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

Nonlinear control design for backflipping with miniature quadcopters

Antal Péter

Scientific Conference of Students

Consultants:

Dr. Péni Tamás
Senior research fellow
SZTAKI Systems and Control Lab

Dr. Tóth Roland
Senior research fellow
SZTAKI Systems and Control Lab

Supervisor:

Dr. Harmati István
Associate professor
BME IIT

Budapest, 2021

Table of Contents

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 driver, 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. These vehicles are often designed based on bio-inspirations, i.e. they mimic the behaviour and flight capabilities of flying insects or birds. 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. Three micro aerial vehicles are shown in Figure 1.1: 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 In these cases, classical flight controllers designed for a linearized dynamical model are no longer applicable and more advanced control methods that exploit the entire operating domain are needed.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.

At AIMotion Lab of ELKH SZTAKI there is a flying arena with five Bitcraze Crazyflie 2.1 miniature quadrotors. These drones are designed for research and education purposes, therefore they are both open hardware and open source, making it possible to implement our own algorithms on-board. Due to their low weight and relatively small mechanical

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

power no on-board cameras can be used, therefore 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 not only because it emphasizes the nonlinearity of the dynamical model, but also requires special considerations regarding the attitude representation and control. In the literature, there are several different approaches to perform the flip maneuver. In [1], energy-based control is applied to overcome the uncontrollability of the quadcopter at singular configurations to follow a circular or clothoidal reference trajectory. Machine learning approaches are utilized in many cases, for example to imitate the maneuver performed by an expert drone pilot with apprenticeship learning [2], or design time-optimal trajectories with deep reinforcement learning [3] and learn acrobatic maneuvers [4, 5].

A simple learning strategy for adaptive open-loop control is proposed in [6], based on the optimization of a parametric motion primitive sequence. As backflipping pushes the actuators of the quadcopter to their physical limits, the 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.

Geometric control is a nonlinear approach for attitude feedback control of rigid bodies in 3D space. In [7], 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. A control law is proposed, on which other researchers build and extend for real-time trajectory generation and aggressive, agile maneuvering [8, 9] .

In this work, firstly the theoretical background of quadrotor dynamics and control is demonstrated. In Sections 3-4, two control approaches are proposed to perform a flip maneuver: open-loop control based on motion primitive optimization, and geometric tracking control for an optimized reference trajectory. In Section 5, the simulation framework, implementation and results are discussed for the dynamical model of the Crazyflie 2.1 quadcopter. The experimental setup, implementation on the real robot, and measurement results are demonstrated in Section 6, with a comparison of the two control approaches. Finally, the conclusion and future work are summarized in Section 7.

2 Mathematical model of a quadcopter

In this section, the mathematical model of a quadcopter is presented, mainly based on [10]. 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 mapping between the control inputs and propeller angular velocities.

2.1 Coordinate frames

There are two main approaches to characterize the quadcopter coordinate frames, namely '×' and '+' configurations. Due to the practicality of on-board camera placement the earlier is more common, therefore we use '×' configuration in this work. With choosing one of the two configurations, three main frames are used: 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. The three frames are displayed in Figure 2.1, with the body frame both in 'X' and '+' configuration. The transformation from $\mathcal{F}^i$ to $\mathcal{F}^v$ is a translation, and from $\mathcal{F}^v$ to $\mathcal{F}^b$ a rotation. In the figure, the rotation is described by the roll, pitch, and yaw (RPY) Euler angles, denoted by ϕ, θ, and ψ, respectively. However, we will also present other techniques to characterize this rotation in Section 2.3.

The matrix representation of the transformation between the vehicle and body frames can be derived from the RPY angle representation as follows:

$$\begin{aligned} R_v^b &= Rot(x,\phi)Rot(z,\theta)Rot(z,\psi) = \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_\phi & S_\phi \\ 0 & -S_\phi & C_\phi \end{bmatrix} \begin{bmatrix} C_\theta & 0 & -S_\theta \\ 0 & 1 & 0 \\ S_\theta & 0 & C_\theta \end{bmatrix} \begin{bmatrix} C_\psi & S_\psi & 0 \\ -S_\psi & C_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} = \\ &= \begin{bmatrix} C_\psi C_\theta & C_\theta S_\psi & -S_\theta \\ C_\psi S_\phi S_\theta - C_\phi S_\psi & C_\phi C_\psi + S_\phi S_\theta S_\psi & S_\phi C_\theta \\ S_\phi S_\psi + C_\phi C_\psi S_\theta & C_\phi S_\theta S_\psi - C_\psi S_\phi & C_\phi C_\theta \end{bmatrix}, \end{aligned} \tag{2.1}$$

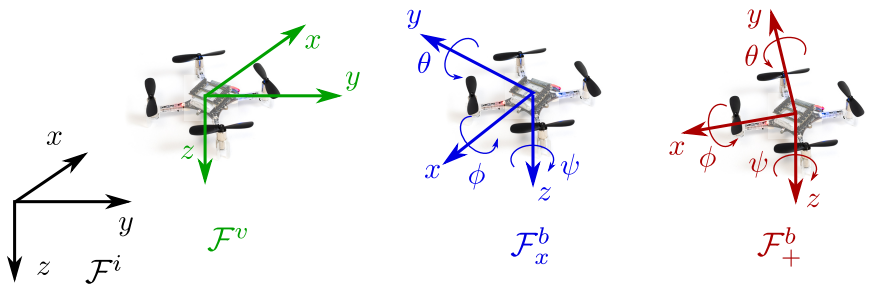


Figure 2.1: Inertial, vehicle, and body frames describing the geometric relations of the vehicle and the environment. The body frame is depicted both in '×' and '+' configurations.

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

$$R_b^v = \left(R_v^b\right)^{-1} = \left(R_v^b\right)^\top . \tag{2.2}$$

2.2 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

$$f = m\frac{\mathrm{d}V}{\mathrm{d}t_i}, \tag{2.3}$$

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 (2.3) results from these terms, as

$$f = R_b^v \begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}, \tag{2.4}$$

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

2.3 Rotational dynamics

There are multiple ways to describe the rotational motion of a rigid body in 3D space, out of which three approaches will be presented in this work. The most common and simple way is to use local coordinates, i.e. describe the attitude with three axes, and three rotation angles around them. An example is the RPY angle representation mentioned in Section 2.1, where the three axes are the body z, y, x axes, and the three angles are the yaw (ψ), pitch (θ) and roll (ϕ), 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, in quadcopter navigation we use RPY (or Euler) angles only for slower maneuvering and trajectory tracking, but in case of an aggressive maneuver, there are more efficient approaches.

Another possible representation of rigid body rotations is using orientation 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 will only use this representation for trajectory design.

The third representation of orientation we describe is the rotation matrix. Any rigid body rotation can be characterized by a 3×3 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 3×3 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 will be 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 2.2 and the rotational dynamics with one of the three attitude representations together characterize the dynamic model of the quadcopter.

2.3.1 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

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \cos\theta\sin\phi \\ 0 & -\sin\phi & \cos\theta\cos\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = W \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}, \tag{2.5}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = W^{-1} \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{\sin\phi}{\cos\theta} & \frac{\cos\phi}{\cos\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{2.6}$$

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

$$\tau = \frac{\mathrm{d}J\omega}{\mathrm{d}t_i}, \tag{2.7}$$

where τ is the torque acting on the body, J is the inertia, and ω 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 (2.7) are calculated with respect to the inertial frame, the formulae of derivation in a moving frame have to be applied, resulting in

$$\frac{\mathrm{d}J^b\omega^b}{\mathrm{d}t_i} = \frac{\mathrm{d}J^b\omega^b}{\mathrm{d}t_b} + \left(\omega^b \times J^b\omega^b\right) = \underbrace{\frac{\mathrm{d}J^b}{\mathrm{d}t_i}\omega^b}_{=0} + J^b\frac{\mathrm{d}\omega^b}{\mathrm{d}t_i} + \left(\omega^b \times J^b\omega^b\right) = \tau, \tag{2.8}$$

$$\Downarrow$$

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \left(J^b\right)^{-1}\left(\tau - \omega^b \times J^b\omega^b\right). \tag{2.9}$$

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

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}\frac{J_y - J_z}{J_x}\cdot q\cdot r + \frac{\tau_x}{J_x}\\ \frac{J_z - J_z}{J_x}\cdot p\cdot r + \frac{\tau_y}{J_y}\\ \frac{J_x - J_y}{J_z}\cdot p\cdot q + \frac{\tau_z}{J_z}\end{bmatrix}. \tag{2.10}$$

