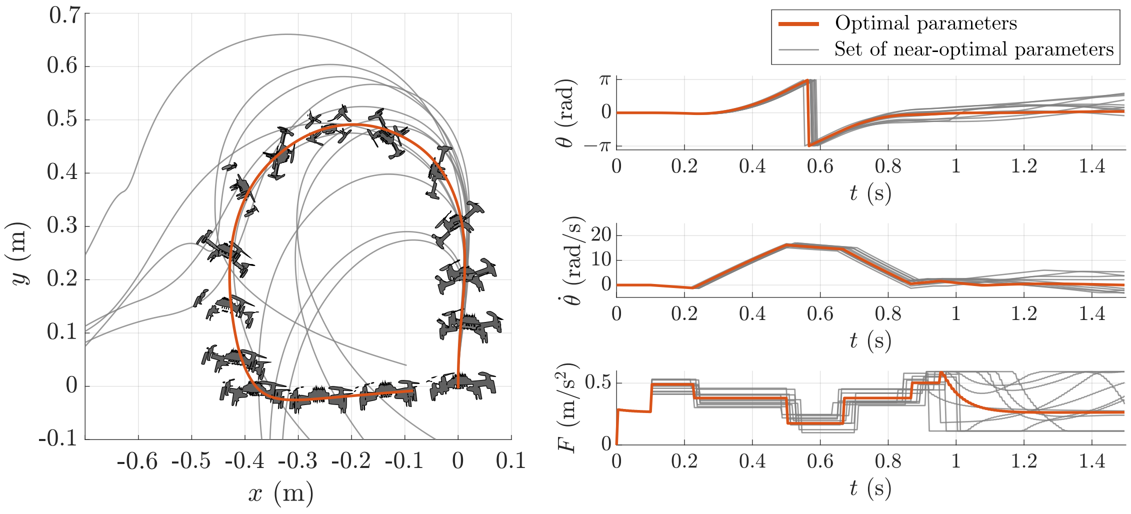


**Figure 6.1:** Open-loop backflip simulation results: the position is displayed on the left, and the pitch angle $\theta$, pitch angular velocity $\dot{\theta}$, and collective thrust $F$ on the right. Orange lines represent the simulation with optimal parameters in (6.1), and grey lines represent the result of small changes in the parameter set.

Chapter 4 can be recognised clearly. The figure illustrates that even small deviations from the optimal parameter set (<10%) result in significantly decreasing performance.

## 6.3 Trajectory planning and geometric control

The second control approach to perform a flip maneuver is trajectory planning and reference tracking with geometric control. Based on the results of the flip with open-loop control, the parameters of the reference pitch trajectory are chosen to $\alpha = 20$ 1/s, $\beta = 0.45$ s, as illustrated in Figure 5.3. In the quadratic optimization problem (5.42) we use the following parameters:

$$\mathcal{X}: \begin{bmatrix} x_- \\ x_+ \\ z_- \\ z_+ \end{bmatrix} = \begin{bmatrix} -0.6 \\ 0 \\ -0.05 \\ 0.45 \end{bmatrix} \text{ m}, \tag{6.3a}$$

$$\mathcal{U}: \; F \in [0, 0.64] \text{ N}, \tag{6.3b}$$

$$T_s = 1/480 \text{ s}. \tag{6.3c}$$

The maximal collective thrust $F_{\max} = 0.64$ N is from [20], and the position bounds are chosen such that the trajectory is feasible, and the quadcopter does not get too far from the initial point, for example we can express the available flying space here to avoid collision with walls or other objects. The duration of the flip is chosen to $T = 0.9$ s, thus the number of simulation steps is $N = T/T_s = 432$. The quadratic optimization problem is solved by the `quadprog` function of Matlab within milliseconds of computation time. The result is the optimal control input and reference position sequence given in the discrete time instants. Finally, we fit cubic splines on the discrete points of the optimized reference trajectory which can be evaluated on-board the quadcopter.

The quadcopter follows the designed reference trajectory with geometric tracking control, based on the control law (5.8). The control gains were determined based on grid search and considering that the Mellinger controller [36] with a similar control law is part of the official Crazyflie 2.1 firmware[2]. The numerical values are as follows:

$$k_{\mathrm{r}} = 4.5, \tag{6.4a}$$

$$k_{\mathrm{v}} = 0.3, \tag{6.4b}$$

$$k_{\mathrm{R}} = 0.2, \tag{6.4c}$$

$$k_{\omega} = 0.003. \tag{6.4d}$$

The simulation results of the trajectory planning and reference tracking with geometric control are displayed in Figure 6.2. Similarly to the open-loop approach, the maneuver starts and ends at hovering, but in this case the stabilizing controller is also the geometric control with position and yaw setpoint. At the beginning of the maneuver, the controller is switched from yaw to pitch reference to follow the specified reference trajectory. The left plot illustrates the reference and simulated pose of the quadcopter during the backflip maneuver, and the right plot contains the trajectory of the pitch angle, angular velocity and collective thrust control input. The trajectory of both the angular velocity and the thrust input are smooth, opposed to the discontinuous angular acceleration and thrust of the open-loop control. In the discontinuities of the control input the unmodelled transient

[2]`https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/`

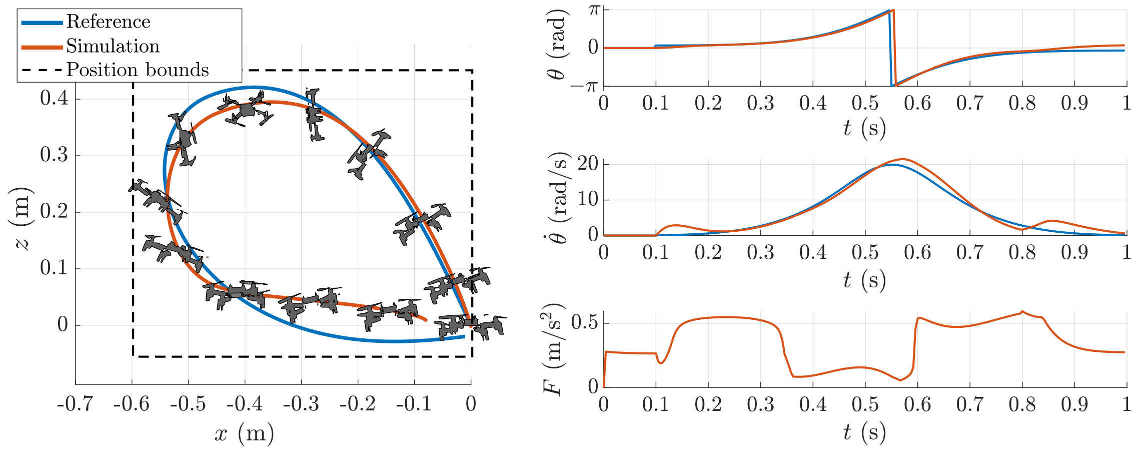


**Figure 6.2:** Backflipping simulation results with geometric control: position and attitude on the left, the pitch angle $\theta$, pitch angular velocity $\dot{\theta}$, and collective thrust $F$ on the right.

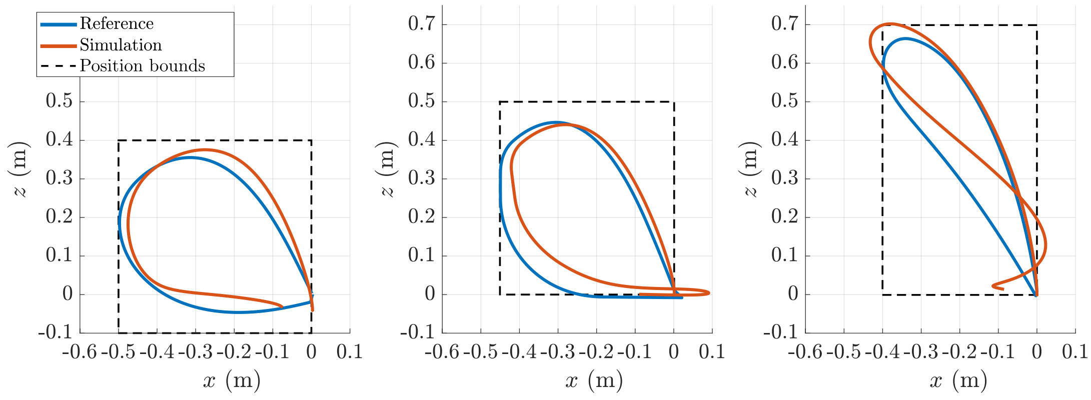


**Figure 6.3:** Backflipping simulation results with geometric control. The position reference constraint is different for each simulation, illustrated by the dashed lines.

behaviour of the actuator dynamics can be significant, therefore the geometric control approach is more robust to such uncertainties compared to the open-loop method.

It can be seen that the orientation reference tracking is more precise than the position. Fast and accurate attitude tracking is required to stabilize the quadcopter, however, the reference trajectory is designed with the assumption that the attitude equals to the reference at all times, therefore there are more significant errors in the position.

Next, we evaluate the performance of the proposed robust adaptive geometric controller for backflipping. We assume that the most significant modelling errors compared to the real drone arise in the rotational dynamics, due to the inaccuracy of the inertia matrix and center of gravity, and other aerodynamic effects. In simulation, we apply an external disturbance characterized by

$$\Delta_{\mathrm{R}} = \begin{bmatrix} 0.002 & 0.002 & 0 \end{bmatrix}^\top \sin\left(\frac{\phi}{2} + \frac{\theta}{2}\right) \text{ Nm}, \tag{6.5}$$

and shift the center of gravity by 10 mm in $x$ direction. We use the controller gains given by (6.4), and choose the following constant values based on the stability conditions detailed in [31]: $\tau = 3, c_1 = 1, c_2 = 0.1, \epsilon_r = 0.0004, \epsilon_R = 0.0004$.

