

AMERICAN UNIVERSITY OF BEIRUT

A Simple Approach for Mapping Controllers
between Quadrotors for Similar Performance

by

Mohamad Jawad El-Lakkis

A thesis

submitted in partial fulfillment of the requirements
for the degree of Master of Engineering
to the Department of Electrical and Computer Engineering
of the Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
February 2020

AMERICAN UNIVERSITY OF BEIRUT

A Simple Approach for Mapping Controllers between Quadrotors for Similar Performance

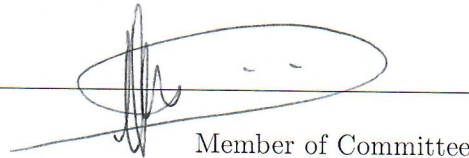
by
Mohammad Jawad El-Lakkis

Approved by:



Dr. Naseem Daher, Assistant Professor
Electrical and Computer Engineering

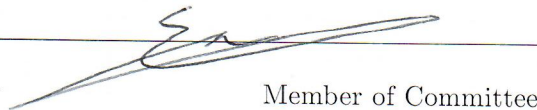
Advisor



Dr. Imad Elhajj, Professor
Electrical and Computer Engineering

Member of Committee

Dr. Elie Shammass, Associate Professor
Mechanical Engineering



Member of Committee

Date of thesis defense: February 18, 2020

AMERICAN UNIVERSITY OF BEIRUT

THESIS, DISSERTATION, PROJECT RELEASE FORM

Student Name: El-Lakkis Mohamad Jawad Bilal
Last First Middle

☒ Master's Thesis ☐ Master's Project ☐ Doctoral Dissertation

☐ I authorize the American University of Beirut to: (a) reproduce hard or electronic copies of my thesis, dissertation, or project; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes.

☒ I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of it; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes after: **One** ___ year from the date of submission of my thesis, dissertation or project.

Two ___ years from the date of submission of my thesis, dissertation or project.

Three ✓ years from the date of submission of my thesis, dissertation or project.

Ali Konso : on behalf of Mahamad Lakkis

Ali Konso
Signature

Date

This form is signed when submitting the thesis, dissertation, or project to the University Libraries

Acknowledgements

To my family, friends, and teachers.

An Abstract of the Thesis of

Mohamad Jawad El-Lakkis for Master of Engineering
Major: Electrical and Computer Engineering

Title: Trajectory Generation and Control for Quadrotors

Based on the differential flatness property of quadrotors, we propose a simple approach for transferring control policies between different quadrotors. We show that to guarantee identical performance between different quadrotors, a simple non-linear transformation between the control signals of the two quadrotors is sufficient. This transformation depends on the weight and inertia matrix of both quadrotors.

We also study the problem of efficiently generating dynamically feasible trajectories for quadrotors. A supervised learning approach is used to train a deep neural network with two hidden layers. The training data is generated from a well-known trajectory generation method that minimizes jerk given a fixed interval time. More than a million dynamically feasible trajectories between two random points in 3D space are generated and used as training data. The input of the neural network is a vector composed of initial and desired states, along with the final trajectory time. The output of the neural network generates the motion primitives of the trajectories, as well as the duration or final time of a segment. Simulation results show very fast dynamically feasible trajectory generation by the proposed deep learning algorithm.

Contents

Acknowledgements	v
Abstract	vi
1 Introduction	1
1.1 Related Work	2
1.1.1 Quadrotor Modeling	2
1.1.2 Control Algorithms	3
1.1.3 Trajectory generation	4
1.2 Scope	5
2 Modeling	7
2.1 Introduction	7
2.2 Coordinate Frame	7
2.2.1 Rigid Body Dynamics	8
2.2.2 Forces and Moments	9
2.3 System Identification	11
2.4 Conclusion	12
3 Quadrotor Control Methods	13
3.1 Introduction	13
3.2 Cascaded PID Control	14
3.2.1 Control Design	14
3.2.2 Simulation Results	15
3.3 Adaptive Control	16
3.3.1 Linearized Model	16
3.3.2 Control Design	18
3.3.3 Simulation Results	22
3.4 Conclusion	25
4 Transferring Control Policies between Quadrotors	27
4.1 Introduction	27
4.2 Transferring Control Policies	28

4.3	Simulation Results	30
4.3.1	PD Control	31
4.3.2	Adaptive Control	32
4.4	Conclusion	33
5	Stability Analysis	35
5.1	Introduction	35
5.2	Linear Controllers Stability	36
5.2.1	PD Control Stability	37
5.2.2	Full State Feedback Stability	38
5.3	Nonlinear Control Stability Analysis	39
5.3.1	System B Stability	40
6	Generating Polynomial Trajectories Using Deep Learning	42
6.1	Introduction	42
6.2	Trajectory Generation	45
6.3	Data Collection	46
6.4	Model Selection and Training	47
6.5	Results	50
6.6	Conclusion	51
7	Conclusion	54
A	Abbreviations	56

List of Figures

2.1	Global and body frames of references of quadrotor in 3D space. .	8
3.1	Position and attitude control approach as described in [1]. The position controller computes desired attitude angles along with thrust command U_1 . The attitude controller computes desired torque commands U_2	14
3.2	Simulation results of the PD controller from AR drone. The quadrotor is started at position $[0, 0, 0]$ with initial pitch angle $\theta = \frac{\pi}{2}$ to test for recovery from large attitude deviations that are far from hover conditions.	15
3.3	Simulation of trajectory following using cascaded PD control structure. Continuous lines show desired trajectories for $[x, y, z]$. Dotted lines show actual trajectories for $[x, y, z]$. At $t = 10$, one of the actuators losses 50% of it total thrust.	17
3.4	LOE simulation with different percentages for baseline (left column) and adaptive (right column) controllers. Continuous curves represent actual signals. Dotted lines represent desired signals. Red, green, and blue curves represent x, y, and z respectively. . .	24
3.5	Total thrusts commanded by the baseline and adaptive controllers with 60% LOE.	25
3.6	Power spectrum of total thrust commands in the baseline LQR (blue curve) and adaptive controllers (red curve).	26
4.1	Block diagram of control transfer method for any baseline controller.	30
4.2	Simulation results of the transferred PD controller from AR drone to Asctec Hummingbird. The secondary quadrotor is started at position $[0, 0, 0]$ with initial pitch angle $\theta = \frac{\pi}{2}$ to test for recovery from large attitude deviations that are far from hover conditions.	32
4.3	Both quadrotors traversing a circular trajectory of radius 2m at different speeds. Notice that the behavior of both quadrotor is very similar.	33

4.4	Simulation result of transferred adaptive controller for step response with 80% LOE at $t = 15s$. Continuous lines represent actual state signals. Dotted lines represent desired states. Red, green, and blue represent x , y , and z respectively. Notice that the new quadrotor B with the transferred controller is able to recover quickly.	34
6.1	Overall block diagram for a quadrotor motion control system. A trajectory generator computes the desired position. The desired position and the current state estimates of the quadrotor are used by the position controller to compute the total thrust and desired orientation. The attitude controller computes the desired torques. The clocks represent different frequencies for the trajectory generator and controller. This chapter is related to the trajectory generation block.	44
6.2	Fully connected neural network architecture used for computing motion primitives and final trajectory time. The input layer is formed of the initial and final states described by position, velocity, and acceleration. The two hidden layers have 1000 nodes each. The network has around one million trainable parameters.	48
6.3	Flow chart for real-time feasibility checking of the trajectories generated using deep learning.	50
6.4	Actual vs predicted motion primitives. The vertical line on 1.37 seconds represents the predicted final time of the trajectories. . . .	52

List of Tables

4.1	Parrot AR and Asctec Hummingbird propertiess [2]	31
6.1	The final selected training hyperparameters.	49

Chapter 1

Introduction

There has been a lot of advancement in the field of micro Unmanned Aerial Vehicles (UAV) such as quadrotors. These robots are popular due to their high mobility in 3D space. Such mobility made the quadrotor an important research platform due to its potential in various applications such as aerial photography, construction, manipulation, and precise agriculture [3], [4]. The diversity of applications led to unlimited designs of quadrotors with different inertial and actuator characteristics, which adds more challenges to autonomous quadrotor flights. Such challenges arise from the difficulty in modeling, motion planning, and control due to the dynamic environment and nonlinear model of quadrotors. A common approach to overcome these challenges is to split the problem into two main tasks: *trajectory generation* and *control* [1].