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 (2.1) and (2.4), for the rotation from (2.6), and for the rotational velocity from (2.10). Gathering these to a matrix form, the following state space representation is formulated:

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{V}_x\\ \dot{V}_y\\ \dot{V}_z\\ \dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\\ \dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix}V_x\\ V_y\\ V_z\\ -(\sin\phi\cdot\sin\psi + \cos\phi\cdot\cos\psi\cdot\sin\theta)\frac{F}{m}\\ -(\cos\phi\cdot\sin\psi\cdot\sin\theta - \cos\psi\cdot\sin\phi)\frac{F}{m}\\ -(\cos\phi\cdot\cos\theta)\frac{F}{m} + g\\ p + \sin\phi\cdot\tan\theta\cdot q + \cos\phi\cdot\tan\theta\cdot r\\ \cos\phi\cdot q - \sin\phi\cdot r\\ \frac{\sin\phi}{\cos\theta}\cdot q + \frac{\cos\phi}{\cos\theta}\cdot r\\ \frac{J_y - J_z}{J_x}\cdot q\cdot r + \frac{\tau_x}{J_x}\\ \frac{J_z - J_x}{J_y}\cdot p\cdot r + \frac{\tau_y}{J_y}\\ \frac{J_x - J_y}{J_z}\cdot p\cdot q + \frac{\tau_z}{J_z}\end{bmatrix}. \tag{2.11}$$

2.3.2 Quaternion based model

The second presented approach for rigid body attitude representation is using orientation quaternions. A quaternion is a hyper complex number of rank 4, which can be represented in multiple forms, see for example equation (2.12). The quaternion elements from q_1 to q_3 are called the vector part of the quaternion, while q_0 is the scalar part.

$$q = q_0 + q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^\top = \begin{bmatrix} q_0 \\ \mathbf{q} \end{bmatrix} \tag{2.12}$$

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.

$$p \otimes q = \underbrace{p_0 q_0 - \mathbf{p} \cdot \mathbf{q}}_{\text{scalar part}} + \underbrace{p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p} \times \mathbf{q}}_{\text{vector part}}, \tag{2.13}$$

$$p \otimes q = Q(p)q = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p_2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} p_0 q_0 - p_1 q_1 - p_2 q_2 - p_3 q_3 \\ p_0 q_1 + p_1 q_0 + p_2 q_3 - p_3 q_2 \\ p_0 q_2 - p_1 q_3 + p_2 q_0 + p_3 q_1 \\ p_0 q_3 + p_1 q_2 - p_2 q_1 + p_3 q_0 \end{bmatrix}. \tag{2.14}$$

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

$$\|q\| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}. \tag{2.15}$$

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

$$q^* = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{bmatrix}^\top. \tag{2.16}$$

Given that the angular velocity is in the body frame, the derivative of the quaternion rotation in the dynamic equations has the form [11]

$$\dot{q} = -\frac{1}{2}\omega^b \otimes q. \tag{2.17}$$

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 (2.4) can be characterized by two quaternion products, namely

$$f = q \otimes \begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} \otimes q^* + \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}, \tag{2.18}$$

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

The state space representation using (2.11) and the above expressions has the following form:

$$\begin{bmatrix} \dot{r} \\ \dot{V} \\ \dot{q} \\ \dot{\omega}^b \end{bmatrix} = \begin{bmatrix} V \\ q \otimes \frac{1}{m} \begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} \otimes q^* + \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \\ -\frac{1}{2}\omega^b \otimes q \\ \left(J^b\right)^{-1} \left(\tau - \omega^b \times J^b \omega^b\right) \end{bmatrix}. \tag{2.19}$$

2.3.3 Quadcopter model on $SE(3)$

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 Euclidean group, $SE(3)$. The proper mathematical background of group theory and Lie-groups for rigid body orientation can be found for example in [12, 13]. In this paper, only the most important principles are described with an emphasis on applications.

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

$$A \in O(n) \Leftrightarrow AA^\top = I, \tag{2.20}$$

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* $SO(n)$ 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

$$\left\{ \begin{pmatrix} R & v \\ 0 & 1 \end{pmatrix} \;\middle|\; R \in SO(n) \text{ and } v \in \mathbb{R}^n \right\}. \tag{2.21}$$

The matrix R achieves the rotation, and the vector v the translation. The result is the *special Euclidean group* $SE(n)$, which is homeomorphic to $\mathbb{R}^n \times SO(n)$, because the rotation matrix and translation vectors may be chosen independently.

Rigid body translation and rotation is described in the 3D space, therefore the representation is in the special Euclidean group $SE(3)$, and the rotation matrix in the special orthogonal group $SO(3)$. The attitude dynamics have a simple form using rotation

matrices, namely

$$\dot{R} = R\hat{\omega}, \tag{2.22}$$

$$\dot{\omega} = J^{-1}\left(\tau - \omega \times J\omega\right), \tag{2.23}$$

where R is the rotation matrix between the vehicle and body frames, and the *hat map* $\hat{\cdot} : \mathbb{R}^3 \to SO(3)$ 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 (2.4) and (2.9) are omitted for clarity, the inertia and angular velocity are always defined in the body frame.

2.4 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 will be described in this section.

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

$$T_i = k\omega_i^2, \tag{2.24}$$

where k is the *thrust constant*.

The direction of rotor thrust, angular velocity and the numbering of propellers are displayed in Figure 2.2. 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

$$\tau_x = \frac{l}{\sqrt{2}}(T_3 + T_4 - T_1 - T_2) = \frac{l}{\sqrt{2}}k\left(\omega_3^2 + \omega_4^2 - \omega_1^2 - \omega_2^2\right), \tag{2.25}$$

$$\tau_y = \frac{l}{\sqrt{2}}(T_1 + T_4 - T_2 - T_3) = \frac{l}{\sqrt{2}}k\left(\omega_1^2 + \omega_4^2 - \omega_2^2 - \omega_3^2\right). \tag{2.26}$$

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

$$\tau_z = b\left(\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2\right), \tag{2.27}$$

where b is the *drag constant*. We can write the input equations (2.24)–(2.27) in a matrix form, resulting in

$$\begin{bmatrix} F \\ \tau \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -\frac{l}{\sqrt{2}} & -\frac{l}{\sqrt{2}} & \frac{l}{\sqrt{2}} & \frac{l}{\sqrt{2}} \\ \frac{l}{\sqrt{2}} & -\frac{l}{\sqrt{2}} & -\frac{l}{\sqrt{2}} & \frac{l}{\sqrt{2}} \\ \frac{b}{k} & -\frac{b}{k} & \frac{b}{k} & -\frac{b}{k} \end{bmatrix} \begin{bmatrix} k\omega_1^2 \\ k\omega_2^2 \\ k\omega_3^2 \\ k\omega_4^2 \end{bmatrix}. \tag{2.28}$$

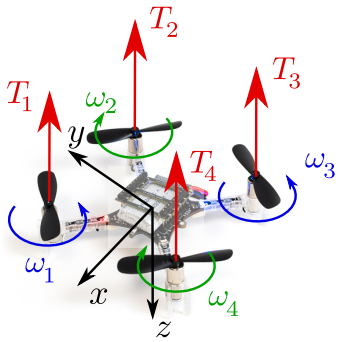


Figure 2.2: Thrusts and angular velocities of the rotors with the body frame.

Since there is a constant mapping between the motor angular velocity and the vector of collective thrust and moments, we consider the $\left[F,\ \tau^\top\right]^\top$ 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. The connection of rotor angular velocity, rotor thrust and torque are often determined experimentally, resulting in more complex models than the equations described by (2.24) and (2.27) [14].

3 Optimization-based open-loop control

The flip maneuver is a 360 degree rotation around one of the horizontal axes of the quadcopter body frame. In the literature there are several different approaches to perform the maneuver, for example based on apprenticeship learning [2], energy-based control [1], or Lyapunov method [15]. We decided to first apply an open-loop control method based on the optimization of a parametrized motion primitive sequence [6].

In this section, firstly the 2D equations of motion of the quadrotor are derived that is suitable for the applied control method. Then the parametrized primitive is described which consists of the sections of the maneuver, both beginning and ending in hover mode. Finally, the optimization problem of the primitive parameters is presented, and two different methods for the solution.