In the example of the backflip maneuver, we use only two scalar adaptive terms: the roll and pitch term of $\eta_{\mathrm{R}}$, namely $\eta_{\mathrm{R,x}}$ and $\eta_{\mathrm{R,y}}$, because most of the uncertainties arise in the roll and pitch motions. The adaptive terms are represented by two independent Gaussian Processes, as follows:

$$\hat{\Delta}_{\mathrm{R,x}} = \mathcal{GP}_{\mathrm{R,x}}(\chi) \sim \mathcal{N}(\eta_{\mathrm{R,x}}(\chi), \Sigma_{\mathrm{R,x}}(\chi)), \tag{6.6a}$$

$$\hat{\Delta}_{\mathrm{R,y}} = \mathcal{GP}_{\mathrm{R,y}}(\chi) \sim \mathcal{N}(\eta_{\mathrm{R,y}}(\chi), \Sigma_{\mathrm{R,y}}(\chi)), \tag{6.6b}$$

with the following four-dimensional input:

$$\chi = \begin{bmatrix} q_1 \\ q_2 \\ \omega_{\mathrm{x}} \\ \omega_{\mathrm{y}} \end{bmatrix}, \tag{6.7}$$

where $q_1, q_2, \omega_{\mathrm{x}}, \omega_{\mathrm{y}}$ are the $x$ and $y$ elements of the attitude quaternions and the angular velocity vector, respectively.

The adaptive GP terms introduced in Chapter 5 depend on the full state vector, however, in case of the backflip scenario, only these four inputs are relevant, and by reducing the input dimension, the model complexity is decreased radically without losing its expressiveness. For the Gaussian Processes, we use zero mean and squared exponential covariance function, given by (2.4). The scaling ($\sigma_{\mathrm{f}}$) and smoothness ($\Lambda$) hyperparameters are trained using maximum likelihood estimation on 125 training points. Due to the relatively small input dimension and number of training points, the training and evaluation of the GPs are fast and efficient.

The backflipping simulations by robust adaptive geometric control are illustrated in Figure 6.4, and the trajectory followed by the quadrotor is displayed in Figure 6.2. We compare the results using nominal geometric control (without adaptive and robust terms), geometric control with GP mean (without robust terms), and robust adaptive geometric control given by (5.8), (5.16). The most important variable is the attitude tracking error $\Psi$, described by (5.3). Our results show that the attitude error of the nominal geometric controller during backflipping is reduced by 40% using the mean of the GPs, and 80% using the proposed robust adaptive geometric control. The difference in the position error norm ($\|e_{\mathrm{r}}\|$) is less significant due to the effect of control input saturation, however, the adaptive and robust controllers achieve a better result here, as well. The figure also shows that the adaptive terms $\eta_{\mathrm{R,x}}, \eta_{\mathrm{R,y}}$ have significant influence on the motion control, as their peak magnitude is almost half of the control inputs $\tau_{\mathrm{R,x}}, \tau_{\mathrm{R,y}}$ of the quadrotor. Moreover, in case of the GP robust adaptive control (plotted in yellow color), the robust terms $\mu_{\mathrm{R,x}}, \mu_{\mathrm{R,y}}$ have almost the same peak magnitude as the control inputs. When interpreting the results of Figure 6.4, it is important to note that the control inputs $\tau_{\mathrm{R,x}}, \tau_{\mathrm{R,y}}$ include saturation due to the actuator limits of the quadrotor.

## 6.4 Conclusions

The simulation results clearly show that backflipping is successfully implemented using both the optimization-based feedforward control proposed in Chapter 4, and the trajectory planning with geometric feedback control proposed in Chapter 5. The Bayesian optimization of the former method is computationally expensive compared to the model-based quadratic optimization of the latter approach. Moreover, the results in Figure 6.1

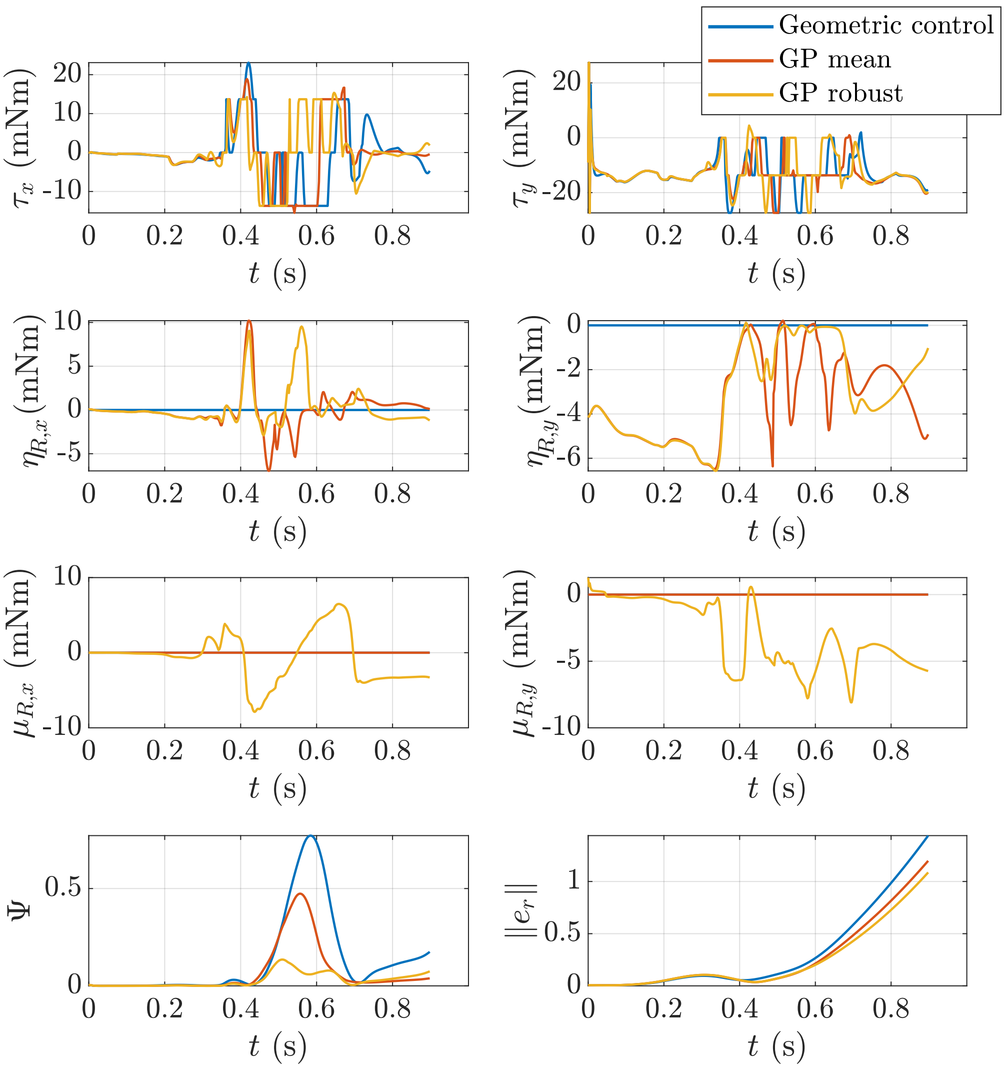


**Figure 6.4:** Comparison of nominal geometric control, adaptive geometric control with GP mean, and robust adaptive geometric control with GP mean and variance in the presence of external disturbance characterized by (6.5). The roll and pitch component of the control input $(\tau_\mathrm{x}, \tau_\mathrm{y})$, the adaptive and robust terms $(\eta_{\mathrm{R,x}}, \eta_{\mathrm{R,y}}, \mu_{\mathrm{R,x}}, \mu_{\mathrm{R,y}})$, the attitude error $(\Psi)$ and the position error norm $(\|e_\mathrm{r}\|)$ are displayed.

show that the feedforward method is very sensitive to parameter changes. According to Figure 6.3, the trajectory planning with geometric control is less sensitive to the change of parameters (in this case the position constraints), indicating that the method is able to provide good performance in a range of different environments.

The performance of the proposed robust adaptive controller is evaluated by applying an external disturbance to the rotational dynamics while executing the backflip maneuver, i.e., following the trajectory displayed in Figure 6.2. The results show that even though the performance of the controller is limited by the saturation of the actuators, the robust adaptive controller outperforms both the nominal geometric controller and the adaptive controller. In the presence of such high external disturbance, the feedforward control quickly becomes unstable and it is unable to execute the backflip maneuver.

# Chapter 7

# Implementation and Experimental Study

## 7.1 Experimental setup

In this chapter, real-world implementation of the two approaches to perform the backflip maneuver with the Crazyflie 2.1 drone, the experimental setup, and measurement results are presented. The experimental setup consists of the quadcopter with on-board sensors and microcontroller units (MCUs), the OptiTrack motion capture system, OptiTrack server, and ground control PC. The block diagram of the setup is displayed in Figure 7.1 with the direction and content of the information flow between the components.

### 7.1.1 The Bitcraze Crazyflie 2.1 drone

This section presents an overview of the hardware and software of the Crazyflie 2.1 nano quadrotor, developed by the Swedish company Bitcraze AB. The vehicle is designed to be a development platform for research and education, therefore it is both fully open source and open hardware. Firstly the hardware specifications are summarized, and afterwards the structure of the firmware and control implementation possibilities are discussed.

The quadcopter is an out-of-the-box device, the user only needs to assemble the parts. The central unit contains IMU sensors with accelerometer, gyroscope, magnetometer and barometer, and two microcontrollers: a STM32F405 for running the main application (e.g. state estimator, controller), and a nRF51822 for radio communication and power management. The four 4.2 V coreless BDC motors are connected to the central unit with plastic motor mounts. The propellers are fixed to the motor shafts using interference fit, and a 250 mAh LiPo battery is connected to the central unit to provide electric power. The drone weighs 28 grams, and the propeller-to-propeller distance is 92 millimeters.

Using the stock battery the flight time is 7 minutes without any external payloads. The maximum recommended payload is 15 grams including the extension decks which can be mounted to the drone. We only use one extension deck, on which the OptiTrack markers are mounted (see Figure 7.2), but there are several possibilities to extend the functionality and visual effects, for example the flow deck for optical flow measurement or the LED-ring deck[1].