While navigating quadrotors, it is essential to generate a dynamically feasible trajectory that satisfies some task-related constraints. For example, tasks that require long-distance traveling should optimize energy consumption, and hence the trajectories generated must minimize energy required to move the quadrotor from point A to point B. Alternatively, the task could have time-critical con-

straints, and thus the energy consumed is less relevant. Therefore, the trajectory generation requires finding a proper cost function to minimize, subject to the dynamic constraints of the system at hand. In order for the quadrotor to follow these generated trajectories, a control method is needed. For simple trajectories with near hover conditions can be tracked with linear controllers [5]. However, More specifically, when the quadrotor is moving at higher speeds and large angles of attack, nonlinear dynamics become more dominant. Therefore, complex nonlinear controllers are required to stabilize quadrotors about these aggressive trajectories that deviate from the hover state by a large margin [6]. To address these challenges, we present below a brief literature review on work done in the field of quadrotor modeling, trajectory generation, and control.

1.1 Related Work

1.1.1 Quadrotor Modeling

To understand how quadrotors function, it is required to develop their mathematical model. In [3] and [7], Newton’s laws of motion were used to derive a nonlinear model. While these models treat the quadrotor as a point mass, the model proved sufficient for designing controllers that were successfully used in real-world scenarios. More complex models that account for nonlinear aerodynamics were studied in [3], [8], and [9]. These models take into account blade flapping and drag forces and moments. While these more complex models capture more complex dynamics of quadrotors, it is difficult to utilize them for control design. Hence, blade flapping and drag are often ignored when designing controllers. In [1], it is shown that quadrotors are differentially flat systems. This property helps to simplify the trajectory generation problem by allowing us to

generate trajectories in the space of differentially flat outputs instead of the full state space of the system. This will be discussed in more detail in chapter 3.

1.1.2 Control Algorithms

In the past decade, many control strategies have been proposed and tested for quadrotors. During the early stages of research on quadrotors, most control design was focused on the linearized model and operated at low speed and near hover conditions. In [5] and [10], PD (Proportional-Derivative) controllers were used to hover quadrotors and slowly move between a set of given waypoints. Similarly, the LQR (Linear Quadratic Regulator) controller was used in [11]. Like linear PD controllers, the LQR controller was also operated near hover conditions.

As shown in [9], modeling nonlinear dynamics is difficult. Moreover, even if a high fidelity model exists, it is very difficult to utilize the model in control design. To overcome these issues, adaptive control [12] can be used. For example, in [13], a model reference adaptive controller (MRAC) is designed to combat the loss of actuator effort. This also helps compensate for nonlinear dynamics that become dominant during takeoff and landing. However, as it will be shown in chapter 3, such controller performs well near hover conditions only. In particular, this controller fails to recover from large roll and pitch angles. In [14], a geometric tracking controller was derived with the capability to stabilize quadrotors from large deviations from hover conditions. A similar controller was used in [1], [15], and [6]. The controller was used for aggressive trajectories such as flips and inverted perching. In another work [16], TVLQR (time-varying LQR) is used to control quadrotors in cluttered environment. While this controller enables aggressive trajectory following, it relies on a simple PD attitude controller (similar to [1]) that runs at a faster rate than the TVLQR.

Other approaches rely on learning trajectories and control parameters. For example, in [17], expert demonstrations for RC helicopter were used to learn control policies to perform complex maneuvers such as flips, rolls, and inverted flight. In [18], controller parameters were iteratively tuned until the quadrotor was able to perform multiple flips. In a different approach, concurrent learning [19] was used to tune PID control parameters that are imported to a quadrotor with different inertial properties than those initially designed.

1.1.3 Trajectory generation

Trajectory generation was the focus of many research papers in the past decade. Depending on the application requirements, the choice of trajectory generation algorithm may vary. In a harsh environment, where large disturbances are expected, it is best to generate soft trajectories that only slightly require the quadrotor to deviate from hover conditions. Paired with an adaptive controller such as [13] that are designed for near hover conditions, these trajectories can be safely followed.

On the other hand, some applications require fast navigation between waypoints, where time-optimal trajectories generation is required. In [20], time-optimal trajectories are generated such that Pontryagin’s minimum principle with respect to time-optimality is satisfied. These trajectories are based on approximate models in a two-dimensional space (ignoring dynamics along the y-axis), whereas in realistic situations, maneuvers are executed in 3D space. Additionally, these trajectories take hours to compute. Thus, this method cannot be used in a dynamic environment where new trajectories must be generated mid-flight. To overcome this issue, real-time trajectory generation methods are required [21], [22], [23], [24]. However, to be able to achieve this, the condition for time opti-

ality has to be relaxed. For example, instead of minimizing the final trajectory time, the jerk [24] can be minimized to generate trajectories.

In [1], the differential flatness property of quadrotors is exploited to generate smooth minimum snap trajectories. In [25], an efficient method to compute piecewise minimum snap trajectories between waypoints is introduced. Nevertheless, choosing the final trajectory time has to be manually tuned.

1.2 Scope

In this work, a method for transferring control policies between different quadrotors is proposed. We show that to guarantee identical performance between different quadrotors, a simple nonlinear transformation between the control signals of the two quadrotors is sufficient. This transformation depends on the weight and inertia matrix of both quadrotors.

Moreover, the problem of efficiently generating dynamically feasible trajectories for quadrotors is also studied in chapter 6. A supervised learning approach is used to train a deep neural network with two hidden layers. The training data is generated from a well-known trajectory generation method that minimizes jerk given a fixed interval time. More than a million dynamically feasible trajectories between two random points in 3D space are generated and used as training data. The input of the neural network is a vector composed of initial and desired states, along with the final trajectory time. The output of the neural network generates the motion primitives of the trajectories, as well as the duration or final time of a segment. Simulation results show a very fast dynamically feasible trajectory generation by the proposed deep learning algorithm.

In chapter 2, the dynamical model of the quadrotor system is introduced. The

notation used in this chapter will be adopted through out this thesis. In chapter 3, two control algorithms will be described. These controllers are treated as baselines to compare with the results of the control transfer method. In chapter 4, the method to transfer controllers from one quadrotor to another is described. In chapter 4, the stability of transferred controller is studied. In chapter 6, a method for trajectory generation with deep learning is described and analysed. Our contribution in this thesis is highlighted in chapters 4, 5, and 6. Finally, a conclusion for the results of this thesis is presented in chapter 7.

Chapter 2

Modeling

2.1 Introduction

In this chapter, the coordinate frame system and the dynamical model of the quadrotor are introduced. Our notation closely follows [3] and [1]. This model will be used in the following chapters to derive control and trajectory generation methods.

2.2 Coordinate Frame

It is important to specify the frames of reference. Let $\{A\}$ denote the inertial frame, and $\{B\}$ denote the body frame. The vector $r = [x \ y \ z]^T$ denotes the position of the center of mass of $\{B\}$ in $\{A\}$ [3]. The quadrotor has six degrees of freedom: $\{x, y, z, \phi, \theta, \psi\}$. The rotation angles ϕ , θ , and ψ correspond to roll, pitch, and yaw, respectively. Therefore, define the state vector to be:

$$X^T = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix} \quad (2.1)$$

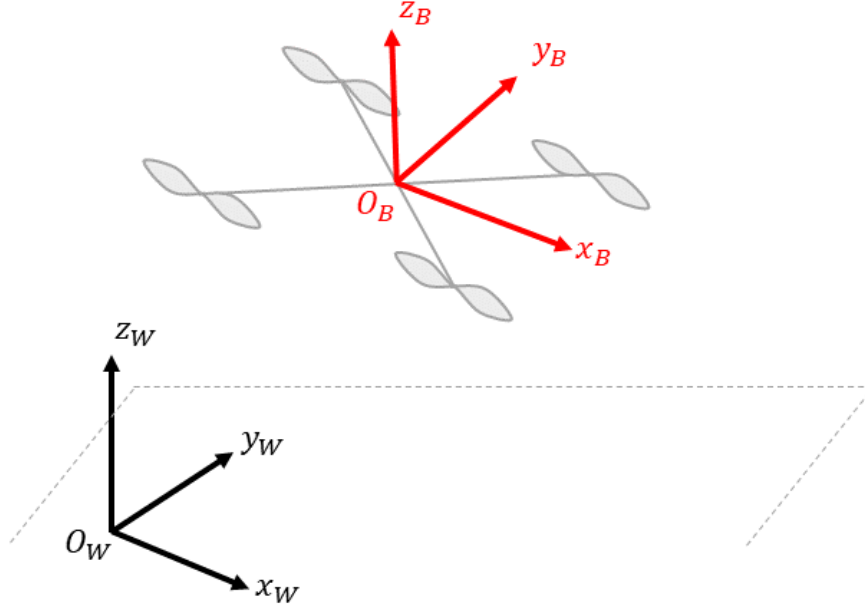


Figure 2.1: Global and body frames of references of quadrotor in 3D space.