3.1 2D Quadrotor model

In this section, a two dimensional dynamic model of the quadcopter is used as the desired trajectory of the flip motion is within the $x - z$ plane of the body frame. The model consists of the thrust and moment of the four rotors, and the gravitational field. Other dynamics, such as the vehicle yaw are stabilized separately and are now ignored. Unlike the maneuver described in [6], we use $\times$ configuration as two rotors can produce a larger torque for the flip than one. From the Newtonian equations of motion using the direction conventions displayed in Figure 3.1 and Euler angle representation:

$$m\ddot{x} = -(T_1 + T_2 + T_3 + T_4)\sin\theta, \tag{3.1}$$

$$m\ddot{z} = -(T_1 + T_2 + T_3 + T_4)\cos\theta + mg, \tag{3.2}$$

$$J_{yy}\ddot{\theta} = l(T_1 + T_4 - T_2 - T_3), \tag{3.3}$$

where θ is the pitch angle, g is the gravitational acceleration, m is the mass of the drone, l is the distance of a propeller from the center of mass of the vehicle, J_{yy} is the moment of inertia about the axis of the flip (out-of-plane principal axis), and T_i, $i = 1, 2, 3, 4$ are the

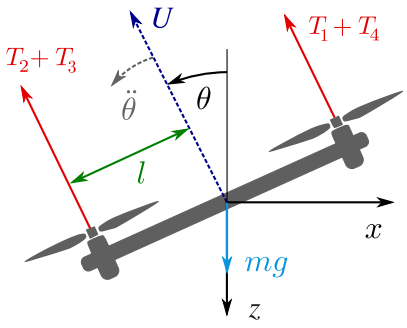


Figure 3.1: The vehicle frame, the orientation and forces acting on the quadcopter in two dimensions.

thrusts of each rotor. The collective acceleration U is the sum of all rotor thrusts divided by the mass of the drone,

$$U = \frac{T_1 + T_2 + T_3 + T_4}{m}. \tag{3.4}$$

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

3.2 Parametrized primitive of the maneuver

The goal of the maneuver is to perform a flip, therefore the states at the start and end should be the same, except for the pitch angle which is offset by 2π. From these considerations, the initial and final state conditions for the maneuver can be formulated as:

$$x_0 = x_f = 0, \tag{3.5a}$$
$$z_0 = z_f = 0, \tag{3.5b}$$
$$\dot{x}_0 = \dot{x}_f = \dot{z}_0 = \dot{z}_f = 0, \tag{3.5c}$$
$$\theta_f = \theta_0 + 2\pi = 0. \tag{3.5d}$$

We use a simple control approach without explicit optimization for execution time. Research has shown that for different types of dynamic systems, bang-bang control strategies provide near-optimal results, with small complexity [16, 17]. We use such control actions within a restricted control envelope, denoted as a range of accelerations $[\underline{\beta}, \overline{\beta}]$, in order to take into consideration modelling uncertainties and reserve some resources for the onboard stabilizing controller. This restriction can be mathematically formulated for the rotating rotor thrusts as

$$T_{\min} \leq \frac{m\underline{\beta}}{4} \leq F_i \leq \frac{m\overline{\beta}}{4} \leq T_{\max}. \tag{3.6}$$

The control strategy of the maneuver consists of five sections, all of which has two constant inputs, the desired collective acceleration U_{des}, and desired angular acceleration $\ddot{\theta}_{\text{des}}$. These five steps are illustrated in Figure 3.2, and defined as follows.

1. **Accelerate:** Gain elevation and kinetic energy with nearly maximum collective acceleration, while rotating slowly to the negative direction.
2. **Start Rotate:** Increase angular velocity with torque, i.e. maximal differential thrust.
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 nearly maximal collective thrust, and try to get back to hover mode.

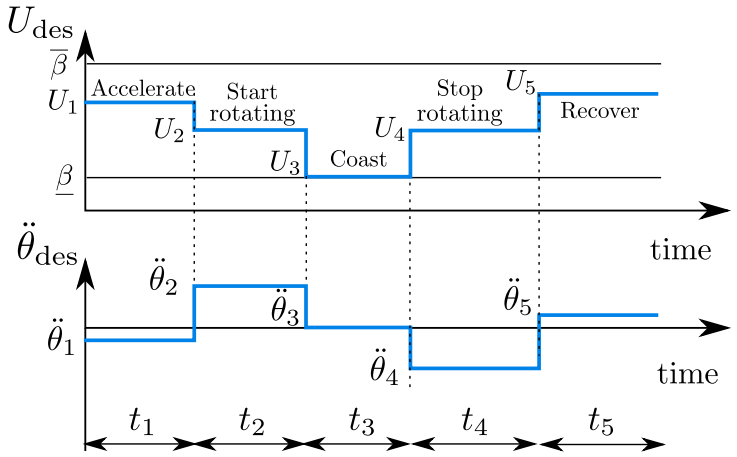


Figure 3.2: Parametrized flip primitive.

Each section has three parameters, the collective acceleration U_i, duration t_i, and angular acceleration $\ddot{\theta}_i$. Since we use the properties of bang-bang control, U_i and $\ddot{\theta}_i$ fully determine each other. Based on [6], we select

$$P = \begin{bmatrix} U_1 & t_1 & t_3 & U_5 & t_5 \end{bmatrix}^\top \tag{3.7}$$

as the parameters to optimize, and calculate the other ten quantities as detailed in the paper. The calculation of an initial guess is also discussed in [6], which we utilize in the implementation.

From the parameters, the control input vector can be calculated as

$$\begin{bmatrix} F_i \\ \tau_i \end{bmatrix} = \begin{bmatrix} mU_i \\ 0 \\ J_{yy}\ddot{\theta}_i \\ 0 \end{bmatrix}, \tag{3.8}$$

where m is the mass of the quadcopter.

3.3 Parameter optimization via numerical simulations

The five parameters are tuned in order to minimize the norm of the final state error E, mathematically described as

$$E = \begin{bmatrix} x_f & z_f & \dot{x}_f & \dot{z}_f & \theta_f \end{bmatrix}^\top, \tag{3.9a}$$

$$P^* = \arg\min_P \|E\|_2. \tag{3.9b}$$

In the original paper [6], the numerical optimization of the five parameters is based on iterative learning, 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. Thus we have chosen Bayesian optimization which does not require any functional form and is suitable for global optimization of cost functions that are expensive to evaluate [18, 19, 20].

The main concept of Bayesian optimization is to treat the unknown objective function as a random function in a parametric structure, fit the model parameters using Bayesian inference techniques, and maximize the output of the fitted surrogate model. The high-level steps of the optimization method are described in Algorithm 1. There are multiple

Algorithm 1 Basic pseudo-code for Bayesian optimization [18]

1: Place a Gaussian process prior on f
2: Observe f at n_0 initial points, set $n = n_0$
3: **while** $n \leq N$ **do**
4: Update the posterior probability distribution on f using all available data
5: Let x_n be a maximizer of the acquisition function over x, where the acquisition function is computed using the current posterior distribution
6: Observe $y_n = f(x_n)$
7: $n = n + 1$
8: **end while**
9: Return a solution: either the point evaluated with the largest $f(x)$, or the point with the largest posterior mean

methods to define the parametric structure, the most common of which is using *Gaussian processes* (GPs) [21]. GP provides a flexible, nonlinear structure representing a single layer neural network, depends on relatively few parameters and its training is fast and efficient. Moreover, the covariance of GP gives a measure of the uncertainty of the approximation which is then used to systematically select the next evaluation point of the optimizer, and thus help the balance between exploration and exploitation.

By definition, a Gaussian process $\mathcal{GP} : \mathbb{R}^d \to \mathbb{R}$ is a mapping that assigns to every point $x \in \mathbb{R}^d$ a random variable $\mathcal{GP}(x) \in \mathbb{R}$ such that for any finite set $x^{(1)} \dots x^{(n)}$ the joint probability distribution of $\mathcal{GP}\left(x^{(1)}\right), \dots, \mathcal{GP}\left(x^{(n)}\right)$ is Gaussian with mean m and covariance K, expressed as

$$m = \left[m\left(x^{(1)}\right), \dots m\left(x^{(n)}\right)\right]^\top \tag{3.10a}$$

$$[K]_{ij} = \kappa\left(x_i, x_j\right), \tag{3.10b}$$

where $[\cdot]_{ij}$ denotes the (i, j)-th entry of a matrix and κ is a suitable kernel function. Both the mean and kernel depend on the *hyperparameters* denoted by θ, which are tuned during the optimization process. The goal is to learn the objective function $f : \mathbb{R}^d \to \mathbb{R}$ using a training set generated from the function evaluations $(x, f(x))$. An intuitive interpretation of the training is to find the GP that is most probable to have generated the training set, assuming that f is a sample of the GP.