[1]`https://www.bitcraze.io/documentation/system/platform/cf2-expansiondecks/`

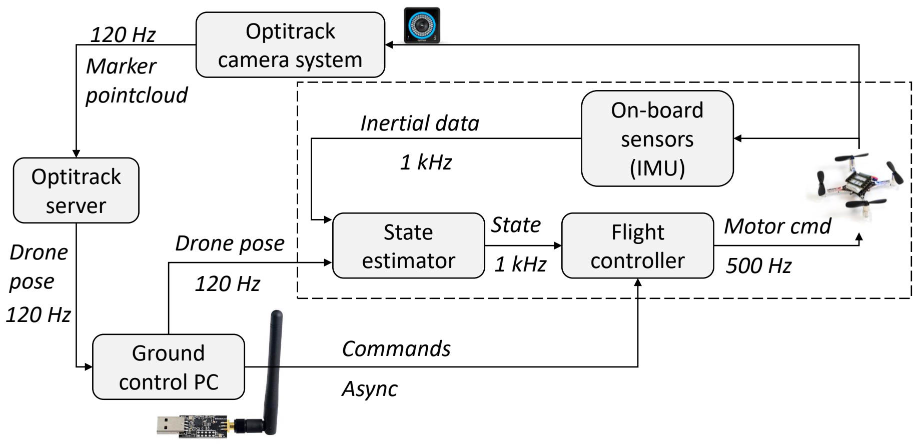


**Figure 7.1:** Block diagram of the experimental setup: indoor quadcopter navigation with internal and external measurement system.

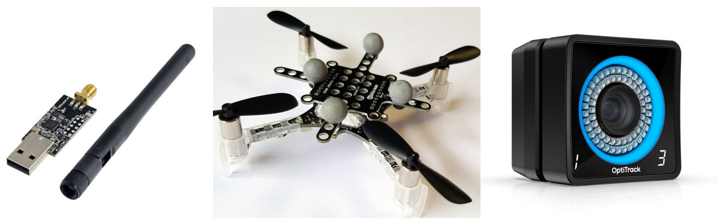


**Figure 7.2:** The Crazyradio PA, Crazyflie 2.1 with motion capture markers, and an OptiTrack Prime 13 camera.

The ground control PC communicates with the drone via the Crazyradio PA 2.4 GHz USB dongle. The radio communication makes it possible to update the firmware without a cable, and communicate with the quadrotor mid-flight, i.e. send external position data, update parameters, or log measurement data. The main parts of the hardware are displayed in Figure 7.2.

The open-source software continuously developed by Bitcraze[2] includes the firmware, a Python client application and Python library with examples. In addition, we use the Crazyswarm platform [42] to easily handle the high-level control of multiple drones simultaneously, the OptiTrack measurement data, logging, and communication.

Out of all software components, we mostly use the firmware written in C language to implement our on-board controller, and the Python API for high-level commands and computation. The nRF51822 microcontroller manages the communication and power distribution, in which areas the default implementation is sufficient for us. However, the firmware of the STM32F405 contains the stabilizer unit responsible for handling high-level commands, state estimation and control input calculation. Two state estimators are implemented in the stock firmware of the Crazyflie: a complementary filter and extended Kalman filter. We use the latter, as it is more accurate for the fusion of internal and external sensor data.

There are three built-in controllers in the default firmware, a cascaded PID controller, the Mellinger controller [36], and the Incremental Nonlinear Dynamic Inversion (INDI) controller [45]. In this project, we use the PID controller for setpoint stabilization before and

[2]https://www.bitcraze.io/documentation/repository/

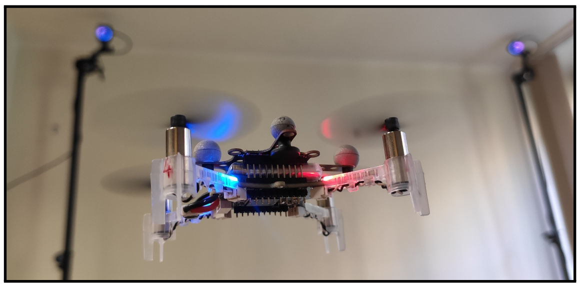


**Figure 7.3:** Experimental setup for the presented measurement results: 2 OptiTrack Prime 13 infrared cameras and a Crazyflie 2.1 quadcopter with reflective markers.

after the open-loop flip maneuver. Our own implementation of the geometric controller is responsible for backflipping with pitch reference, setpoint stabilization, and more complex, aggressive trajectory tracking.

### 7.1.2 OptiTrack motion capture system

At SZTAKI AIMotion Lab, we use OptiTrack for the real-time pose measurement of the drones which is a high precision motion capture system with submillimeter resolution. The system includes six Prime 13 infrared cameras sending the position information of specific markers at 120 Hz. Using the Motive software of OptiTrack, we are able to define rigid bodies with unique identifiers (both drones and obstacles), and broadcast their position and orientation information directly through local network.

Although OptiTrack is a very precise and efficient measurement device, sometimes it is not easy to arrange the markers on the drone such that at least four cameras see them at all times, especially during the flip maneuver. In these situations, the state estimator of the drone automatically switches to use only the information of the inertial measurement unit, which is sufficient for hovering and attitude stabilization.

## 7.2 Backflip maneuver implementation

In the Crazyswarm framework (running on the workstation PC) there are built-in examples for the high-level control of the quadcopter: for example it is enough to call the `takeOff()`, `goTo()`, and `land()` methods from a script to take off, hover for a specified duration, go to a given point in space, and land – these commands are sent to the on-board microcontroller through a ROS C++ layer and the Crazyradio PA. In the `.launch` file of ROS, the user can specify the initial position of the drone, the controller type, state estimator type, logging variables, and other properties. The firmware parameters can be modified mid-flight, for example to adjust controller gains, or switch between controller types.

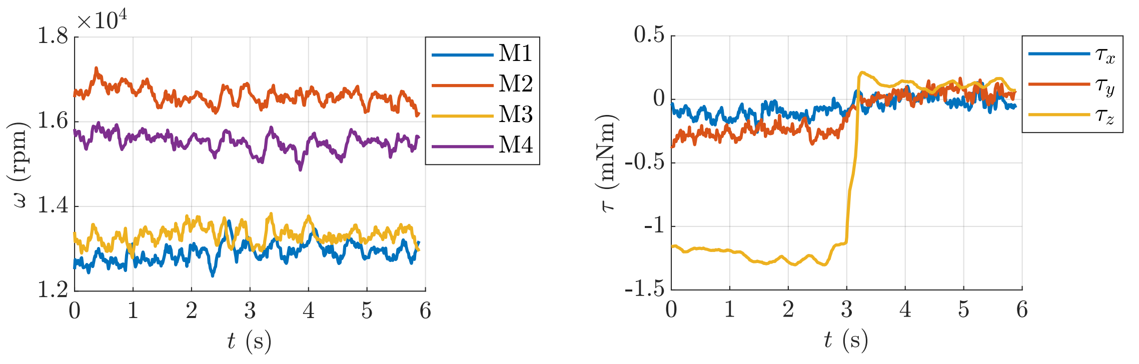
Our own implementation is a modified version of Crazyswarm, available at the GitHub repository `TDK2021/real_quadcopter/crazyswarm`[3]. Our main contributions are in the controller of the firmware (open-loop and geometric control), and in the Python API for high-level commands, and simultaneous control of multiple drones.

[3]`https://github.com/antalpeter1999/TDK2021`

### 7.2.1 Motor speed correction

In the early implementation phase of the open-loop control, we have noticed that there was a significant difference between the characteristics of each motor on a Crazyflie quadcopter. As it is shown in Figure 7.4a, there is more than 3000 rpm and 18% difference between the angular speed of motor M1 and M2 when the quadcopter is hovering, although the angular speeds should be equal based on the dynamic model. This effect results in the offset of the torque control inputs in all directions, as it is illustrated in Figure 7.4b when $t < 3$ s.

Hence, we have implemented a correction algorithm to overcome the negative effect on the performance of the backflip. The `motorCorrect` method is called when the drone is hovering. For 3 seconds it averages the angular speed of each motor and afterwards the control inputs are corrected in order to avoid the offset, as it is shown in Figure 7.4b when $t > 3$ s. The correction method has successfully improved the performance of the open-loop backflipping without complex identification of the actuator dynamics.



**(a)** Angular speed of the four rotors: in hover mode the mean values should be equal.

**(b)** Torque control input: the correction is activated at $t = 3$ s, after which the mean of each control input is around zero.

**Figure 7.4:** Motor speed correction in hovering.

### 7.2.2 Control algorithm implementations

The high-level script to perform a backflip maneuver is quite simple, because all control algorithms are implemented on-board. However, we need to build and flash the firmware of the quadcopter MCU after every change in the source code, therefore parameter setting and execution of high-level tasks is easier using Python scripts. The steps of the high-level script to perform the backflip are presented in Algorithm 2.

**Algorithm 2** High-level script running on the ground control PC

```
1: Send parameters to the Crazyflie: controller type (open-loop or geometric), trajectory
   information, controller gains
2: Send commands to take off and hover at the initial point of the flip
3: Call motorCorrect method to correct the motor angular speeds
4: Call performOpenLoopFlip or performGeomFlip to start backflipping with open-loop
   or geometric control, and wait until the maneuver is over
5: Send command to land
```

The evaluation of the open-loop control law, i.e. the calculation of control inputs from the current time has small computational cost, therefore it is practical to be implemented on-board. The implementation is detailed in Algorithm 3, the maneuver is triggered from the ground station with a high-level command, it is executed automatically, and the quadcopter ends in hovering mode afterwards.