We use $Z - X - Y$ Euler angles for the rotation matrices. Let R_B^A , or simply R , denote the rotation from $\{B\}$ to $\{A\}$ coordinates.

$$R = \begin{bmatrix} \cos\psi\cos\theta - \sin\phi\sin\psi\sin\theta & -\cos\phi\sin\psi & \cos\psi\sin\theta + \cos\theta\sin\phi\sin\psi \\ \cos\theta\sin\psi + \cos\psi\sin\phi\sin\theta & \cos\phi\cos\psi & \sin\psi\sin\theta - \cos\psi\cos\theta\sin\phi \\ -\cos\phi\sin\theta & \sin\phi & \cos\phi\cos\theta \end{bmatrix} \quad (2.2)$$

2.2.1 Rigid Body Dynamics

From Newton's second law:

$$m\ddot{\mathbf{r}} = \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} + R \begin{bmatrix} 0 \\ 0 \\ F_1 + F_2 + F_3 + F_4 \end{bmatrix} \quad (2.3)$$

Similarly, for rotational motion:

$$I \frac{d\vec{\omega}}{dt} + \vec{\omega} \times I \vec{\omega} = \vec{\tau} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \quad (2.4)$$

Where I is the inertia matrix, and $I\vec{\omega}$ is the angular momentum. Since the AR Drone platform is symmetric with equally distributed mass, the inertial matrix I becomes diagonal:

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \quad (2.5)$$

2.2.2 Forces and Moments

The thrust of a single motor is [2]:

$$F_i = c_T \omega_i^2 \quad (2.6)$$

and the drag force (force acting against the rotation of the blades) for each rotor is:

$$M_i = c_Q \omega_i^2 \quad (2.7)$$

Where c_T and c_Q are thrust and moment constants, which are properties of the blades and quadrotor frame. The total thrust is defined as:

$$T_\Sigma = \sum_{i=1}^4 c_T \omega_i^2 \quad (2.8)$$

and the moment about the x-axis is:

$$\tau_1 = d(F_2 - F_4) \quad (2.9)$$

where d is the distance from the rotors to the center of mass of the quadrotor.

The moment about the y-axis is:

$$\tau_2 = d(F_1 - F_3) \quad (2.10)$$

The moment about the z-axis is:

$$\tau_3 = M_1 - M_2 + M_3 - M_4 \quad (2.11)$$

The previous forces and moments can be summarized by the equation below:

$$\begin{bmatrix} T_\Sigma \\ \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix} = \begin{bmatrix} c_T & c_T & c_T & c_T \\ 0 & dc_T & 0 & -dc_T \\ -dc_T & 0 & dc_T & 0 \\ -c_Q & c_Q & -c_Q & c_Q \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} \quad (2.12)$$

The controller will compute required thrust and moments, and then using (2.12) the individual rotor speeds will be calculated. The processing unit will then issue a command $\bar{\omega}_{desired}$ related to $\bar{\omega}_i$ by terms of the following first-order differential equation:

$$\bar{\omega}_i = K_m (\bar{\omega}_{desired} - \bar{\omega}_i) \quad (2.13)$$

where K_m is a motor gain specific to the motors.

2.3 System Identification

From the previous section, we see that to perform the same maneuver on quadrotor B, we have to learn its inertia matrix. To do this, we go back to equation (2). Note that the $\vec{\omega}$ and $\dot{\vec{\omega}}$ can be estimated from onboard sensors, and the torque is our input. Hence, we can learn the I from data. To do so, first rewrite $\vec{\omega} \times I\vec{\omega}$ as:

$$\begin{aligned}
\vec{\omega} \times I\vec{\omega} &= \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \\
&= \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx}p + I_{xy}q + I_{xz}r \\ I_{yx}p + I_{yy}q + I_{yz}r \\ I_{zx}p + I_{zy}q + I_{zz}r \end{bmatrix} \\
&= \begin{bmatrix} -r(I_y \cdot \vec{\omega}) + q(I_z \cdot \vec{\omega}) \\ r(I_x \cdot \vec{\omega}) - p(I_z \cdot \vec{\omega}) \\ -q(I_x \cdot \vec{\omega}) + p(I_y \cdot \vec{\omega}) \end{bmatrix}
\end{aligned} \tag{2.14}$$

Where I_x , I_y and I_z represent the rows of I. Therefore, equation (2) can be written as:

$$I \frac{d\vec{\omega}}{dt} + \vec{\omega} \times I\vec{\omega} = \begin{bmatrix} I_x \cdot \vec{\omega} - r(I_y \cdot \vec{\omega}) + q(I_z \cdot \vec{\omega}) \\ I_y \cdot \vec{\omega} + r(I_x \cdot \vec{\omega}) - p(I_z \cdot \vec{\omega}) \\ I_z \cdot \vec{\omega} - q(I_x \cdot \vec{\omega}) + p(I_y \cdot \vec{\omega}) \end{bmatrix} = \begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} \tag{2.15}$$

Therefore, from the first row of the equations above, we can select a feature vector:

$$\phi(\omega, \dot{\omega}) = \begin{bmatrix} \dot{p} & \dot{q} & \dot{r} & -rp & -rq & -r^2 & qp & q^2 & qr \end{bmatrix} \tag{2.16}$$

First, we try to use stochastic gradient descent (SGD) to update our estimate of the elements of I after every measurement of the speeds and acceleration.

However, the fact that the quadrotor has unstable dynamics introduces a problem to this approach. In order to stabilize the quadrotor, our input τ is selected to be a simple feedback controller. Running SGD (or recursive least squares, RLS) on the collected data, we notice that our estimates diverge from the true value. This problem is also known as "loss of system's identifiability", and it is due to the use of feedback controllers [26] [27]. To overcome this issue, a small signal that slightly pushes the quadrotor to instability is repeatedly added to the feedback controller until we have sufficient data. Then, we can compute our estimate of I using least squares estimation.

2.4 Conclusion

In this chapter, we introduced the quadrotor dynamics and frames of references. Despite the simplicity of this model, it accounts for the most dominant dynamics for quadrotors. It has been shown in many previous projects, such as [3], [24], [20], that this model is sufficient for developing autonomous controllers for quadrotors. This model will be used for control design in chapter 3. Moreover, we will show that this model defines a differentially flat system. This is a key property that allows efficient trajectory generation for quadrotors, as it will be shown in chapter 3.

Chapter 3

Quadrotor Control Methods

3.1 Introduction

In this chapter, two popular nonlinear control strategies will be introduced. The first controller described in section 3.2.1 is capable of stabilizing quadrotor about aggressive trajectories far from hover conditions. However, this controller fails to accommodate for unmodeled disturbances. For this purpose, another adaptive controller will be introduced in 3.3. Simulation results for each controller will be shown. In chapter 4, it will be shown that knowing the inertial characteristics of two quadrotors is sufficient to transfer control policies from one to another. Nevertheless, to guarantee stability, controllers must be transferred from bigger to smaller quadrotors. This stability analysis of the transferred controller is discussed in detail in chapter 4.

3.2 Cascaded PID Control

In this section, we leverage the work in [1]. A nonlinear controller, split into a position controller and an attitude controller, is presented.