The training starts with model selection, i.e. choosing a suitable mean and kernel function. By subtracting the mean of the data from all elements, constant zero mean is a common choice. The kernel is the core of the GP, it characterizes the function class in which we search the objective function. For smooth functions with approximately constant characteristic length the Squared Exponential kernel is a common and good choice, but there are also more complex kernel functions, e.g. the Matérn class.

Next step of the training is hyperparameter tuning, most commonly obtained by maximizing the *marginal likelihood* of the training samples with respect to θ. For proper scaling, often the logarithm of the likelihood is maximized, characterized by

$$\log p(\overline{y} \mid X, \theta) = \log \int_{-\infty}^{\infty} p(\overline{y} \mid f, X, \theta) p(f \mid \theta) \mathrm{d}f, \tag{3.11}$$

where $\mathcal{T} = (x^{(i)}, \overline{y}^{(i)})_{i=1}^{n}$ is the training set, $X = [x^{(1)} \dots x^{(n)}]$, $\overline{y} = [\overline{y}^{(1)} \dots \overline{y}^{(n)}]^\top$ and $p(y \mid X, \theta)$ denotes the n dimensional joint Gaussian distribution of $\mathcal{GP}(x^{(1)}) \dots \mathcal{GP}(x^{(n)})$. Since the gradient of the expression is easy to evaluate, a gradient ascent algorithm can be applied to maximize the marginal likelihood.

The evaluation of a trained GP, i.e. the approximation of f at a test point $x_* \in \mathbb{R}^d$ is calculated as follows. We take the $n+1$ dimensional joint distribution

$$p([y^{(1)}, \dots, y^{(n)}, y] \mid [X, x_*], \theta),$$

and calculate the one dimensional conditional distribution

$$p(y \mid x_*, y^{(1)} = \overline{y}^{(1)}, \dots, y^{(n)} = \overline{y}^{(n)}, X, \theta).$$

The mean of this distribution m_* is the approximation for $f(x_*)$, and the variance σ_* gives a measure of the uncertainty of the regression. The evaluation of a GP requires only elementary matrix operations, therefore it is computationally efficient.

With a trained GP model and rules for the evaluation, we need an acquisition function to determine the next test point. The goal of the acquisition function is to balance between exploration and exploitation while trying to find the global maximum. The trade-off between exploration and exploitation is usually provided by maximizing the sum of the mean and variance of the posterior distribution scaled by different (and time dependent) gains. Common acquisition functions are expected improvement, knowledge gradient, and entropy search.

With the mathematical model of the quadcopter and a suitable optimization algorithm, it is possible to simulate the maneuver with different parameter sets, optimize the motion, and implement on the vehicle. For the implementation of the open-loop flip, a stabilizing flight controller is also required to balance the quadcopter at the beginning and after the end of the maneuver. Setpoint stabilization is a common task in quadcopter control, a classical LTI feedback controller is suitable designed for the linearized dynamics around hovering, for example PID or LQR.

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 modeling of the quadcopter dynamics.

4 Trajectory planning and geometric tracking control

The second presented approach for performing a flip maneuver is optimization-based trajectory planning, and tracking the designed trajectory with geometric control. An important advantage of this approach compared to the open-loop control is generality, i.e. the proposed trajectory design method and motion control can be applied to a variety of different maneuvers, while the presented open-loop approach is designed specifically to the backflip maneuver.

Another important aspect is that using feedback control instead of an open-loop strategy aims to balance the negative effect of model uncertainties, external disturbances, and noise for the performance of the controller. However, classical feedback control algorithms (PID, LQR) fail for large roll and pitch angles, therefore a more complex, nonlinear control method is required. In the literature, such controllers include quaternion based attitude control [11], adaptive incremental nonlinear dynamic inversion [22], and machine learning approaches mainly based on deep reinforcement learning [3, 4, 5].

Geometric control of a rigid body is similar to quaternion based control in the sense that it is nonlinear and covers the whole operating domain of the quadcopter state space. However, literature has shown that geometric control approaches have better performance and stability guarantees when following complex trajectories or performing aggressive maneuvers [7, 8, 9]. In this section, the trajectory planning with quaternion attitude representation and the geometric tracking control of the designed trajectory are presented.

4.1 Geometric tracking control

The nonlinear geometric tracking control used in this work is based on the ones presented in [7] and [8], designed to track a 3D trajectory on $SE(3)$. To synthesize the control law (2.4), (2.22) and (2.23) are used to describe the dynamics of the quadcopter on $SE(3)$, gathered here for clarity:

$$m\ddot{r} = (-FR + mg)e_3, \tag{4.1a}$$

$$\tau = J\dot{\omega} + \omega \times J\omega, \tag{4.1b}$$

$$\dot{R} = R\hat{\omega}, \tag{4.1c}$$

where $r \in \mathbb{R}^3$ is the position of the drone in the inertial frame, $\omega \in \mathbb{R}^3$ is the angular velocity, $R \in SO(3)$ is the rotation matrix between the vehicle and body frames, $F \in \mathbb{R}$ is the collective thrust, and $e_3 = [0, 0, 1]^\top$. The hat operator $\hat{\cdot} : \mathbb{R}^3 \to SO(3)$ is defined by the condition that $\hat{x}y = x \times y$ for all $x, y \in \mathbb{R}^3$. Parameters are the mass of the drone m, the gravity g, and the inertia J.

Following the attitude control method proposed in [7], the *control law* for the force and torque inputs is defined as

$$F = (-K_r e_r - K_v e_v + mge_3 + m\ddot{r}_d)Re_3, \tag{4.2a}$$

$$\tau = -K_R e_R - K_\omega r_\omega + \omega \times J\omega, \tag{4.2b}$$

with gain matrices $K_r, K_v, K_R, K_\omega \in \mathbb{R}^{3\times3}$, and error terms

$$e_r = r - r_d, \tag{4.3a}$$

$$e_v = \dot{r} - \dot{r}_d, \tag{4.3b}$$

$$e_R = \frac{1}{2\sqrt{1 + \operatorname{tr}\left(R_d^\top R\right)}} \left(R_d^\top R - R^\top R_d\right)^\vee, \tag{4.3c}$$

$$e_\omega = \omega - R^\top R_d \omega_d, \tag{4.3d}$$

where r_d, R_d and ω_d are the position, orientation and angular velocity reference, $\operatorname{tr}(\cdot)$ is the trace operator, and the vee operator $(\cdot)^\vee$ is the inverse of the hat operator such that $(\cdot)^\vee : SO(3) \to \mathbb{R}^3$. Similarly to [9] we use diagonal matrix gains instead of scalars, because it works much better on the real system this way. With control gains selected carefully, the proposed attitude control approach is proved to be stable in the full space of rotation matrices (excluding the exact inversion), as derived in [7].

The geometric controller demonstrated in this work has two operating modes: tracking of reference trajectories described by time functions of the flat outputs $x_d = [r_d^\top, \psi_d]^\top$, and tracking the reference flip trajectory described by time functions of the position and pitch angle. We use the former for takeoff, land and hovering, and the latter during the backflip maneuver. In case of trajectory planning for the flat outputs, the reference is designed such that $R_d = [r_1, r_2, r_3]$ and [8]

$$r_1 = r_2 \times r_3, \tag{4.4a}$$

$$r_2 = \frac{r_3 \times [\cos\psi_d, \sin\psi_d, 0]}{\|r_3 \times [\cos\psi_d, \sin\psi_d, 0]\|}, \tag{4.4b}$$

$$r_3 = \frac{-K_r e_r - K_v e_v + mge_3 + m\ddot{r}_d}{\|-K_r e_r - K_v e_v + mge_3 + m\ddot{r}_d\|}. \tag{4.4c}$$

Using the orientation reference in (4.4), 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 $[\cos\psi_d, \sin\psi_d, 0]$, 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_d altogether.

However, during the flip maneuver we intend to specify R_d from the pitch reference, therefore we do not use (4.4), but design the trajectory such that the reference attitude is consistent with the reference position, as they are coupled in the dynamical model.