**Algorithm 3** `performOpenLoopFlip` on-board implementation

```
1: Lift with PID control to gain momentum
2: while the flip is not over do
3:     Decide which section we are in (out of the 5 in Figure 4.2)
4:     Apply the control input of the current section based on (4.5)
5: end while
6: Stabilize the quadcopter with PID control
7: Get back to initial position
```

The second control approach, i.e. reference trajectory tracking with geometric control is also implemented on-board. The coefficients of the designed trajectory represented by 7th degree polynomials, and the corresponding time span are saved to a csv file after the simulations. In the Python API, it is loaded and sent to the Crazyflie with the tuned controller gains, before taking off. The on-board implementation of the control approach is detailed in Algorithm 4. Similarly to the open-loop approach, the command to start the maneuver comes from the ground station, and then the backflip is performed automatically.

**Algorithm 4** `performGeomFlip` on-board implementation

```
1: Lift with geometric control to gain momentum
2: while the flip is not over do
3:     Get the current state from the state estimator module
4:     Get the setpoint by evaluating the reference trajectory
5:     Calculate the control inputs from the control law (5.8)
6:     Apply the control inputs to the motors
7: end while
8: Stabilize the quadcopter with geometric control
9: Get back to initial position
```

When the user wants to start the backflip maneuver, similarly to the open-loop method, only needs to set the value of `isFlipControl` to true. The on-board algorithm then starts executing the previously saved trajectory, switches to pitch reference from yaw reference, and switches back after the maneuver is over.

A big advantage of the on-board control is that the maneuver can be performed by multiple quadcopters simultaneously without any problems coming from communication delays and lost packages between the ground control PC and the drone microcontroller.

## 7.3 Experimental results

Experiments to perform the backflip maneuver started with the open-loop method. Firstly, we used the optimal parameter set from (6.1), but due to the differences of the simulation model and the real quadcopter dynamics, the drone almost performed a double flip with these parameters, and the stabilization was not successful at the end of the maneuver. After realizing the differences between the measurements and the simulations, we manually tuned the parameters so that the flip is executed with minimal final state error. The refined experimental parameter set is

$$p^{\text{exp}} = \begin{bmatrix} U_1^{\text{exp}} & t_1^{\text{exp}} & t_3^{\text{exp}} & U_5^{\text{exp}} & t_5^{\text{exp}} \end{bmatrix}^\top = \begin{bmatrix} 14.29 & 0.2 & 0 & 14.29 & 0.075 \end{bmatrix}^\top .$$

The measurement results are displayed in Figure 7.5, showing the trajectory of the position and orientation during the maneuver. It is important to note that an additional lift phase is added to the implementation to gain enough vertical velocity and height, because the quadcopter falls a significant distance in the recovery phase. During the additional lift phase the PID controller[4] is used to achieve exact vertical lifting and horizontal orientation.

Figure 7.5 shows that the flip is executed with almost zero final error in the pitch, and also quite small position error. Compared to the simulation displayed in Figure 6.1, the maximal displacement from the origin in $x$ direction is around a half, and in $z$ direction around double. The difference is due to the manual refinement of the motion sequence parameters, and the uncertainties of the simulation model. It is important to note that the performance of the open-loop controller is very sensitive to uncertainties in the dynamics and initial conditions. For example, if the flip maneuver begins when the orientation of the quadcopter is not exactly horizontal, the stability can be lost at the recovery phase. Hence the open-loop flip is only successful in about 6-7 trials out of 10.

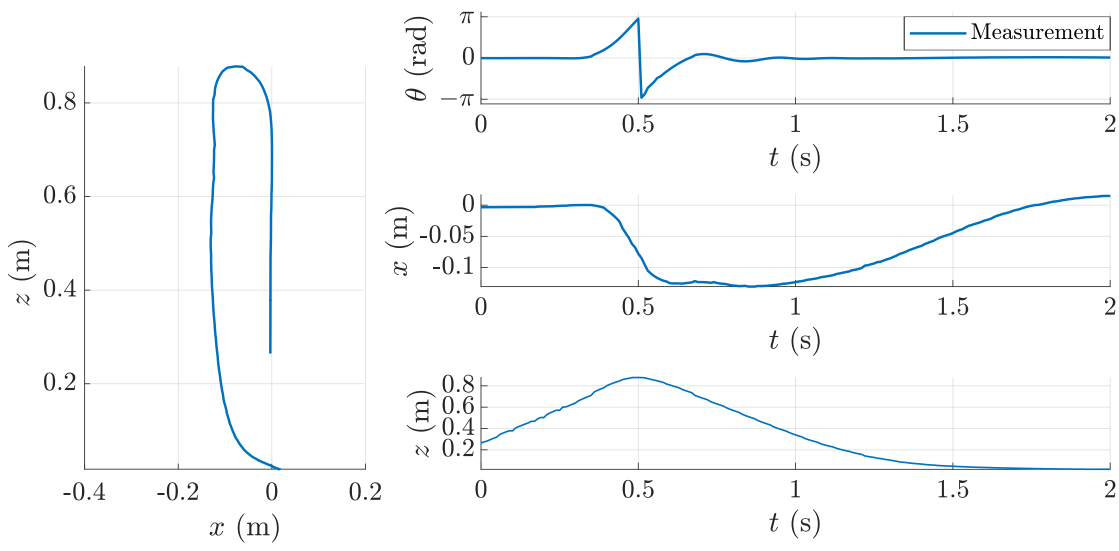


**Figure 7.5:** Backflipping measurement results with open-loop control. The position and pitch angle of the quadcopter are displayed.

The experimental results of the backflipping with nominal geometric control are displayed in Figure 7.6. The most important part of reference tracking is the pitch angle $\theta$, because

[4]https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/sensor-to-control/controllers/#cascaded-pid-controller

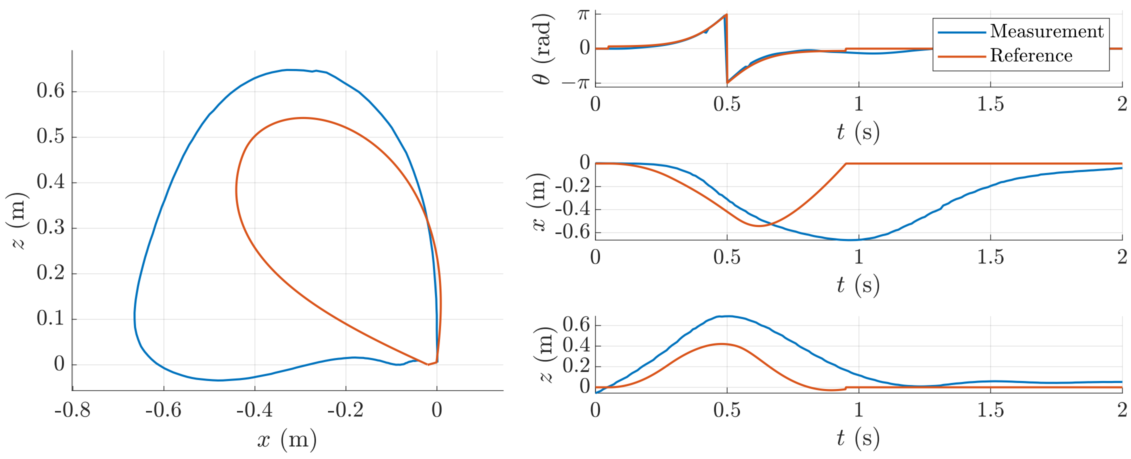


**Figure 7.6:** Backflipping measurement results with geometric control. The trajectories show that the maneuver is performed successfully, and the drone gets back to the initial position at the end.

a fast, stable and accurate attitude tracking is required to perform the flip maneuver, and recover successfully. As it is shown on the upright plot of the measurement results, the pitch is very close to the reference.

However, the error of the position tracking is significantly higher during the maneuver, both in $x$ and $z$ directions. There are many possible reasons of the high positioning error, the following are the most likely. Firstly, we assume at the trajectory design that the orientation is equal to the reference at all time steps, which is not far from reality, but there is always a non-zero attitude tracking error. It was another assumption of trajectory design that the derivative of the control input can be arbitrarily large. However, when the quadcopter is upside down, a near-minimal collective thrust is required, and shortly after a near-maximal thrust to recover and follow the reference, as it is illustrated in Figure 7.6. Small DC motors are usually modelled as first order systems with a small time constant, but here the transient can be significant, the motor needs time to build up the thrust of the propellers. The third possible explanation is also connected to model uncertainty. The modelling of aerodynamic effects such as blade flapping, induced drag or downwash [35] are very complex, therefore we exclude these terms from the dynamic model, however, they can have a negative effect on the performance of the nominal feedback controller when performing the backflip maneuver.

Next, we demonstrate the experiments using the proposed adaptive geometric controller. First, we collect data points by backflipping with the nominal geometric controller, and fit a Gaussian Process on the measurement data. The motion control algorithm runs on-board the quadcopter at 500 Hz and the computational capacity of the microcontroller is limited, therefore the real-time evaluation of the GP would not be feasible. To address this problem, we generate a lookup-table by evaluating the trained GP on a grid of the input variables on the desktop PC, and upload it to the on-board computer of the quadcopter. The evaluation of the lookup-table requires only memory operations and linear interpolation, therefore it is computationally cheap. Another possible solution would be the use of sparse GP methods, however, lookup-tables are easier to implement and provide good results.