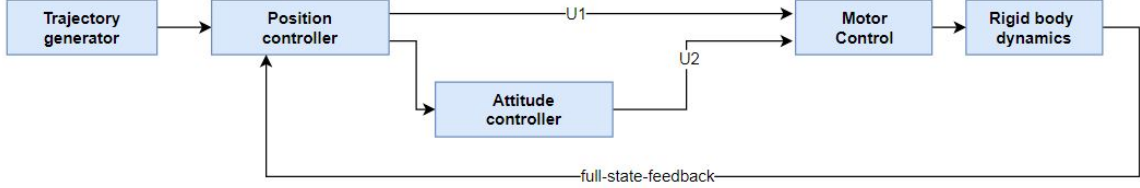


Figure 3.1: Position and attitude control approach as described in [1]. The position controller computes desired attitude angles along with thrust command U_1 . The attitude controller computes desired torque commands U_2 .

3.2.1 Control Design

To show that controllers can be easily transformed from one quadrotor to another, we assume that a well-tuned nonlinear controller is available for the reference quadrotor. The control law is described by the following equations:

$$F_{des} = -K_p e_p - K_v e_v + mgz_w + m\ddot{r}_T \quad (3.1)$$

This desired force is not always feasible, since it may not be aligned with z_b . Therefore, the commanded thrust is obtained by projecting the desired force along z_b . Hence,

$$u_1 = F_{des} \cdot z_b \quad (3.2)$$

The commanded torques are computed as follows:

$$\tau = -K_R e_R - K_w e_w \quad (3.3)$$

Where K_p , K_v , K_R , and K_w are diagonal matrices of gains. Additionally, e_p is the error in position, e_v is the error in velocity, e_R is the error in rotation, and e_w is the error in angular speeds.

3.2.2 Simulation Results

To test this controller, a random set of waypoints is selected. The quadrotor is commanded to move from one point to another at random timestamp. This is equivalent to a step response test in all direction $[x, y, z]$. Notice in 3.2 that for all axis, there is less than 15% overshoot and less than one second rise time. The initial pitch angle of the quadrotor is randomized between $\theta = [0, \frac{\pi}{2}]$. For all simulation results, it is noticed that the quadrotor stabilizes within two seconds. In 3.2, the worst case of initial $\theta = \pi/2$ is shown.

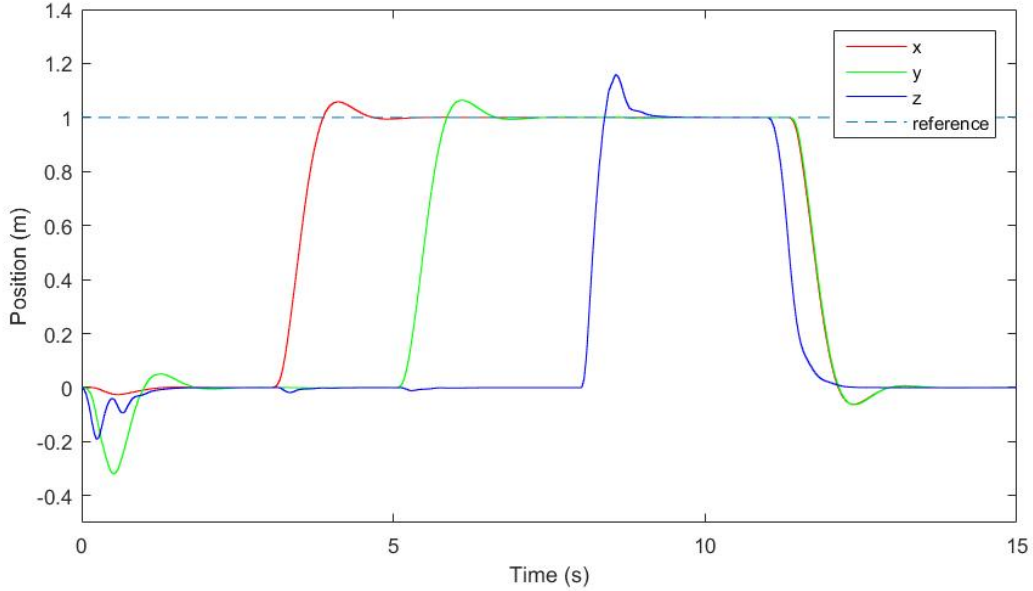


Figure 3.2: Simulation results of the PD controller from AR drone. The quadrotor is started at position $[0, 0, 0]$ with initial pitch angle $\theta = \frac{\pi}{2}$ to test for recovery from large attitude deviations that are far from hover conditions.

3.3 Adaptive Control

In the previous section, a nonlinear controller capable of stabilizing quadrotors about aggressive trajectories was introduced. Nevertheless, this controller does not account for changes in the dynamical model of the system described by the set of equations 2.3 and 2.4. For example, when one of the blades of the quadrotor is cut, the actuator would lose its thrust. This loss of thrust introduces a disturbance to the quadrotor dynamical model, which makes the controller designed in the previous section fail at stabilizing the quadrotor even for simple trajectories. To verify this, a simulation experiment is carried where the quadrotor is commanded to navigate between several waypoints. At $t = 10s$ one of the actuators is set to have 50% of its commanded thrust instead of 100%. Results shown in figure 3.3 show that after loss of effort occurs, the quadrotor will quickly deviate from its trajectory and crash.

To deal with such problems, a model reference adaptive controller (MRAC) is designed to account for disturbances on the quadrotor dynamical model [13], such as loss of effort in one of the actuators. The controller is designed based on a linearized model near hover conditions. In the following sections, the importance of such controllers is shown by comparing simulation results of a well designed LQR controller with an MRAC controller.

3.3.1 Linearized Model

The equations of motion 2.3 and 2.4 presented in the chapter 2 are nonlinear. The adaptive controller is designed for a model linearized near hover conditions. The near hover assumptions are: $\theta = \phi = \psi = 0$, $\dot{r} = 0$, $\dot{\phi} = \dot{\theta} = \dot{\psi} = 0$.

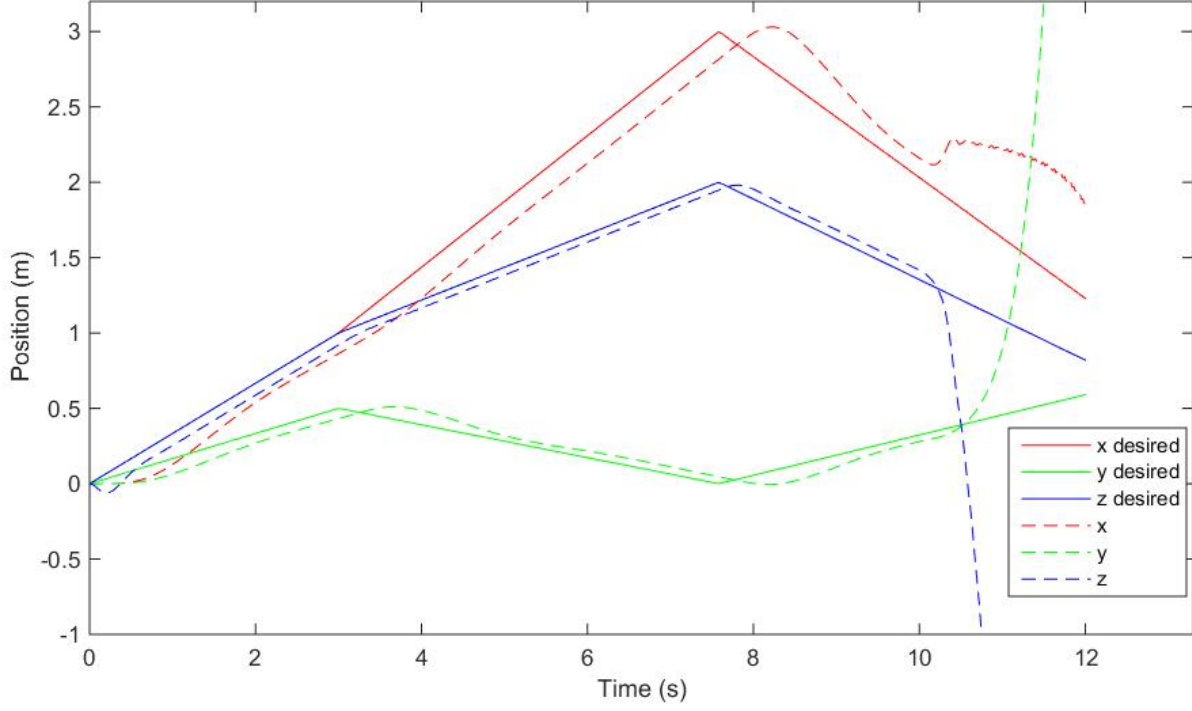


Figure 3.3: Simulation of trajectory following using cascaded PD control structure. Continuous lines show desired trajectories for $[x, y, z]$. Dotted lines show actual trajectories for $[x, y, z]$. At $t = 10$, one of the actuators losses 50% of it total thrust.