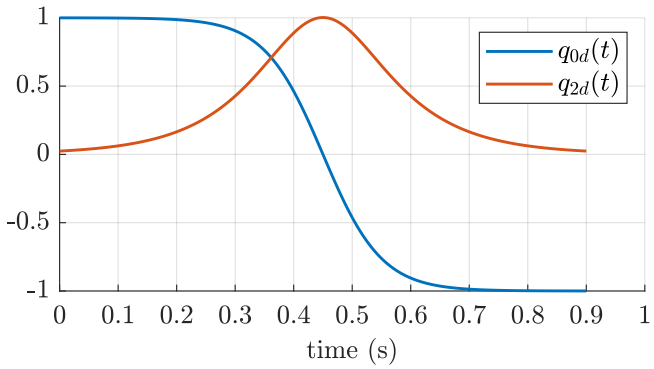


Figure 4.1: Attitude quaternion reference trajectory for the backflip maneuver with $\alpha = 20$ 1/s, $\beta = 0.45$ s.

4.2 Trajectory planning for the flip maneuver

Similarly to the open-loop control 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 [11]

$$\begin{aligned} q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} &= \begin{bmatrix} \cos(\psi/2) \\ 0 \\ 0 \\ \sin(\psi/2) \end{bmatrix} \begin{bmatrix} \cos(\theta/2) \\ 0 \\ \sin(\theta/2) \\ 0 \end{bmatrix} \begin{bmatrix} \cos(\phi/2) \\ \sin(\phi/2) \\ 0 \\ 0 \end{bmatrix} = \\ &= \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix}. \end{aligned} \tag{4.5}$$

The reference quaternion is denoted by $q_d = [q_{0d}, q_{1d}, q_{2d}, q_{3d}]^\top$, where q_{0d} is the scalar part of the quaternion, and q_{2d} corresponds to the pitch angle, as $q_{1d} = q_{3d} = 0$, because both the roll and yaw angles are zero during the flip, $\phi = \psi = 0$. Utilizing that q_d is a unit quaternion,

$$q_{2d} = \sqrt{1 - q_{0d}^2}, \tag{4.6}$$

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

A 360 degree turn around the pitch 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 $q_{0d} \in [-1, 1]$ range, because only unit quaternions describe rotation. We have chosen a smooth sigmoid function

$$q_{0d} = \frac{2}{1 + e^{-\alpha(t-\beta)}} - 1 \tag{4.7}$$

to describe the scalar part of the reference attitude, where the parameters are the speed of the maneuver α and half of the duration of the flip β. The attitude quaternion reference trajectory is displayed in Figure 4.1.

The position reference need to be design considering that the rotational and translational equations of the dynamical model are coupled. The motion of the flip maneuver is within the $x - z$ plane, therefore $y_d(t) = 0$. The other two equations of (4.1a) are

$$m\ddot{x} = -Fr'_{13}, \tag{4.8a}$$

$$m\ddot{z} = -Fr'_{33} + mg, \tag{4.8b}$$

where r'_{ij} is the element of the rotation matrix R in the ith row and jth coloumn. However, assuming that the attitude tracking converges fast enough to the reference, we can substitute the reference rotation matrix in (4.8), resulting in the translational state space representation

$$\frac{\mathrm{d}}{\mathrm{d}t}\underbrace{\begin{bmatrix} x \\ \dot{x} \\ \tilde{z} \\ \dot{\tilde{z}} \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} x \\ \dot{x} \\ \tilde{z} \\ \dot{\tilde{z}} \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ r_{13}/m \\ 0 \\ r_{33}/m \end{bmatrix}}_{\mathbf{B}} F \tag{4.9}$$

where r_{ij} are the corresponding elements of the reference rotation matrix R_d, converted from the reference quaternion, and $\mathbf{A}, \mathbf{B}$ 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 (4.9) is a linear state space representation with the thrust force F as the only control input. With discretizing the system, a quadratic programming problem can be formulated to a finite horizon, similarly to model predictive control.

The discrete state space model can be calculated as [23]

$$\mathbf{x}_{k+1} = \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d u_k, \tag{4.10}$$

$$\begin{bmatrix} \mathbf{A}_d & \mathbf{B}_d \\ \mathbf{0} & \mathbf{I} \end{bmatrix} = \exp\left(\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} T_s \right), \tag{4.11}$$

where $\mathbf{A}_d, \mathbf{B}_d$ are the discrete state space matrices, $\mathbf{0}$ and $\mathbf{I}$ denote the zero and identity matrices with suitable size, T_s is the sampling time and the $\exp(\cdot)$ operator is the matrix exponent. 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 optimization problem is formulated as

$$\min_u \sum_{k=1}^{N} \left[(\mathbf{x}_k - \mathbf{x}_{d,k})^\top \mathbf{Q}_k (\mathbf{x}_k - \mathbf{x}_{d,k}) + u_k^\top R_k u_k \right], \text{ s.t.} \tag{4.12a}$$

$$\mathbf{x}_{k+1} = \mathbf{A}_d \mathbf{x}_k + \mathbf{B}_d u_k, \tag{4.12b}$$

$$\{\mathbf{x}_k\}_{k=1}^{N} \in \mathcal{X}, \tag{4.12c}$$

$$\{u_k\}_{k=0}^{N} \in \mathcal{U}, \tag{4.12d}$$

where $\mathbf{Q}_k \in \mathbb{R}^{4\times4}$ and $R_k \in \mathbb{R}$ are weight matrices, and $\mathcal{X}$,$\mathcal{U}$ are the sets of constraints for the states and the control input, respectively. In this case, we can specify linear constraints for the states, for example $x \in [x_-, x_+], z \in [z_-, z_+]$, and also linear constraint for the control input, namely

$$\frac{\|\tau_k\|}{l} \leq u_k = F_k \leq F_{\max} - \frac{\|\tau_k\|}{l}, \tag{4.13}$$

where τ_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 τ_k is calculated from the reference attitude R_d based on (4.2b) assuming that the orientation and angular velocity errors are zero.

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 $R_k = 0, \mathbf{Q}_k = 0$ for $k = 1 \dots N$, except for the weight of the final state that is

$$\mathbf{Q}_N = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4.14}$$

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

$$\mathbf{x}_{d,N} = \begin{bmatrix} x_{d,N} \\ \dot{x}_{d,N} \\ \tilde{z}_{d,N} \\ \dot{\tilde{z}}_{d,N} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2}g(T_s N)^2 \\ 0 \end{bmatrix}. \tag{4.15}$$

The solution of a quadratic optimization program with linear constraints is a built-in feature of most engineering programming languages, for example the `quadprog` function in Matlab.

5 Simulations

In this section, the applied simulation framework and the implemented simulations are presented for the quadrotor flip maneuver, followed by the achieved results. After some literature survey about drone simulators such as [24] and [25], we decided to choose an OpenAI Gym environment based on PyBullet [26]. 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 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.

5.1 Optimized open-loop control

Based on the control approach presented in Section 3, we implemented the open-loop flip controller which evaluates the five sections of the parametrized motion primitive, and the parameter optimizer program using the Bayesian Optimization Python module [27].

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 (3.9). The optimal parameters were calculated with Bayesian optimization, using 250 random initial function evaluations and 1000 iterations. The numerical values of these are

$$P^* = \begin{bmatrix} U_1^* & t_1^* & t_3^* & U_5^* & t_5^* \end{bmatrix}^\top = \begin{bmatrix} 17.5 & 0.12 & 0.16 & 17.95 & 0.075 \end{bmatrix}^\top, \tag{5.1}$$

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,

$$F = mU, \tag{5.2}$$

where $m = 0.028$ g is the mass of the quadcopter. Using the optimal parameter set, simulation results are displayed in Figure 5.1. On the left plot, the position of the quadcopter during the flip is shown, with snapshots from the simulation. The end of the open-loop flip 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 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 with the five sections defined in Section 3. The backflip starts at $t = 0.1$ s, and ends at $t = 0.95$ s, the duration is 0.85 s.