The performance of the proposed adaptive geometric controller is evaluated with external disturbance added to the quadcopter dynamics. We have found that a bolt attached to the drone has significant influence on the attitude dynamics, however, the vehicle is still able

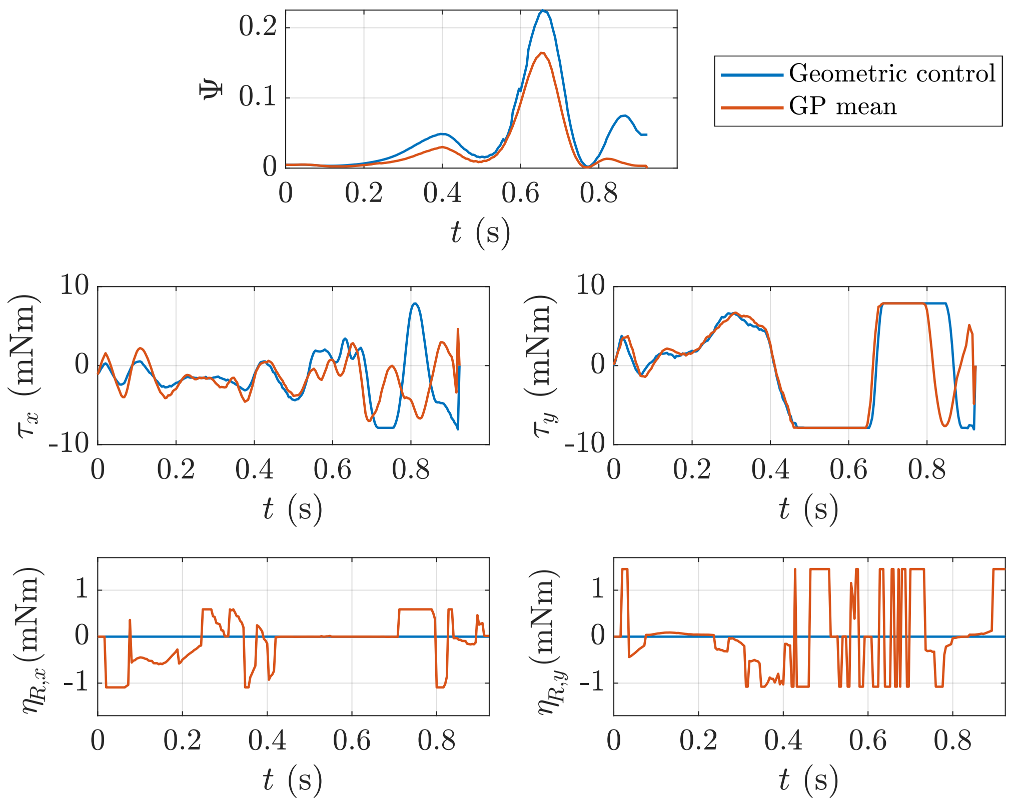


**Figure 7.7:** Backflipping measurement results with nominal and adaptive geometric control, with a bolt attached to the quadcopter. The attitude tracking error ($\Psi$) of the proposed adaptive controller is significantly smaller compared to the nominal controller. The roll and pitch components of the control input ($\tau_\mathrm{x}, \tau_\mathrm{y}$), and the adaptive terms ($\eta_{\mathrm{R,x}}, \eta_{\mathrm{R,y}}$) are also displayed.

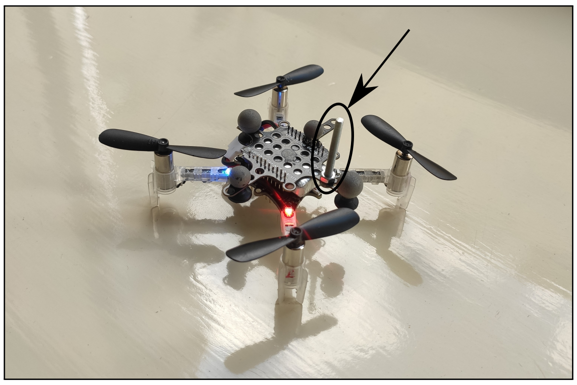


**Figure 7.8:** Crazyflie 2.1 quadcopter with reflective markers and a bolt (marked with an arrow) used as additional disturbance.

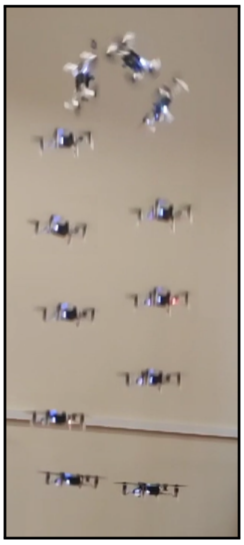
to perform the backflip maneuver. The modified configuration is illustrated in Figure 7.8, and the measurement results in Figure 7.7. Similarly to the simulation results, the attitude tracking error is significantly reduced by using the proposed control algorithm. It is also important that at the end of the backflip, the attitude error of the adaptive controller is almost zero, making it easier to recover and get back to hovering. The control input signal $\tau_y$ is saturated during a significant part of the maneuver, making it difficult to compensate for the disturbance, however, the adaptive algorithm provides better reference tracking in these regions, as well. Compared to the simulation results visualized in Figure 6.4, it can be seen that the adaptive terms $\eta_{\mathrm{R,x}}, \eta_{\mathrm{R,y}}$ have less influence on the control inputs, indicating that the disturbance caused by the bolt attached to the quadrotor is smaller than the artificial disturbance characterized by (6.5), applied in simulation.

## 7.4 Conclusions

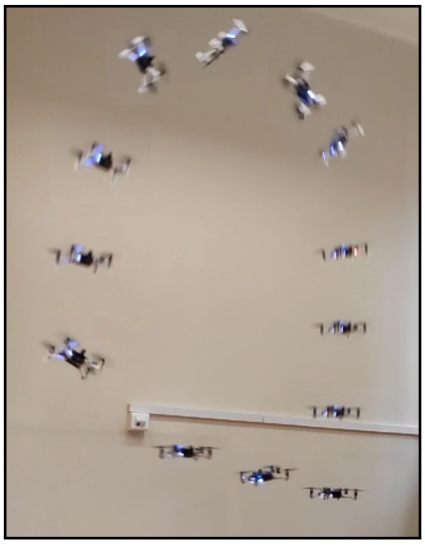
The presented measurement results are satisfactory for both control methods, the quadcopter performed a backflip maneuver successfully. However, the open-loop approach is very sensitive to parameter uncertainties, therefore it needs specific tuning for each Crazyflie. Moreover, it is also sensitive to initial conditions, because the open-loop control input is the same even if there is a nonzero angle or displacement at the starting point. Hence the quality of the maneuver often varies even with the manually refined parameters. Due to the inconsistency of the performance, we could not apply the open-loop control for simultaneous flip with multiple drones.

However, geometric tracking control overcomes the problematics caused by parameter uncertainties using feedback, providing a highly robust and consistent performance for the backflip maneuver. Moreover, the proposed robust adaptive controller enables the quadcopter to perform the backflip even in the presence of external disturbance arising from a bolt attached to the vehicle, that changes two important parameters of the dynamic model: the moment of inertia and the center of mass. Utilizing the robustness of the control approach, the maneuver has been implemented for simultaneous backflipping with three drones. A video of the acrobatic maneuver is available at `https://youtu.be/AhqfXZ-CPqM`.

The scalability of the proposed control algorithms for other types of quadcopters (e.g. medium or large-sized) is out of the scope of this work, however, the possibilities are briefly discussed in the followings. The open-loop method is applied in [34] to a medium-sized drone successfully, therefore we are convinced that our modified version could also be applied to other types. Geometric control is even more general than the open-loop approach, it is applied not only for different types of quadcopters [49], but also for other autonomous systems, such as robotic manipulators [8]. Hence the proposed control algorithms could be used in industrial applications, as well.



**(a)** Optimization-based open-loop control, the trajectory is displayed in Figure 7.5.



**(b)** Geometric reference tracking control, the trajectory is displayed in Figure 7.6.

**Figure 7.9:** Composite images of the measurement results with both proposed control methods implemented on a Bitcraze Crazyflie 2.1 quadcopter. A video of the measurements is available at `https://youtu.be/AhqfXZ-CPqM`.

# Chapter 8

# Towards Control Improvement by Reinforcement Learning

## 8.1 Overview of reinforcement learning for control

In this chapter, we introduce a reinforcement learning based method to improve the performance and adaptability of the demonstrated control methods for backflipping with quadcopters. Reinforcement learning (RL) is a field of machine learning, used for goal-directed learning and decision making. Unlike supervised learning, labeled outputs are not available, the agent learns its behaviour from interacting with its environment, and analysing the reward of state-action pairs. The goal of the learning process is to find the optimal behaviour which maximizes the cumulative reward. In case of control design, the optimal behaviour is an optimal control policy. The typical reinforcement learning scenario is displayed in Figure 8.1.

Early reinforcement learning control methods learned optimal behaviour by maximizing the *value function*, that assigns high values to the states from which the target is reached quickly and the resulting cumulative cost is low. These value-based RL methods, such as Q-learning and state-action-reward-state-action (SARSA), can be applied efficiently in discrete state and action spaces, e.g. playing Atari games [37]. However, their representation capability is limited, and the overall learning algorithm is inefficient in stochastic dynamic environments with continuous states and actions.

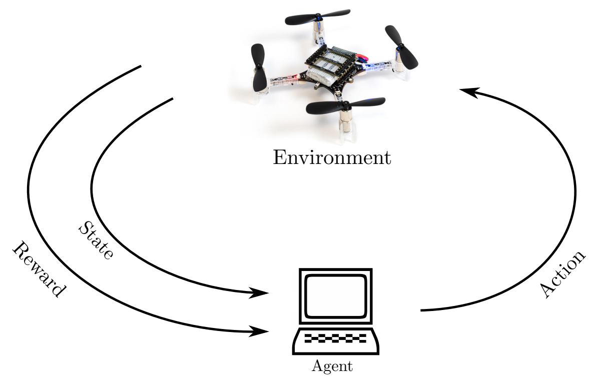


**Figure 8.1:** Typical framing of a reinforcement learning scenario

To improve the performance of the value-based RL algorithms, *actor critic methods* appeared, where the "critic" estimates the value function, and the "actor" represents a control policy. During the learning process, the actor updates the policy based on the value of each state determined by the critic. Most of the widely spread RL control algorithms are based on the actor critic method, e.g. advantage actor-critic (A2C), trust region policy optimization (TRPO), proximal policy optimization (PPO), and deep deterministic policy gradient (DDPG). These algorithms have been successfully applied for complex control problems, e.g. quadrotor control [27].