The linearized equations for motions become:

$$\begin{aligned}
 \ddot{x} &= \ddot{r}_1 = g\theta \\
 \ddot{y} &= \ddot{r}_2 = -g\phi \\
 \ddot{z} &= \ddot{r}_3 = \frac{1}{m}u_1 - g
 \end{aligned} \tag{3.4}$$

and for rotational motion:

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \tau_1/I_x \\ \tau_2/I_y \\ \tau_3/I_z \end{bmatrix} \tag{3.5}$$

3.3.2 Control Design

In this section, we derive an adaptive controller that compensates for any uncertainties in the dynamics of the quadrotor. The state-space equation of our system can be written as:

$$\dot{x} = A_p x_p + B_p \Lambda u \quad (3.6)$$

where $A_p \in \text{Re}^{12 \times 12}$, $B_p \in \text{Re}^{12 \times 4}$, $x_p \in \text{Re}^{12}$ is the state vector, $u \in \text{Re}^4$ is the input vector, and are all derived from the dynamics equations 3.4 and 3.5. Λ is an unknown 4×4 matrix that models the loss of thrust in one of the blades. The state consists then of $x, y, z, \phi, \theta, \psi$ and their derivatives. Our goal is to track the reference command $y_m \in \text{Re}^4$. So we define the output of system 3.7 as $y_p = C_p x_p$ with y_p being the vector $(x, y, z, \psi)^T$. We define the output tracking error as $e_y = y_p - y_m$, and the integrated output tracking error e_{yI} as $\dot{e}_{yI} = e_y$. We then augment these errors in the state-space model 3.7 which leads to an extended system

$$\dot{x} = Ax + B\Lambda u + B_c y_m \quad (3.7)$$

where $x = (x_p^T \ e_{yI}^T)^T$ is the extended state vector. The extended system's matrices are given by:

$$A = \begin{bmatrix} A_p & 0_{n_p \times m} \\ C_p & 0_{m \times m} \end{bmatrix}, B = \begin{bmatrix} B_p \\ 0_{m \times m} \end{bmatrix}, B_c = \begin{bmatrix} 0_{n_p \times m} \\ -I_{m \times m} \end{bmatrix} \quad (3.8)$$

We will use the model reference adaptive controller approach (MRAC) in order to adapt to any changes in the dynamics of the quadrotor (loss of thrust in one of the blades). By comparing with the system dynamics in 3.7, we choose the

reference model to be:

$$\dot{x}_m = A_m x_m + B_c r \quad (3.9)$$

where x_m is the reference model state vector, B_c is the matrix defined in 3.7, and the reference output is the vector $[x_{ref}, y_{ref}, z_{ref}, \psi_{ref}]^T$. Besides, it will be assumed that there exists a matrix K_x of the LQR baseline controller such that $A - B\Lambda K_x$ is Hurwitz i.e. its eigenvalues are negative. We choose $A_m = A - B\Lambda K_x$. The adaptive control law is defined as:

$$u_{ad} = \hat{K}^T x + \hat{\theta}_r^T r + \hat{\theta}_d^T = \hat{\theta}^T \omega \quad (3.10)$$

where $\hat{\theta}^T = [\hat{K}^T \ \hat{\theta}_r^T \ \hat{\theta}_d^T]$ is a 4×21 matrix of time-varying adaptive parameters and $\omega^T = [x^T \ r^T \ 1]$ is the regressor vector of dimension R^{21} . Notice that the first element of the adaptive controller is $\hat{K}^T x$, which has the same form as the baseline LQR controller. Thus, we propose to augment the adaptive controller to the LQR controller instead of replacing it, and this is claimed to lead to less adaptation gains and more robust performance. So the control law ends up being:

$$u = u_{ad} + u_{LQR} = \hat{\theta}^T \omega - K_x x \quad (3.11)$$

In order to choose a convenient adaptation law of the parameters matrix $\hat{\theta}$ that ensures stability and tracking of our system to the reference model, we proceed with the Lyapunov-based approach and propose the following candidate Lyapunov function that we claim will do the job:

$$V(e, \theta) = e^T P e + tr(\hat{\theta}^T \Gamma^{-1} \hat{\theta} \Lambda) \quad (3.12)$$

where $e = x - x_m$ is the model tracking error, Γ is a 21×21 diagonal matrix of adaptive gains, P is a 16×16 symmetric positive definite (SPD) matrix satisfying the Lyapunov equation, $A_m^T P + P A_m = -Q$ where Q is also a symmetric positive definite matrix. We aim toward deriving an adaptation law that will lead to $\dot{V} < 0$.

We start by deriving the error dynamics of our system. Notice that \dot{e} is the derivative of the model tracking error and is given by the following equation:

$$\begin{aligned}
\dot{e} &= \dot{x} - \dot{x}_m \\
&= (Ax + B\Lambda u + B_c r) - (A_m x_m + B_c r) \\
&= (Ax + B\Lambda(u_{ad} + u_{LQR}) + B_c r) - (A_m x_m + B_c r) \\
&\quad + A_m x - A_m x \\
&= (Ax + B\Lambda(\hat{\theta}^T \omega - K_x x) + B_c r) \\
&\quad - (A_m x_m + B_c r) + A_m x - (A - B\Lambda K_x)x \\
&= A_m e + B\Lambda(\hat{\theta}^T \omega)
\end{aligned} \tag{3.13}$$

By differentiating the Lyapunov function 3.12 with respect to time we get:

$$\begin{aligned}
\dot{V} &= \dot{e}^T P e + e^T P \dot{e} + 2tr([\hat{\theta}^T \Gamma^{-1} \dot{\hat{\theta}}] \Lambda) \\
&= (A_m e + B\Lambda(\hat{\theta}^T \omega))^T P e + e^T P (A_m e + B\Lambda(\hat{\theta}^T \omega)) \\
&\quad + 2tr([\hat{\theta}^T \Gamma^{-1} \dot{\hat{\theta}}] \Lambda) \\
&= e^T (A_m P + P A_m) e + 2e^T P B\Lambda \hat{\theta}^T \omega \\
&\quad + 2tr([\hat{\theta}^T \Gamma^{-1} \dot{\hat{\theta}}] \Lambda)
\end{aligned} \tag{3.14}$$

As we see in the previous equation, the first term clearly simplifies to $-e^T Q e$

which is convenient for us. The other terms, on the other hand, are nasty, and we shall find an adaptation law such that they cancel out. Using a small trick, we can include the second term inside the transpose term. The trick is basically a linear algebra identity which is stated as follows: given any two n dimensional vectors a and b , $a^T b = \text{tr}(ba^T)$.

Consider the second term of equation (3.14), let $a^T = e^T P B \Gamma$ and $b = \hat{\theta}^T \omega$. Therefore, by simply applying the previous identity, we can deduce that $e^T P B \Lambda \hat{\theta}^T \omega = \text{tr}(\hat{\theta}^T \omega e^T P B \Lambda)$. Thus equation (3.14) simplifies to:

$$\begin{aligned} \dot{V} &= e^T (A_m P + P A_m) e + 2e^T P B \Lambda \hat{\theta}^T \omega \\ &\quad + 2\text{tr}([\hat{\theta}^T \Gamma^{-1} \dot{\hat{\theta}}] \Lambda) \\ &= -e^T Q e + 2\text{tr}(\hat{\theta}^T [\omega e^T P B + \Gamma^{-1} \dot{\hat{\theta}}] \Lambda) \end{aligned} \tag{3.15}$$

Therefore, the following adaptation law can be derived in order to get rid of the term inside the transpose:

$$\dot{\hat{\theta}} = -\Gamma \omega e^T P B \tag{3.16}$$

and if this adaptation law was chosen, the time derivative of V becomes:

$$\dot{V} = -e^T Q e < 0 \tag{3.17}$$

which is negative definite if Q was chosen to be a positive definite matrix.