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

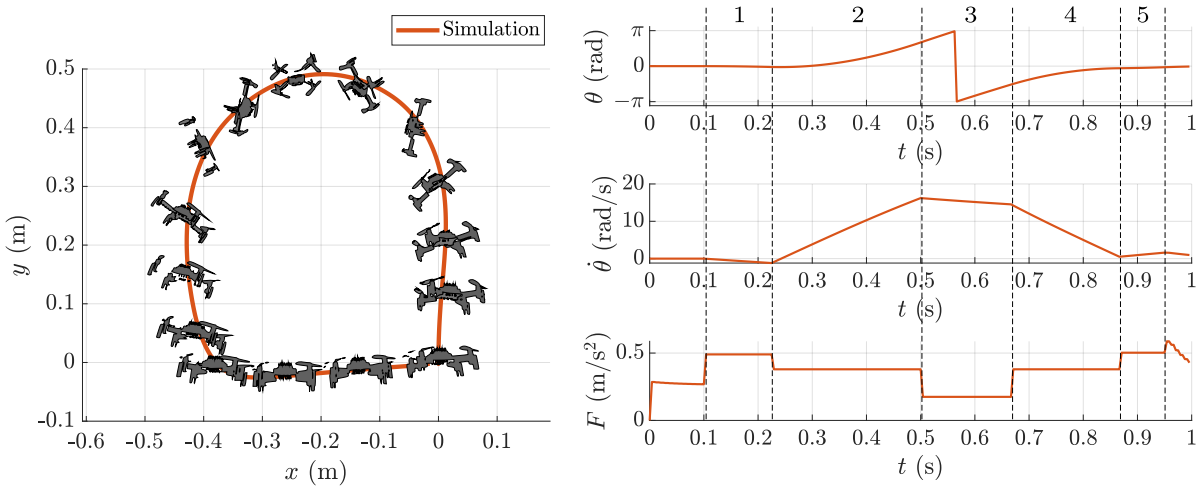


Figure 5.1: Open-loop backflip simulation results: the position is displayed on the left, and the pitch angle θ, pitch angular velocity $\dot{\theta}$, and collective thrust F on the right.

5.2 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 4.1. In the quadratic programming problem (4.12) we use the following parameters:

$$\begin{bmatrix} x_- \\ x_+ \\ z_- \\ z_+ \end{bmatrix} = \begin{bmatrix} -0.6 \\ 0.1 \\ -0.2 \\ 0.5 \end{bmatrix} \text{ m}; \quad F_{\max} = 0.64 \text{ N}; \quad T_s = 1/480 \text{ s}.$$

The maximal collective thrust $F_{\max}$ is from [14], and the position bounds are chosen such that the trajectory is feasible, and the quadcopter does not get too far from the initial point. 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 with negligible computation time. The result is the optimal control input and reference position sequence given in the discrete time instants.

The official Crazyflie 2.1 firmware has built-in trajectory evaluation algorithm for 7th degree polynomials[1], therefore we fit such functions to the optimal position sequence, and use it as the reference for the flip maneuver. As both the x and z solution are smooth, the 7th degree polynomials can be fitted with almost zero error using the Curve Fitting Toolbox of Matlab.

The quadcopter follows the designed reference trajectory with geometric tracking control, based on the control law (4.2) and error terms (4.3). The diagonal gain matrices were

[1] `https://www.bitcraze.io/documentation/repository/crazyflie-firmware/master/functional-areas/trajectory_formats/`

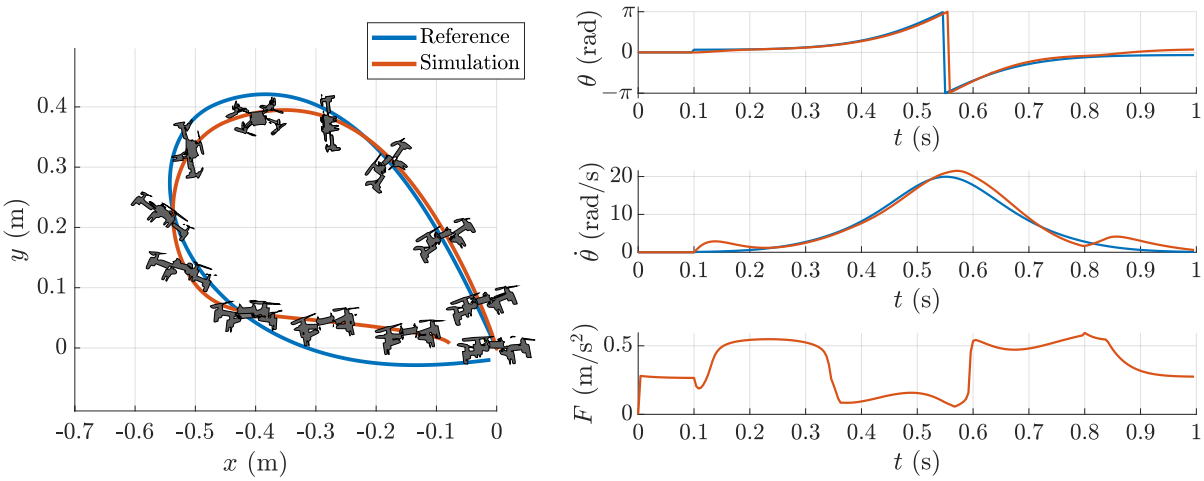


Figure 5.2: Backflipping simulation results with geometric control: position and attitude on the left, the pitch angle θ, pitch angular velocity $\dot{\theta}$, and collective thrust F on the right.

determined based on grid search and considering that the Mellinger controller [9] with a similar control law is part of the official Crazyflie 2.1 firmware[1]. The numerical values are

$$K_r = \mathrm{diag}\left(\begin{bmatrix}0.5\\0.5\\1.25\end{bmatrix}\right); \quad K_v = \mathrm{diag}\left(\begin{bmatrix}0.2\\0.2\\0.8\end{bmatrix}\right); \quad K_R = 0.08 I_{33}; \quad K_\omega = 0.002 I_{33},$$

where the result of $\mathrm{diag}(\cdot)$ is a 3×3 matrix with the argument vector in the diagonal, and I_{33} is the 3×3 identity matrix.

The simulation results of the trajectory planning and reference tracking with geometric control are displayed in Figure 5.2. 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 unmodeled transient 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.

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

6 Real-world implementation and experiments

In this section, 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 6.1 with the direction and content of the information flow between the components.

6.1 The Bitcraze Crazyflie 2.1 drone

This subsection 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.

6.1.1 Hardware

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

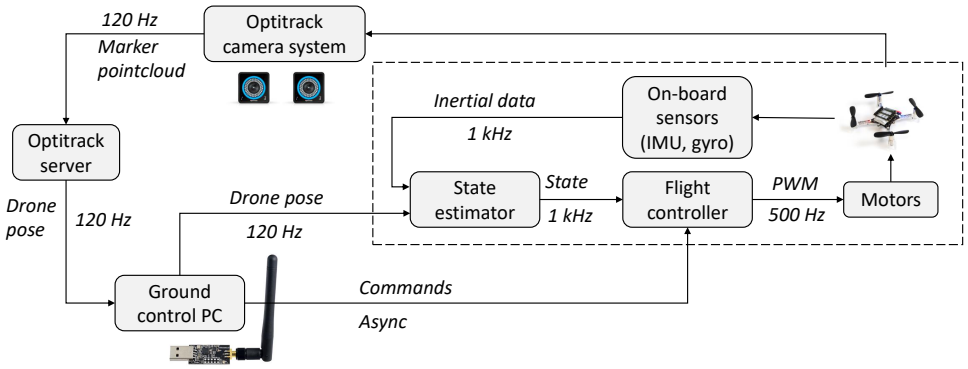


Figure 6.1: Block diagram of the experimental setup: indoor quadcopter navigation with internal and external measurement system.

Figure 6.2: The Crazyradio PA, Crazyflie 2.1 with motion capture markers, and an OptiTrack Prime 13 camera.

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 Fig. 6.2), 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.

The workstation 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 6.2.

6.1.2 Software

The open-source software continuously developed by Bitcraze[1] includes the firmware, a Python client application and Python library with examples. In addition, we use the Crazyswarm platform [28] 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 [9], and the Incremental Nonlinear Dynamic Inversion (INDI) controller [22]. In this project, we use the PID controller for setpoint stabilization before and after the open-loop flip maneuver. Our own implementation of the geometric controller is responsible for backflipping with pitch reference, setpoint stabilization, and

[1] `https://www.bitcraze.io/documentation/repository/`

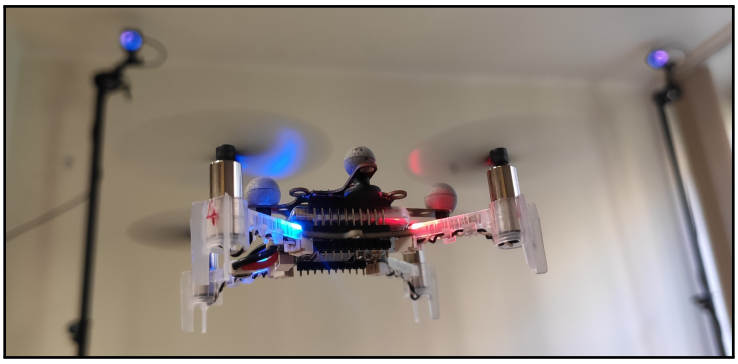


Figure 6.3: Experimental setup for the presented measurement results: 2 OptiTrack Prime 13 infrared cameras and a Crazyflie 2.1 quadcopter with reflective markers.

more complex, aggressive trajectory tracking.