Reinforcement learning algorithms are also categorized based on their representation of the environment. The previously discussed methods are *model-free*, i.e., the optimal behaviour is learnt without modelling the dynamics of the corresponding system. Research has shown, that although current model-free algorithms have very good asymptotic performance, they are computationally expensive to train, which often limits their application to simulated domains [38, 32]. On the other hand, *model-based* RL methods first construct a predictive model of the environment, and then use that model to optimize the control policy, thus aiming at reducing sample complexity and learning from less data points [11]. Both the dynamic model and policy can be represented by various function approximators, for instance probabilistic, nonparameteric structures (e.g. Gaussian Processes), or deterministic, parametric models (e.g. conventional neural networks).

In the next section, we introduce an RL framework developed specifically for robot learning. Then, we apply the main principles of the introduced methodology to augment the presented control schemes for backflipping.

## 8.2 Probabilistic inference for learning control (PILCO)

### 8.2.1 Problem formulation

In [12], a model-based, data-efficient reinforcement learning method is proposed for learning robot control, namely Probabilistic Inference for Learning Control (PILCO). In case of PILCO, model internalization based RL is used, i.e., a probabilistic dynamic model (in the form of Gaussian Processes) is considered to capture the dynamics of the underlying system. The goal of the learning scenario is to find an optimal control policy via the minimization of a specific long-term cost, which is predicted based on the internal GP model. PILCO is successfully used for various control tasks, e.g. balancing a cart-pole mechanism built from an inkjet printer [2], or controlling a low-cost robotic manipulator [13]. In this section, we only introduce the main principles of PILCO, often simplifying expressions to make their understanding easier. The complete documentation of the method can be found in [12].

First, we assume that the unknown dynamic model of the robotic system can be written in a discrete, deterministic form, as follows:

$$\chi_k = f(\chi_{k-1}, u_{k-1}), \quad \chi_0 \in \mathbb{R}^{n_\chi}, \quad u_0 \in \mathbb{R}^{n_u}, \tag{8.1}$$

where $\chi \in \mathbb{R}^{n_\chi}$ denotes the state, $u \in \mathbb{R}^{n_u}$ is the control input, $k$ is the discrete time index, and $f$ is a Lipschitz continuous, bounded map describing the state transition. The initial state and input are denoted by $\chi_0$ and $u_0$, respectively. In order to simulate the state transition and quantify uncertainty regarding the estimations, $f$ is approximated by a Gaussian Process model, characterized by:

$$f_{\mathrm{GP}}(\hat{x}_k, u_k) = \mathcal{GP}(\hat{x}_k, u_k) \approx \mathcal{N}(\mu(\hat{x}_k, u_k), \Sigma(\hat{x}_k, u_k)), \tag{8.2a}$$

$$\hat{x}_k = f_{\mathrm{GP}}(\hat{x}_{k-1}, u_{k-1}), \tag{8.2b}$$

where $\hat{x}$ is a random variable approximating the state vector, and $\mu, \Sigma$ are the mean and covariance of the GP, detailed in Chapter 2. Given that the input of the GP is a random variable, the output is generally not Gaussian, however, it is usually approximated by a normal distribution to enable computationally feasible uncertainty propagation (commonly used approximation methods are discussed later in Section 8.2.2).

We intend to control the robotic system (8.1) using a deterministic control policy:

$$u_k = \pi(x_k). \tag{8.3}$$

The structure of $\pi$ determines the flexibility and also the complexity of the control policy, therefore it is important to choose both the function class and the number of parameters carefully. The PILCO framework introduces two policy structures: radial basis function (RBF) networks, and linear feedback control, however, more compex structures could be also used, e.g. multi-layer perceptron (MLP). In case of quadrotor control, a linear policy could be used for hovering or low-speed waypoint following, however, performing acrobatic maneuvers requires a complex nonlinear structure, represented e.g. by RBF networks or MLPs.

The objective of the RL task is to find a policy $\pi^*$ that minimizes the expected long-term cost, defined as

$$V^{\pi}(x_0) = \mathbb{E}\left[\sum_{k=0}^{N} c(x_k)\right] = \sum_{k=0}^{N} \mathbb{E}[c(x_k)], \tag{8.4}$$

where $N$ is the length of the corresponding trajectory (also called episode in RL formalism), $x_0$ is the initial state, $\mathbb{E}[\cdot]$ denotes the expected value, and $c(\cdot)$ is a user defined, task specific cost function. The cost function encodes the goal of the control problem, e.g. the state error in path tracking or the distance from the setpoint in setpoint position control. In case of the backflipping scenario, the cost function could incorporate e.g. the aim of each motion primitive defined in Section 4.2, or the orientation and position tracking error compared to the reference trajectories detailed in Section 5.4.

### 8.2.2 Policy learning algorithm

The overview of policy learning is illustrated in Figure 8.2 and by Algorithm 5. First, the structure of the policy and the GP model are defined, and their parameters are initialized randomly or using prior knowledge about the task. After initialization, two main loops are executed. The outer loop is responsible for model learning: executes the control policy on the real robot (if it is available) to collect measurement data, and train the GP model characterized by (8.2).

Then, the inner loop performs iterative policy learning by using the simulation model of the robot dynamics which is formulated based on the GP dynamic model (8.2). The objective of policy learning is to find the policy $\pi^*$ that minimizes the expected long-term cost $V^{\pi}$ in terms of the simulated response with the policy. To evaluate the long-term cost, i.e., the simulated response using the GP model, predictive state distributions are computed. Assuming a deterministic initial state $x_0$, the result of the first step is $x_1 = f_{\mathrm{GP}}(x_0, u_0)$,

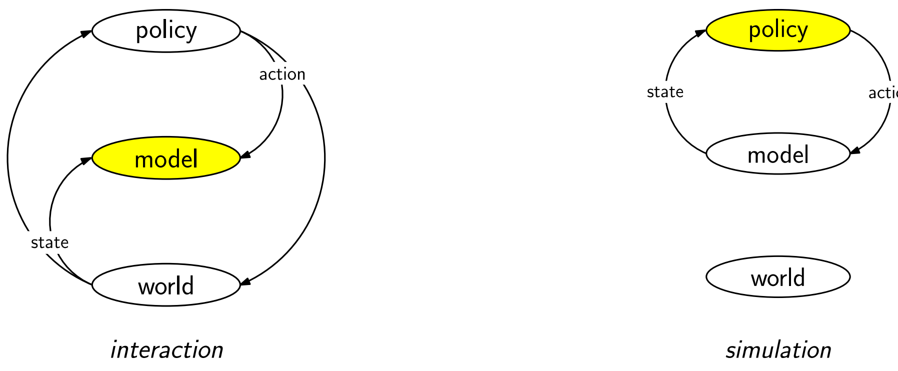


**Figure 8.2:** Two main steps of policy learning in PILCO [12]. Left: model learning by executing the policy on the real robot. Right: policy learning by predicting the long-term cost $V^{\pi}$ based on the learned dynamic model.

**Algorithm 5** High-level steps of policy learning

```
 1: Initialize control policy by a random set of parameters
 2: loop: Model learning (interaction)
 3:     Execute policy on the real system (or simulation)
 4:     Save observations
 5:     Update probabilistic dynamic GP model
 6:     loop: Policy learning (simulation)
 7:         Prediction based on learned dynamics and policy
 8:         Compute expected long-term cost
 9:         Improve policy
10:     end loop
11: end loop
12: return Optimal policy
```

that is a normally distributed random variable, as it is detailed in Chapter 2. However, at the next step, the GP needs to be evaluated at uncertain inputs, and the result is generally not Gaussian and cannot be computed analytically, therefore an approximation method needs to be used to propagate the uncertainty of the predictive distribution. Here, we briefly introduce three uncertainty propagation methods based on [22]:

- *Mean Equivalent Approximation* (MEA) is a computationally cheap approach, that is based on the approximation of a random variable only with its mean, therefore it neglects the accumulation of uncertainty in the GP, often leading to poor approximations.
- *Taylor Approximation* (TA) uses a first-order Taylor approximation of the posterior distribution given by (2.6) to propagate the uncertainty of the GP. The method is computationally more expensive then MEA, but it is able to express uncertainty propagation, resulting in better approximations.
- *Exact Moment Matching* (EMM) utilizes that the first and second moments of the posterior distribution (2.6) can be analytically computed. The algorithm matches these moments to that of a normal distribution, which can be interpreted as an optimal fit of a Gaussian to the true distribution. EMM has similar computation cost to TA, and also similar approximation error in case of one-step predictions

[22]. EMM is used in PILCO to propagate the uncertainty of the dynamics when evaluating the long-term cost.

Given that the policy can be evaluated on the probabilistic GP dynamic model using one of the uncertainty propagation methods, an optimization problem is formulated to find the optimal policy that minimizes the expected long-term cost. In [12], *gradient-based policy search* is employed to find the optimal parameters of the policy. To increase the efficiency of the policy search algorithm, the gradients of the value function with respect to the policy parameters are analytically computed for the linear and RBF policies.

The policy learning algorithm runs until the uncertainty of the probabilistic dynamic model is sufficiently low, and the policy converges. Finally, it returns the optimal control policy that can be directly applied to perform the specified control task.

## 8.3 Learning the backflip maneuver with PILCO

In this section, we discuss the possibilities to apply PILCO for learning the backflip maneuver with quadcopters. We propose various approaches to develop both model learning and policy learning specifically for backflipping, as it is illustrated in Figure 8.3.

### 8.3.1 Model learning

PILCO initializes the reinforcement learning process with a random dynamic model, and uses only measurement data to train the GP. However, real-world experiments are often costly and time consuming, moreover, it is difficult to collect data from the whole operating domain of the robot. To address this problem, we introduce two solutions in this section.