This concludes the derivation of the control law that would maintain the tracking of the quadrotor system even under changes in the dynamics of the quadrotor. In the next section, we show some simulation results for our model

that proves the robustness and practicality of such an adaptive controller.

3.3.3 Simulation Results

In this section, the main challenges in tuning both control and adaptation gains are described. Then, with properly selected gains, simulation results of the loss of effort in one of the quadrotor's propellers are shown. The simulation duration is made 25 seconds. At $t_{LOE} = 15$ seconds, a loss of thrust is introduced:

$$\omega_{LOE}^2 = (1 - loss)\omega^2 \quad (3.18)$$

In a realistic situation, the change will occur in the thrust constant c_T (see 2.6), leading to a LOE that is similar to our simulation. First, we note that the choice of the reference model has a significant effect on the adaptation rate. Recall that the reference model is chosen as:

$$Am = A - BK_x \quad (3.19)$$

where K_x is the LQR gain matrix:

$$K_x = R^{-1}B^TP \quad (3.20)$$

Also, recall that Am is used to solve the Lyapunov equation to find the matrix P . Finally, equation 3.20 shows the dependence of the adaptation gains on P . Therefore, Am has to be carefully selected to satisfy both good performance and good adaptation.

For adaptation gain tuning, states with slow dynamics are assigned smaller gains than states with fast dynamics. For example, according to the dynamic

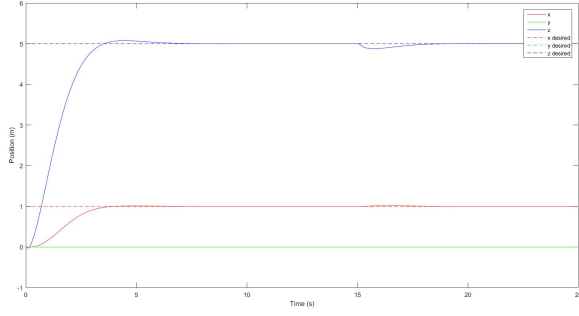
model specified above, the acceleration \ddot{z} is directly related to the control input $u_1 = u_{Thrust}$, whereas \ddot{x} and \ddot{y} are related to u_2 and u_3 with double integrals. Therefore we assign larger gains for z than x and y ¹. A series of simulations are done for 25%, 50% and 80% loss of thrust for both the baseline(LQR) and adaptive controllers. The results of our simulations are shown in the set of figures 3.4. Figures 3.4a and 3.4c show that the baseline controller is robust enough to reject 25% and 50% loss of thrust in one of the actuators. The quadrotor was able to recover its position within 4 seconds in the 50% LOE case, which is acceptable in real world applications. However, for 80%, the quadrotor loses 6 meters in altitude and overshoots by two meters while trying to recover, which is not tolerable.

On the other hand, the adaptive controller performs significantly better than the baseline controller. For 25% and 50% *LOE* (figures 3.4a, and 3.4d) the position is perturbed by few centimeters only. For 80% *LOE*, the adaptive controller was able to recover within 3 seconds and losing only around 20 cm in altitude as in figure 3.4f.

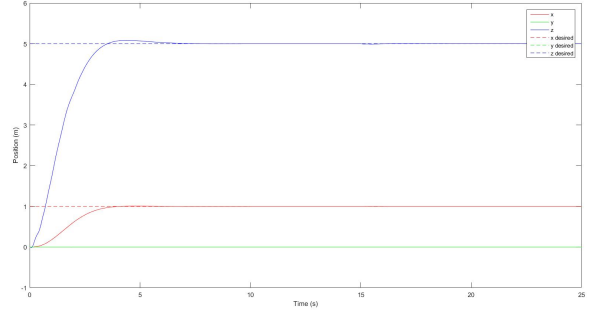
This improvement however comes at the cost of higher control thrust commands; which is predictable. Specifically, the adaptive controller has to command the higher speeds for the motors that lose its effort due to damaged rotors. Figure 3.5 shows that higher thrusts are commanded by the adaptive controller, especially after the loss occurs at $t = 15$ seconds.

Also, note that the adaptive controller commands negative thrusts during take off. This is due to the fact that actuator saturation is not included in the controller design. It is also important to note that the control commands of the

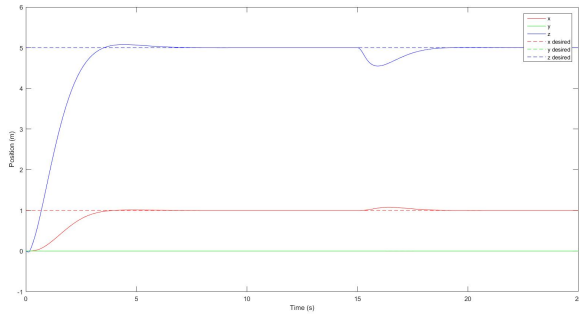
¹When hover conditions are violated, the relationship between the states and the inputs will significantly change. Thus it is important to investigate time varying adaptation gains. One possible solution is to project methods [13]. In this situation, it is sensible to project the gains along the direction of the total thrust.



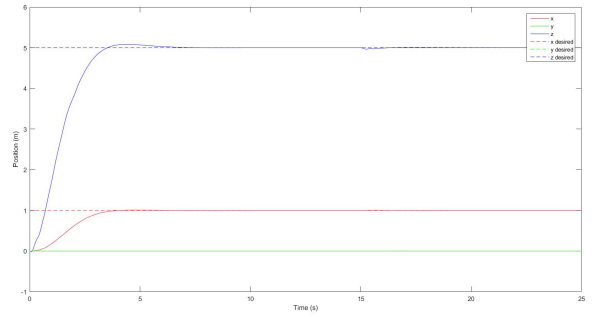
(a) LQR control with 25% LOE.



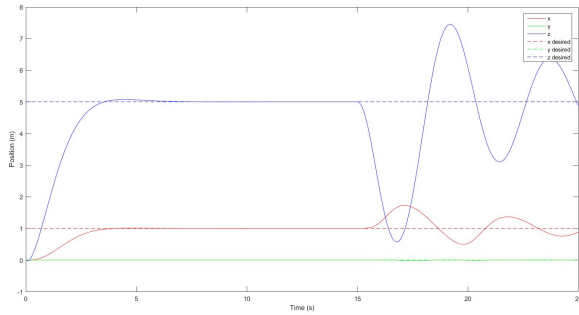
(b) Adaptive control with 25% LOE.



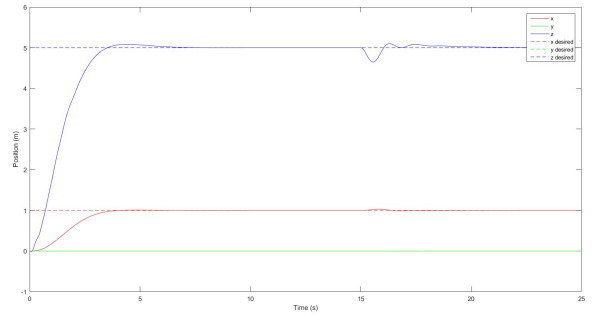
(c) LQR control with 50% LOE.



(d) Adaptive control with 50% LOE.



(e) LQR control with 80% LOE.



(f) Adaptive control with 80% LOE.

Figure 3.4: LOE simulation with different percentages for baseline (left column) and adaptive (right column) controllers. Continuous curves represent actual signals. Dotted lines represent desired signals. Red, green, and blue curves represent x , y , and z respectively.

adaptive controller change more aggressively than the baseline controller. This means that the motors will receive higher frequency commands, especially in the

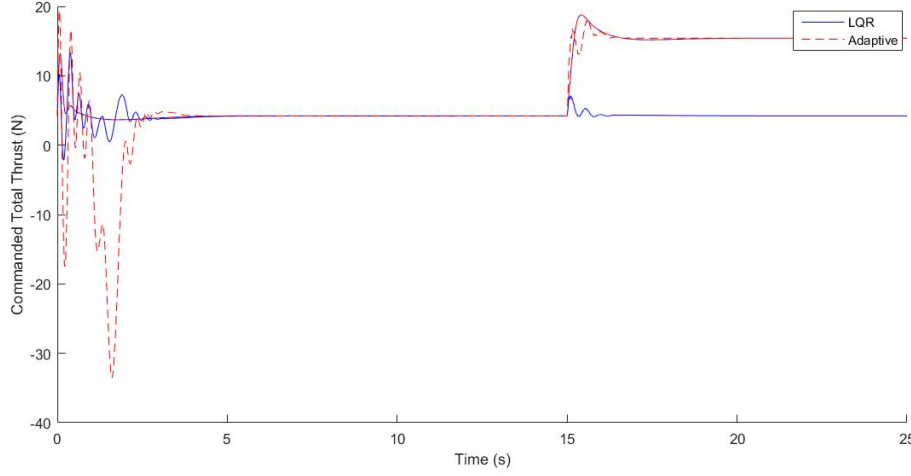


Figure 3.5: Total thrusts commanded by the baseline and adaptive controllers with 60% LOE.

presence of new disturbance. Figure 3.6 shows that higher frequency components of the adaptive controller have more power than those of the baseline controller, which is a price that we have to pay for fast adaptation.