6.2 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 can 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.

6.3 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 onboard 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`[1]. 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.

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 illustrated in Algorithm 2.

Algorithm 2 High-level commands for backflipping

1: Send parameters to the Crazyflie: controller type (open-loop or geometric), trajectory information, controller gains
2: Take off and hover at the initial point of the flip
3: Start the flip maneuver, and wait until it is over
4: 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 radio command, it is executed automatically, and the quadcopter ends in hovering mode afterwards.

Algorithm 3 Open-loop control on-board implementation

1: Hover with PID control, wait for ground station command to start the backflip
2: Lift with PID control to gain momentum
3: **while** the flip is not over **do**
4: Decide in which section we are (out of the 5 in Fig. 3.2)
5: Apply the control input of the current section based on (3.8)
6: **end while**
7: Stabilize the quadcopter with PID control
8: 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 duration 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.

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

Algorithm 4 Geometric tracking control on-board implementation

1: Hover with geometric control, wait for ground station command to start the backflip
2: Lift with geometric control to gain momentum
3: **while** the flip is not over **do**
4: Get the current state from the state estimator module
5: Get the setpoint by evaluating the reference trajectory
6: Calculate the control inputs from the control law (4.2)
7: Apply the control inputs to the motors
8: **end while**
9: Stabilize the quadcopter with geometric control
10: Get back to initial position

6.4 Experimental results

Experiments to perform the backflip maneuver started with the open-loop method. Firstly, we used the optimal parameter set from (5.1), 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 6.4, 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 is used to achieve exact vertical lifting and horizontal orientation.

Figure 6.4 shows that the flip is executed with almost zero final error in the pitch, and also quite small position error.

The experimental results of the backflipping with geometric control are displayed in Figure 6.5. The most important part of reference tracking is the pitch angle θ, because 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, the

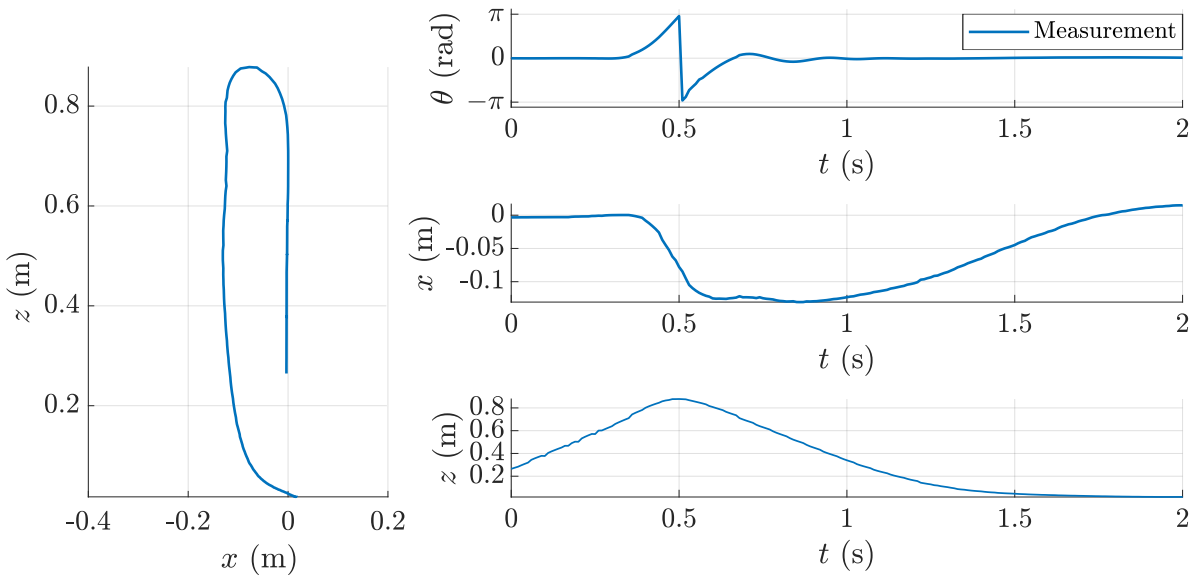


Figure 6.4: Backflipping measurement results with open-loop control.

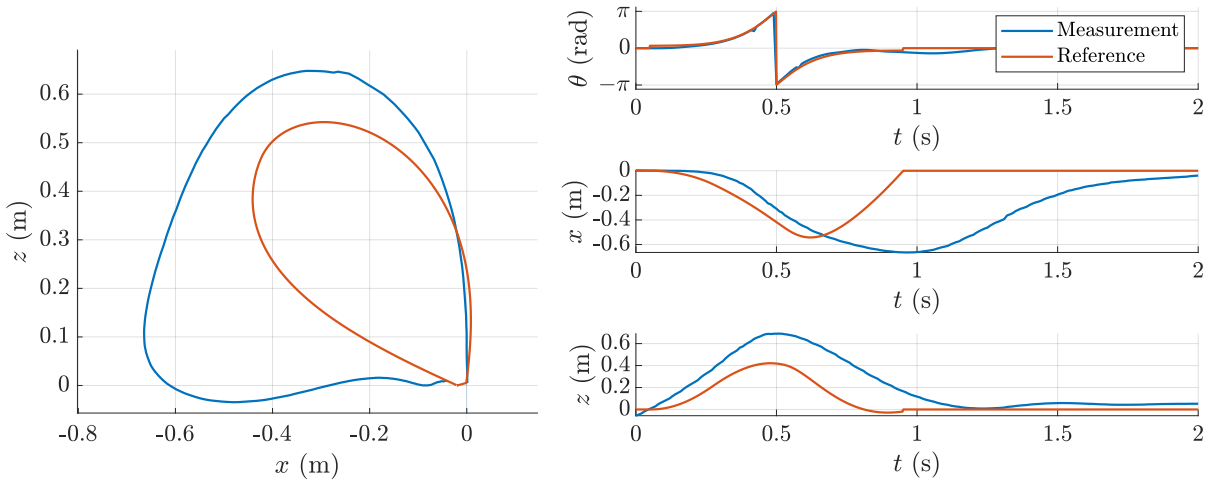
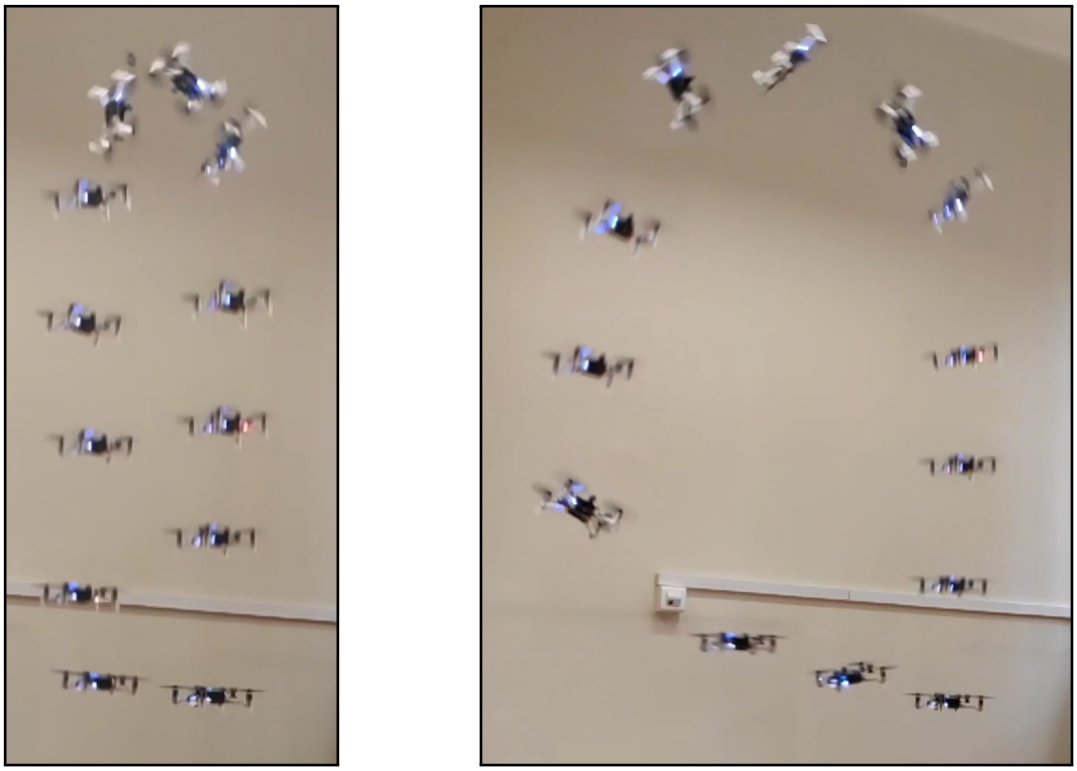


Figure 6.5: Backflipping measurement results with geometric control.

optimization problem formulation does not include an error term or constraint for this variable. 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 6.5. 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 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 [10] are very complex, therefore we exclude these terms from the dynamic model, however, they can have a negative effect on the performance of the feedback controller when performing the backflip maneuver.