First, the nominal nonlinear model of a standard quadcopter vehicle is known, as we presented in Chapter 3. By evaluating the nominal model on a grid of the operating domain, data points can be generated to train an initial GP model and start policy learning when the error of the GP prediction compared to the output of the nominal model is low. Then, more accurate simulations can be conducted, resulting in faster convergence of the policy learning. By collecting measurement data, the GP model can be further improved

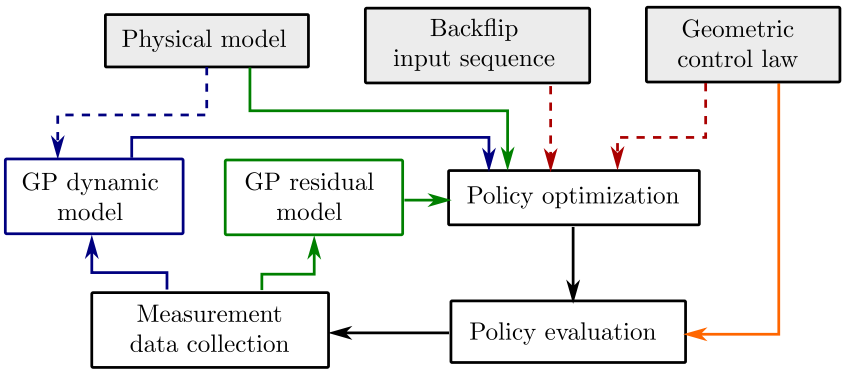


**Figure 8.3:** PILCO augmentation possibilities to learn backflipping with quadcopters. Blue and green lines correspond to model learning, red and orange lines correspond to policy learning, dashed lines denote initialization, and solid lines denote iterative evaluation or optimization.

to outperform the nominal quadcopter model in terms of representation capability of the dynamics of the real robot. However, it is an important design choice to adjust the weight of the data points generated by the nominal model compared to the measurement data. The GP model initialization and learning is depicted by blue color in Figure 8.3.

A second approach would be to augment the nominal model of the quadcopter with a Gaussian Process that represents the unknown residual dynamics, i.e., the difference between the nominal model and the real system. In Figure 8.3, the corresponding blocks and arrows are depicted by green color. The augmented model is formulated as follows:

$$\hat{x}_k = f(\hat{x}_{k-1}, u_{k-1}) + \Delta f_{\mathrm{GP}}(\hat{x}_{k-1}, u_{k-1}), \tag{8.5a}$$

$$\Delta f_{\mathrm{GP}}(\hat{x}_k, u_k) = \mathcal{GP}(\hat{x}_k, u_k) \approx \mathcal{N}(\Delta\mu(\hat{x}_k, u_k), \Delta\Sigma(\hat{x}_k, u_k)), \tag{8.5b}$$

where $f$ contains the discrete time ideal nonlinear physical model of the vehicle based on (5.1), and $\Delta f_{\mathrm{GP}}$ captures the unknown dynamics based on measurement data on the real quadcopter. However, simulating the GP augmented dynamic model (8.5) not only requires GP evaluation at uncertain inputs, but also mapping the state (which is a Gaussian random variable) through a nonlinear function $f$. The former can be handled by the approximate uncertainty propagation methods described in Section 8.2.2, but the latter needs further consideration. Similarly to [22], it is possible to approximate the state, control input and residual dynamics as jointly Gaussian distributed at every time step:

$$\begin{bmatrix} \hat{x}_k \\ u_k \\ \Delta f_{\mathrm{GP},k} \end{bmatrix} \sim \mathcal{N}(\mu_k, \Sigma_k) = \mathcal{N}\left( \begin{bmatrix} \mu_k^x \\ \mu_k^u \\ \Delta\mu_k \end{bmatrix}, \begin{bmatrix} \Sigma_k^x & \Sigma_k^{xu} & \Sigma_k^{x\Delta} \\ \star & \Sigma_k^u & \Sigma_k^{u\Delta} \\ \star & \star & \Delta\Sigma_k \end{bmatrix} \right), \tag{8.6}$$

where $\Delta f_{\mathrm{GP},k} = \Delta f_{\mathrm{GP}}(\hat{x}_k, u_k)$, $\Delta\mu_k = \Delta\mu(\hat{x}_k, u_k)$, $\Delta\Sigma_k = \Delta\Sigma(\hat{x}_k, u_k)$, and $\star$ denotes terms given by symmetry. Then, to compute the uncertainty propagation, the nominal dynamic model can be linearized around the mean of the state and input, as follows:

$$f(\hat{x}, u) \approx f(\mu^x, \mu^u) + \nabla f(\mu^x, \mu^u) \left( \begin{bmatrix} \hat{x} \\ u \end{bmatrix} - \begin{bmatrix} \mu^x \\ \mu^u \end{bmatrix} \right), \tag{8.7}$$

where $\nabla$ denotes the gradient operator. Using the linearized dynamics, the state transition characterized by (8.5) can be rewritten to the following form:

$$\mu_k^x = f(\mu_{k-1}^x, \mu_{k-1}^u) + \Delta\mu(\hat{x}_{k-1}, u_{k-1}), \tag{8.8a}$$

$$\Sigma_k^x = [\ \nabla f(\mu_{k-1}^x, \mu_{k-1}^u) \quad I\ ]\ \Sigma_{k-1}\ [\ \nabla f(\mu_{k-1}^x, \mu_{k-1}^u) \quad I\ ]^\top. \tag{8.8b}$$

Finally, the GP augmented dynamic model can be simulated using (8.8) to evaluate the long-term cost $V^\pi$ and optimize the policy. Note, that (i) instead of the Taylor approximation at (8.7), moment matching could also be used, but it is more complicated [12], and (ii) here we have omitted the update law for the full covariance matrix $\Sigma_k$ for the sake of simplicity, however, it can be found in [22].

### 8.3.2 Policy learning

Similarly to model learning, the policy learning algorithm of PILCO can be also adapted to learn the backflip maneuver. On the one hand, the policy $\pi$ can be initialized using one of the demonstrated methods, regardless of its specific structure (e.g. linear, RBF, or neural network), as it is illustrated in Figure 8.3 using red color. In case of the feedforward solution introduced in Chapter 4, the optimal control input sequence can be directly applied to tune the initial parameters of the policy. However, a single input sequence provides small amount of information, and modifying the parameters of the feedforward approach lead to poor performance, as shown in Figure 6.1. From this point of view, the geometric feedback control approach is more suitable for training an initial policy, because several different trajectories can be generated, all of them resulting in successful backflipping, as it is illustrated in Figure 6.3.

On the other hand, similar to the nominal dynamic model of the quadcopter, the nominal geometric feedback controller can also be augmented with a GP term, as follows:

$$u = \pi_{\text{geom}}(\chi) + \Delta\pi_{\text{GP}}(\chi), \tag{8.9}$$

where $u = [F,\ \tau^\top]^\top$ is the control input of the quadrotor, $\pi_{\text{geom}}(\chi)$ denote the geometric control law characterized by (5.8), and $\Delta\pi_{\text{GP}}(\chi)$ is the GP augmentation. The result is illustrated in Figure 8.3, which can be interpreted as an adaptive version of the geometric controller. In order to evaluate the control law (8.9), the random variable $\chi$ needs to be mapped through the nonlinear function $\pi_{\text{geom}}$, and the Gaussian Process $\Delta\pi_{\text{GP}}$. This case is similar to the evaluation of (8.5), and the same procedure can be applied here, as well.

An advantage of the first method (training the initial parameters of the policy) is that the policy remains flexible, therefore it can be adapted to quickly learn other acrobatic maneuvers, e.g. a double flip [34] or a barrel roll [26]. In contrast, the second method has the advantage that it is based on a mathematically well grounded, Lyapunov-stable controller, therefore the measurements on the real robot can be conducted more safely in the early phase of policy learning.

As a summary, we intend to use the flexibility and computational efficiency of policy learning by model internalization based reinforcement learning to improve the performance of the proposed algorithms for backflipping with quadcopters. In this work, only the theoretical introduction and main ideas of reinforcement learning control have been discussed, however, the subject will be further investigated by implementing the proposed methods and compare the results to the already implemented and tested controllers.

# Chapter 9

# Summary and Conclusions

The aim of this work is to develop and implement efficient control algorithms for the agile maneuvering of autonomous quadcopters. We have chosen the backflip maneuver as an example, because it is fast, highly challenging even for a human drone pilot, and emphasizes the nonlinear behaviour of the vehicle. A thorough literature survey has been conducted about the mathematical modelling of quadcopter dynamics, state-of-the-art control approaches for aggressive maneuvers, specifically for backflipping, in order to identify the most relevant results in this particular field of research. Based on the literature, we set our research goal to compare existing trajectory planning and control methods, and develop novel algorithms to overcome their observed limitations. Accordingly, we have proposed two control methods and their comparison for performing a backflip maneuver with miniature quadcopters: (i) an adaptive feedforward control by Bayesian optimization of a parametric motion primitive sequence, and (ii) trajectory planning by model-based optimization and trajectory tracking by a novel robust adaptive geometric controller.

After the theoretical foundation of the mathematical model and control approaches, we have conducted simulation studies, using the dynamic model of the Bitcraze Crazyflie 2.1 quadcopter. We have found Bayesian optimization to be efficient and suitable for learning an open-loop strategy for backflipping, the optimized parameter set worked with good performance on the simulation model of the quadcopter. However, we have also observed that the introduced method is very sensitive to parameter changes, as it quickly becomes unstable if we disturb the ideal set of parameters. Simulation studies have shown that the trajectory design for the second control method is computationally efficient, flexible, and has good performance even in case of perturbed parameters and conditions. With proper tuning of geometric control gains, we have achieved accurate trajectory tracking both for the position and orientation of the quadcopter, performing the backflip maneuver with small errors. The novel robust adaptive geometric controller guarantees robust stability of the closed loop system even in the presence of external disturbances, which is a significant improvement of the nominal control scheme both from a theoretical and practical point of view. Moreover, it not only provides robustness, but also excellent control performance as it has outperformed the nominal geometric controller in terms of minimizing the reference tracking error.

The experimental setup at SZTAKI AIMotion Lab has made it possible to implement both control methods on a real Crazyflie quadcopter. As it is displayed in Figure 7.9, the implementation was successful, the quadcopter performed the backflip with the feedforward and geometric control, as well. Feedforward control is very sensitive to parameter uncertainty, therefore we have needed to further tune the parameter set for each Crazyflie 2.1 quadcopter. On the other hand, geometric control has provided a significantly more robust method for backflipping, as feedback could reject small parameter uncertainties. The performance of the adaptive geometric tracking control has been demonstrated by performing the backflip maneuver with a bolt attached to the drone, as an external disturbance. Our results show that the proposed adaptive scheme is able to increase the control performance not only in simulation, but in real-world application, as well.