In the last test, the reference signal is set to have a non-zero yaw angle. This means that the linearized model of the quadrotor will be inaccurate since we made the assumption that the yaw angle is zero. Results show immediate divergence from the desired trajectory (and instability of the system). Therefore, it is important to consider alternative control approaches that can compensate for inaccurate modeling.

3.4 Conclusion

In this chapter, two popular nonlinear control strategies have been introduced. The first controller described in 3.2.1 is capable of stabilizing quadrotor about aggressive trajectories far from hover conditions. However, this controller fails to

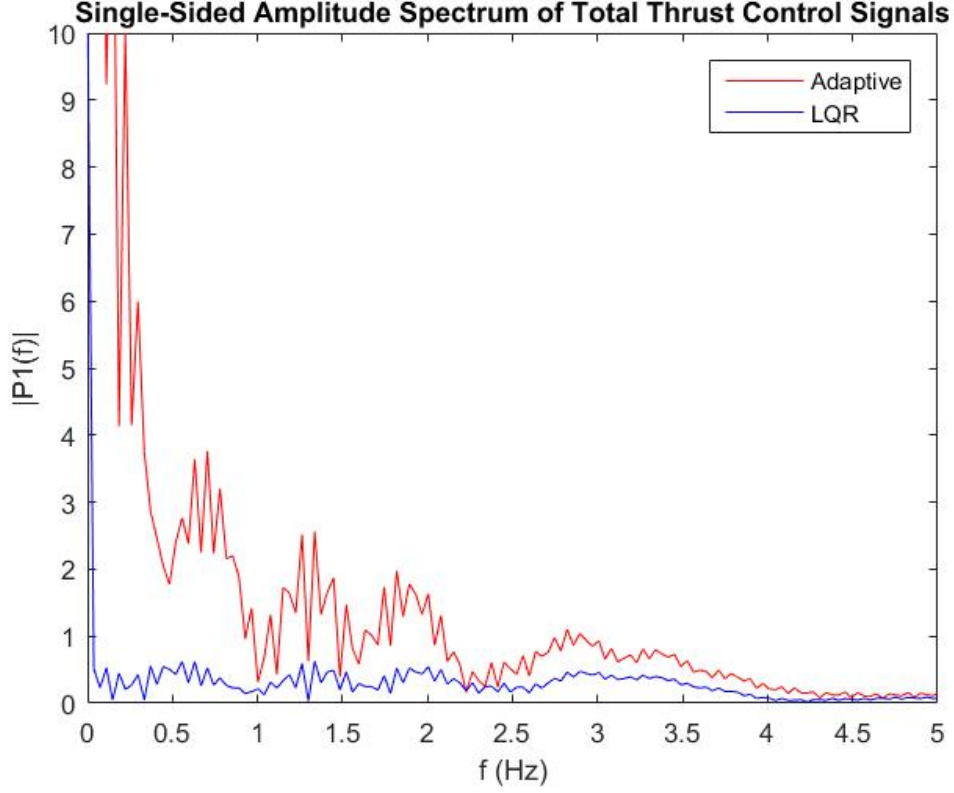


Figure 3.6: Power spectrum of total thrust commands in the baseline LQR (blue curve) and adaptive controllers (red curve).

accommodate for unmodeled disturbances. For this purpose, another adaptive controller was introduced. As it was shown in simulation results, this controller is capable of recovering from a large loss of effort in one of the actuators of the quadrotor. In chapter 4, it will be shown that knowing the inertial characteristics of two quadrotors is sufficient to transfer control policies from one to another. Nevertheless, to guarantee stability, controllers must be transferred from bigger to smaller quadrotors. This stability analysis of the transferred controller is discussed in detail in chapter 4.

Chapter 4

Transferring Control Policies between Quadrotors

4.1 Introduction

A common approach in nonlinear control is to split the problem into two main tasks. First, generate a dynamically feasible trajectory for the system, followed by the design of a faithful controller that stabilizes the system around the designed trajectory. This approach is heavily used for quadrotor control ([1], [25] [22], [27]), which is particularly effective due to the differential flatness property of quadrotors. For a differentially flat system, there is a set of flat outputs such that the states and the inputs to the system can be written as a function of those flat outputs and their derivatives (input equations from previous draft). In [1], quadrotors are proven to be differentially flat, with the flat output $\sigma = [x, y, z, \psi]$. Therefore, generating trajectories can be performed in the space of the four dimensional flat outputs instead of the twelve dimensions of the state vector $X = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \phi, \theta, \psi, \dot{\phi}, \dot{\theta}, \dot{\psi}]$. This property enabled the de-

sign of computationally efficient and dynamically feasible trajectories in a lower-dimensional space, resulting in significant improvements in quadrotor trajectory generation methods.

In this chapter, we extend the use of differential flatness property to transfer controller from one quadrotor to another. After introducing the control transfer method, we show simulation results for transferring two baseline controllers described in 3. In chapter 5, a stability analysis of the overall transferred controller is presented.

4.2 Transferring Control Policies

In this section, we show how control actions can be transferred from one quadrotor to another. Consider two quadrotors A and B with inertia matrices I^A and I^B and weights m^A and m^B . Let $\sigma_{desired}$ denote the desired trajectory describing the differential flat outputs. Assume that $\sigma_{desired}$ is dynamically feasible for quadrotors A and B. Moreover, assume that there is a well-tuned controller for quadrotor A that computes the desired thrust T and desired moments M . From 2.4, we can write:

$$\ddot{r} = gz_w + \frac{T}{m}z_B \quad (4.1)$$

where T is the total thrust commanded by the position controller, and z_B corresponds to the body frame z-axis (not to be confused as a subscript for quadrotor B). Since $r = [x, y, z]$ is a subset of σ , we have $r^A = r^B$. Hence we get:

$$\frac{T^A}{m^A} = \frac{T^B}{m^B} \quad (4.2)$$

Therefore, the total thrust signals required to stabilize both quadrotors A and B are proportional to their weights. This is intuitive since the model ignores the drag forces on the quadrotors. Hence, if we have a control law to find the desired thrust for quadrotor A, then it is possible to compute the desired thrust for quadrotor B via equation 4.2.

To get a similar relation between the moments of the quadrotors, a relationship between the angular velocities ω^A and ω^B should be derived. From [1], the desired angular velocities p and q can be written as:

$$p = -\frac{m}{T}j_2, q = \frac{m}{T}j_1 \quad (4.3)$$

Where $j = \ddot{r} = \dot{a}$ is the jerk. Combining equations 4.2 and 4.3, we obtain:

$$p^A = -\frac{m^A}{T^A}j_2 = -\frac{m^B}{T^B}j_2 = p^B \quad (4.4)$$

And,

$$q^A = \frac{m^A}{T^A}j_1 = \frac{m^B}{T^B}j_1 = q^B \quad (4.5)$$

And since $\psi \in \sigma_{desired}$, it follows that $r^A = r^B$ (r being the rotational velocity around z_{body} , and not to be confused with position vector). Therefore,

$$\omega^A = \omega^B \quad (4.6)$$

Rewriting the rotational equation of motion 4.7:

$$\dot{\omega} = I^{-1}M - I^{-1}\omega \times I\omega \quad (4.7)$$

Taking the derivative of 4.6 and substituting with 4.7, we get:

$$(I^A)^{-1}M^A - (I^A)^{-1}\omega \times I^A\omega = (I^B)^{-1}M^B - (I^B)^{-1}\omega \times I^B\omega \quad (4.8)$$

Finally, by simplifying the above equation, we obtain the map that transfers attitude control law:

$$M^B = I^B(I^A)^{-1}M^A - I^B(I^A)^{-1}\omega \times I^A\omega + \omega \times I^B\omega \quad (4.9)$$

The block diagram below shows how the method of transferring control policies works:

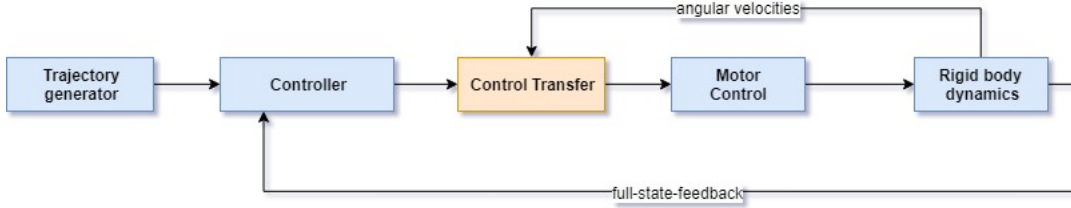


Figure 4.1: Block diagram of control transfer method for any baseline controller.