(a) Optimization-based open-loop control **(b)** Geometric reference tracking control

Figure 6.6: Composite images of the measurement results with both proposed control methods. A video of the measurements is available at ***TODO***

7 Conclusions and future work

The aim of this work is to develop and implement control algorithms for performing a backflip maneuver with miniature quadcopters. A thorough literature survey has been conducted about the mathematical modelling of quadcopter dynamics, control approaches for aggressive maneuvers, and specifically for backflipping. The first proposed control method is an open-loop strategy based on the optimization of a parametric motion primitive sequence. We have interpreted the work of [6], and extended it with Bayesian optimization which we have found more suitable for parameter optimization. The second proposed strategy for backflipping is trajectory design and geometric feedback control for reference trajectory tracking. The nonlinear geometric control is part of the work in [7], providing an almost globally stable control law on $SE(3)$. We have used basic assumptions from the open-loop approach as well (e.g. duration of the maneuver) to define a reference trajectory for the attitude of the quadcopter, represented in unit quaternions. The attitude reference has been used to design the reference position trajectory, computed as the solution of a constrained quadratic optimization problem on the discrete time model of the drone. The trajectory design is computationally efficient, flexible, and most importantly general, therefore it can be used to learn other complex maneuvers in a similar framing in the future.

After the theoretical foundation of the mathematical model and control approaches, we implemented the maneuver in simulation, using the parameters of the Bitcraze Crazyflie 2.1 quadcopter. The parameter optimization of the open-loop approach proved to be successful, the drone has learnt to perform the flip with good performance. The trajectory design for the second control method has been implemented in Matlab, and the geometric control in the same Python framework as the open-loop approach. With the proper tuning of geometric control gains, simulations has shown accurate trajectory tracking both for the position and orientation of the quadcopter, performing the backflip maneuver with small errors.

The experimental setup at SZTAKI AIMotion Lab has made it possible to implement both control methods on a real Crazyflie quadcopter. The implementation of the open-loop backflipping has not been successful at first, it required manual refinement of the motion primitive parameters. Open-loop control methods are very sensitive to parameter uncertainty, therefore we need to use a slightly different parameter set for each Crazyflie 2.1 quadcopter, although their nominal parameters are the same. Moreover, even small changes on the drone dynamics (e.g. replacing a reflective marker) can lead to instability of the motion control.

The geometric control approach provides a significantly more robust method for backflipping, as feedback balances small parameter uncertainties. The results of the second strategy has shown that precise attitude tracking can be achieved with the trajectory tracking control, although there are larger position errors. We also examined the possible reasons of differences between the simulation model and reality, highlighting complex

aerodynamic effects and actuator dynamics.

The simulation and measurement results can be concluded as follows. The optimization-based open-loop approach has a simple structure, it is easy to understand and implement, and works well both in simulation and reality. However, it is designed specifically for back-flipping, therefore it can not be easily extended to other complex maneuvers. Moreover, it is very sensitive to uncertain parameters, it needs manual tuning during the implementation on a real vehicle. The second strategy is more general, it can be easily extended to other maneuvers (e.g. a barrel roll) by redefining the attitude reference trajectory and the objective of the optimization. It is also more robust to model uncertainties, the exact same parameters are used for all Crazyflies opposed to the open-loop approach, and there is no need for manual tuning of the control gains.

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 sufficient to iteratively learn the desired motion. One of the most popular research topics in machine learning control is reinforcement learning (RL). RL is widely applied for learning and optimization of quadcopter maneuvers by trial and error, often combined with deep neural network policies. In our oncoming research work, we intend to use machine learning methods to learn complex maneuvers with less expert knowledge and extend the capabilities of the miniature quadcopters even more.

List of Figures

References

[1] A. El-Badawy and M. Bakr, "Quadcopter aggressive maneuvers along singular configurations: An energy-quaternion based approach," *Journal of Control Science and Engineering*, vol. 2016, 01 2016.

[2] P. Abbeel, A. Coates, and A. Y. Ng, "Autonomous helicopter aerobatics through apprenticeship learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, 2010. [Online]. Available: https://doi.org/10.1177/0278364910371999

[3] Y. Song, M. Steinweg, E. Kaufmann, and D. Scaramuzza, "Autonomous drone racing with deep reinforcement learning," 2021.

[4] E. Kaufmann, A. Loquercio, R. Ranftl, M. Müller, V. Koltun, and D. Scaramuzza, "Deep drone acrobatics," 2020.

[5] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, p. 2096–2103, Oct 2017. [Online]. Available: http://dx.doi.org/10.1109/LRA.2017.2720851

[6] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 1642–1648.

[7] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se(3)," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 5420–5425.

[8] M. Turpin, N. Michael, and V. R. Kumar, "Trajectory design and control for aggressive formation flight with quadrotors," *Autonomous Robots*, vol. 33, pp. 143–156, 2012.

[9] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 2520–2525.

[10] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[11] E. Fresk and G. Nikolakopoulos, "Full quaternion based attitude control for a quadrotor," in *2013 European Control Conference (ECC)*, 2013, pp. 3864–3869.

[12] A. Isidori, *Nonlinear Control Systems*, ser. Communications and Control Engineering. Springer London, 1995. [Online]. Available: https://books.google.hu/books?id=fPGzHK_pto4C

[13] T. Lee, "Computational geometric mechanics and control of rigid bodies." 01 2008.

[14] J. Förster, "System identification of the crazyflie 2.0 nano quadrocopter," 2015.

[15] Y. Chen and N. O. Pérez-Arancibia, "Lyapunov-based controller synthesis and stability analysis for the execution of high-speed multi-flip quadrotor maneuvers," in *2017 American Control Conference (ACC)*, 2017, pp. 3599–3606.

[16] T. Kalmár-Nagy, R. D'Andrea, and P. Ganguly, "Near-optimal dynamic trajectory generation and control of an omnidirectional vehicle," *Robotics and Autonomous Systems*, vol. 46, no. 1, pp. 47–64, 2004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0921889003001684

[17] D. Csuzdi and P. Antal, "Inverz inga egyensúlyozása tintasugaras nyomtató mehanizmusával," *BME Tudományos Diákköri Konferencia*, 2019.

[18] P. Frazier, "A tutorial on bayesian optimization," *ArXiv*, vol. abs/1807.02811, 2018.

[19] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2016.

[20] E. Brochu, V. M. Cora, and N. de Freitas, "A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning," 2010.

[21] D. Gángó, T. Péni, and R. Tóth, "Learning based approximate model predictive control for nonlinear systems," *IFAC-PapersOnLine*, vol. 52, pp. 152–157, 01 2019.

[22] E. Smeur, Q. Chu, and G. Croon, "Adaptive incremental nonlinear dynamic inversion for attitude control of micro air vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 39, pp. 1–12, 12 2015.

[23] R. A. DeCarlo, *Linear Systems: A State Variable Approach with Numerical Implementation*. USA: Prentice-Hall, Inc., 1989.

[24] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[25] Y. Song, S. Naji, E. Kaufmann, A. Loquercio, and D. Scaramuzza, "Flightmare: A flexible quadrotor simulator," 2020.

[26] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to fly – a gym environment with pybullet physics for reinforcement learning of multi-agent quadcopter control," 2021.

[27] F. Nogueira, "Bayesian Optimization: Open source constrained global optimization tool for Python," 2014–. [Online]. Available: https://github.com/fmfn/BayesianOptimization

[28] J. A. Preiss, W. Hönig, G. S. Sukhatme, and N. Ayanian, "Crazyswarm: A large nano-quadcopter swarm," in *IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3299–3304, software available at https://github.com/USC-ACTLab/crazyswarm. [Online]. Available: https://doi.org/10.1109/ICRA.2017.7989376