The future goal of this project is to design even more general control methods for fast, complex maneuvers of miniature quadcopters using machine learning techniques. The advantage of learning algorithms is that they require less expert knowledge, i.e. a simple task formulation is suitable to iteratively learn the desired motion. Reinforcement learning (RL), one of the most popular research topics in machine learning control, is widely applied for learning and optimization of quadcopter maneuvers by trial and error, often combined with deep neural network policies. In this work, we have introduced a model internalization based RL framework and its possible use to efficiently learn control policies for backflipping with quadcopters. In our oncoming research work, we intend to use these learning methods to perform complex maneuvers with less expert knowledge and extend the capabilities of the miniature quadcopters even more.

Another interesting direction for further development would be to combine the advantages of the two proposed backflipping methods, and design a feedback controller to track the parametrized motion primitive introduced in the feedforward method. This way, we could achieve more robust and reliable operation and also exploit the flexibility and performance of Bayesian optimization.

# Acknowledgements

I would like to thank *Tamás Péni* and *Roland Tóth* at SZTAKI for the huge amount of help and professional support when facing both theoretical and practical problems. I would also like to thank *István Harmati* at BME Department of Control Engineering and Information Technology for the useful advices regarding the documentation.

The research in this work was supported by the European Union within the framework of the National Laboratory for Autonomous Systems. (RRF-2.3.1-21-2022-00002)

# List of Figures

# Bibliography

[1] Pieter Abbeel, Adam Coates, and Andrew Y. Ng. Autonomous helicopter aerobatics through apprenticeship learning. *The International Journal of Robotics Research*, 29 (13):1608–1639, 2010.

[2] Péter Antal. Self-learning control of mechatronic systems using Gaussian Process. Bachelor's Thesis, Budapest University of Technology and Economics, 2020.

[3] Péter Antal, Tamás Péni, and Roland Tóth. Nonlinear control method for backflipping with miniature quadcopters. *IFAC-PapersOnLine*, 55(14):133–138, 2022. ISSN 2405-8963. 11th IFAC Symposium on Intelligent Autonomous Vehicles.

[4] Péter Antal, Tamás Péni, and Roland Tóth. Backflipping with miniature quadcopters by Gaussian Process based control and planning. *submitted to IEEE Transactions on Control Systems Technology*, 2022. URL `https://arxiv.org/abs/2209.14652`.

[5] Lucas M. Argentim, Willian C. Rezende, Paulo E. Santos, and Renato A. Aguiar. PID, LQR and LQR-PID on a quadcopter platform. In *Proc. of the International Conference on Informatics, Electronics and Vision*, pages 1–6, 2013.

[6] Mahdis Bisheban and Taeyoung Lee. Geometric adaptive control with neural networks for a quadrotor in wind fields. *IEEE Transactions on Control Systems Technology*, 29(4):1533–1548, 2021.

[7] Eric Brochu, Vlad Cora, and Nando Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *Computing Research Repository*, 12 2010.

[8] Francesco Bullo and Andrew Lewis. *Geometric Control of Mechanical Systems. Modeling, Analysis, and Design for Simple Mechanical Control Systems.* Springer-Verlag, 06 2004. ISBN 978-0-387-22195-3.

[9] Eduardo F. Camacho and Carlos Bordons. *Model Predictive Control.* Springer London, 2007.

[10] Ying Chen and Néstor O. Pérez-Arancibia. Lyapunov-based controller synthesis and stability analysis for the execution of high-speed multi-flip quadrotor maneuvers. In *Proc. of the American Control Conference*, pages 3599–3606, 2017.

[11] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Proc. of the 32nd Conference on Neural Information Processing Systems*, page 4759–4770, 2018.

[12] Marc Deisenroth. *Efficient Reinforcement Learning using Gaussian Processes.* PhD thesis, Karlsruhe Institute of Technology, 11 2010.

[13] Marc Deisenroth, Carl Rasmussen, and Dieter Fox. Learning to control a low-cost manipulator using data-efficient reinforcement learning. In *Robotics: Science and Systems*, 06 2011.

[14] Marc Deisenroth, Dieter Fox, and Carl Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37:408–423, 02 2015.

[15] Ayman El-Badawy and Mohamed Bakr. Quadcopter aggressive maneuvers along singular configurations: An energy-quaternion based approach. *Journal of Control Science and Engineering*, 2016, 01 2016.

[16] Karam Eliker, Guoqing Zhang, Said Grouni, and Weidong Zhang. An optimization problem for quadcopter reference flight trajectory generation. *Journal of Advanced Transportation*, 2018:1–15, 07 2018.

[17] Thilina Fernando, Jiten Chandiramani, Taeyoung Lee, and Hector Gutierrez. Robust adaptive geometric tracking controls on SO(3) with an application to the attitude dynamics of a quadrotor uav. In *Proc. of the IEEE Conference on Decision and Control*, pages 7380–7385, 12 2011.

[18] P. Frazier. A tutorial on Bayesian optimization, 2018.

[19] Emil Fresk and George Nikolakopoulos. Full quaternion based attitude control for a quadrotor. In *Proc. of the 2013 European Control Conference*, pages 3864–3869, 2013.

[20] Julian Förster. System identification of the crazyflie 2.0 nano quadrocopter. Bachelor's Thesis, ETH Zurich, Zurich, 2015.

[21] Farhad Goodarzi, Daewon Lee, and Taeyoung Lee. Geometric adaptive tracking control of a quadrotor unmanned aerial vehicle on SE(3) for agile maneuvers. *Journal of Dynamic Systems Measurement and Control*, 137, 06 2015.

[22] L. Hewing, J. Kabzan, and M. N. Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28 (6):2736–2743, 2019.

[23] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2:2096–2103, 7 2017.

[24] Jemin Hwangbo, Inkyu Sa, Roland Siegwart, and Marco Hutter. Control of a quadrotor with reinforcement learning. *IEEE Robotics and Automation Letters*, 2(4):2096–2103, Oct 2017. ISSN 2377-3774.

[25] A. Isidori. *Nonlinear Control Systems.* Communications and Control Engineering. Springer London, 1995. ISBN 9783540199168.

[26] Elia Kaufmann, Antonio Loquercio, Rene Ranftl, Matthias Mueller, Vladlen Koltun, and Davide Scaramuzza. Deep drone acrobatics. In *Robotics: Science and Systems*, 07 2020.

[27] Elia Kaufmann, Leonard Bauersfeld, and Davide Scaramuzza. A benchmark comparison of learned control policies for agile quadrotor flight. In *Proc. of the IEEE International Conference on Robotics and Automation*, 02 2022.

[28] Daewon Lee and Shankar Sastry. Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. *International Journal of Control, Automation and Systems*, 7(3):419–428, 06 2009.

[29] Taeyoung Lee. *Computational Geometric Mechanics and Control of Rigid Bodies*. PhD thesis, University of Michigan, 2008.

[30] Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. Geometric tracking control of a quadrotor UAV on SE(3). In *Proc. of the 49th IEEE Conference on Decision and Control*, pages 5420–5425, 2010.

[31] Taeyoung Lee, Melvin Leok, and N. Harris McClamroch. Nonlinear robust tracking control of a quadrotor UAV on SE(3). *Asian Journal of Control*, 15:391–408, 2013.

[32] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

[33] M. Liu, G. Chowdhary, B. Castra da Silva, S. Liu, and J. P. How. Gaussian processes for learning and control: A tutorial with examples. *IEEE Control Systems Magazine*, 38(5):53–86, 2018.

[34] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 1642–1648, 2010.

[35] Robert Mahony, Vijay Kumar, and Peter Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, 19(3): 20–32, 2012.

[36] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. of the IEEE International Conference on Robotics and Automation*, pages 2520–2525, 2011.

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, 2013.

[38] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei Rusu, Joel Veness, Marc Bellemare, Alex Graves, Martin Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–33, 02 2015.

[39] Artem Molchanov, Tao Chen, Wolfgang Honig, James A. Preiss, Nora Ayanian, and Gaurav S. Sukhatme. Sim-to-(multi)-real: Transfer of low-level robust control policies to multiple quadrotors. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, pages 59–66, 11 2019.

[40] Fernando Nogueira. Bayesian Optimization: Open source constrained global optimization tool for Python, 2014–.

[41] Jacopo Panerati, Hehui Zheng, SiQi Zhou, James Xu, Amanda Prorok, and Angela P. Schoellig. Learning to fly—a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, 2021.

[42] James A. Preiss, Wolfgang Hönig, Gaurav S. Sukhatme, and Nora Ayanian. Crazyswarm: A large nano-quadcopter swarm. In *Proc. of the International Conference on Robotics and Automation*, pages 3299–3304. IEEE, 2017.

[43] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, 2006.

[44] S. Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, pages 621–635, 2018.

[45] Ewoud Smeur, Q. Chu, and Guido Croon. Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles. *Journal of Guidance, Control, and Dynamics*, 39:1–12, 12 2015.

[46] Yunlong Song and Davide Scaramuzza. Policy search for model predictive control with application to agile drone flight. *IEEE Transactions on Robotics*, 38:2114–2130, 8 2022.

[47] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Proc. of the Conference on Robot Learning*, 2020.

[48] Yunlong Song, Mats Steinweg, Elia Kaufmann, and Davide Scaramuzza. Autonomous drone racing with deep reinforcement learning. In *Proc. of the IEEE International Conference on Intelligent Robots and Systems*, pages 1205–1212, 2021.

[49] Matthew Turpin, Nathan Michael, and Vijay R. Kumar. Trajectory design and control for aggressive formation flight with quadrotors. *Autonomous Robots*, 33:143–156, 2012.

[50] Roland Tóth and Maarten Schoukens. *Lecture notes in Machine Learning for Systems and Control (5SC28).* Eindhoven University of Technology, 2019.