4.3 Simulation Results

In chapter 3, two control methods were introduced. In this section, we show the results of transferring controllers policies from one quadrotor to another for the two baseline controllers chosen in the previous chapter. The model of the reference quadrotor is the same as AR drone, and the model for the secondary quadrotor is the same as hummingbird. Inertial parameters for both quadrotors are summarized in 4.1.

Quadrotor Model	Parrot AR	Asctec Hummingbird
Weight (Kg)	0.429	0.18
I_x (Kg.m)	0.002237568	0.00025
I_y (Kg.m)	0.002985236	0.000232
I_z (Kg.m)	0.00480374	0.0003738
d (m)	0.1785	0.086

Table 4.1: Parrot AR and Asctec Hummingbird properties [2] .

4.3.1 PD Control

In this section, the simulation results of transferring the PD controller described in section 3.2.1 will be studied. To test our approach, the same controller that is used to generate simulation results for AR drone is used. In the first test, the secondary quadrotor is commanded to traverse a set of waypoints. The initial position of the quadrotor is at $[0, 0, 0]$. The initial pitch angle is randomized between $\theta = [0, \frac{\pi}{2}]$ to test the ability of the transferred controller to recover from large deviations from hover conditions. Results in figure 4.2 show that the secondary quadrotor transitions smoothly between waypoints. Moreover, the plot shows that the quadrotor also recovers within two seconds from the large pitch deviations.

In another test, both quadrotors are commanded to follow circular paths with different overall time t_f . For slow trajectories with $t_f = 5s$, both quadrotors have similar behavior and can smoothly follow the circular trajectories (4.2a and 4.2b). When the overall desired path time is set to $t_f = 3s$, both quadrotors have the same deviation from the nominal trajectory. Thus we conclude that the cascaded PD controller designed for quadrotor A (Parrot AR) is successfully transferred to quadrotor B (Hummingbird).

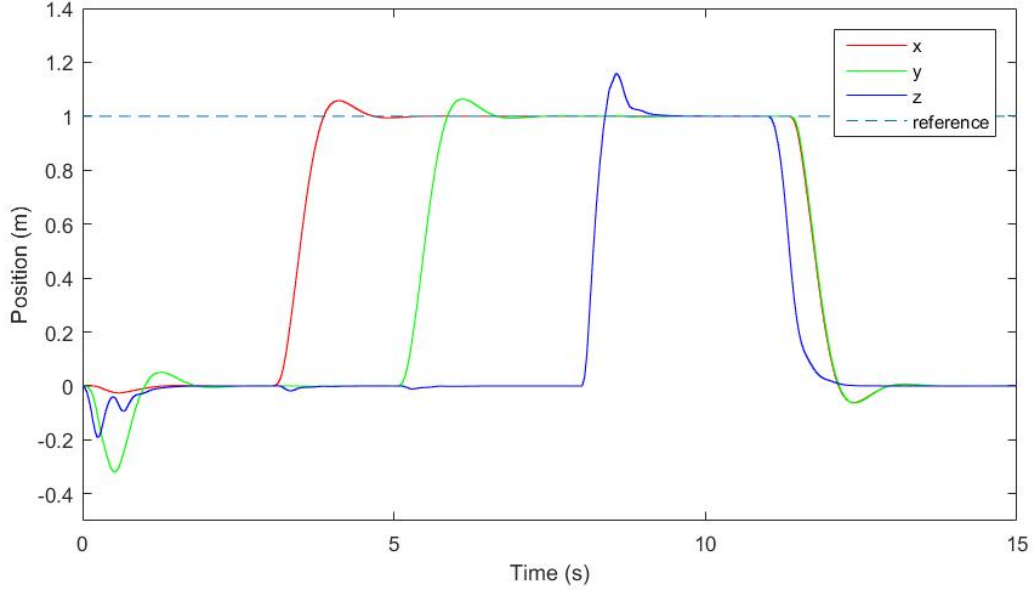
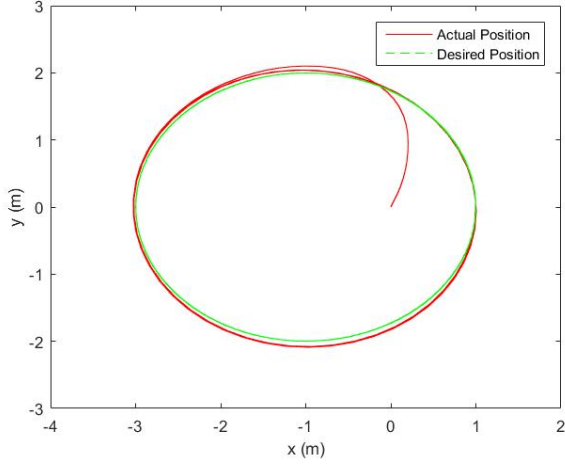


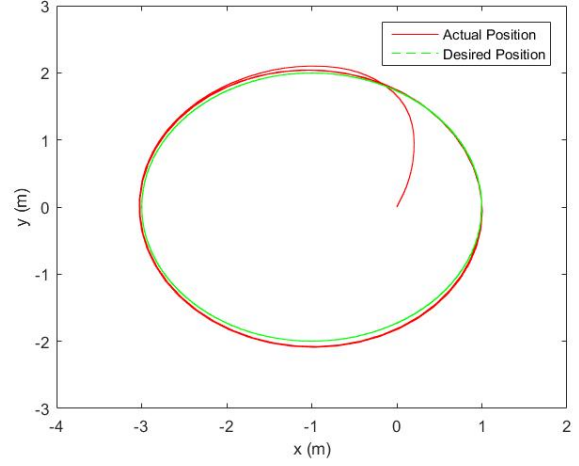
Figure 4.2: Simulation results of the transferred PD controller from AR drone to Asctec Hummingbird. The secondary quadrotor is started at position $[0, 0, 0]$ with initial pitch angle $\theta = \frac{\pi}{2}$ to test for recovery from large attitude deviations that are far from hover conditions.

4.3.2 Adaptive Control

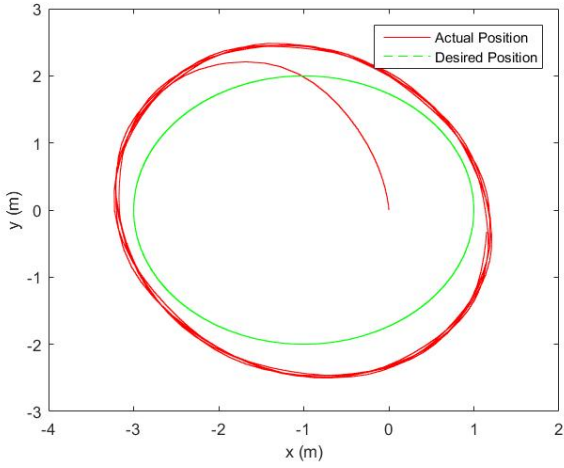
In [13], a model reference adaptive controller is designed for quadrotors. This controller was demonstrated to effectively mitigate the loss of thrust in one of the actuators of the quadrotor, as shown in chapter 3. Similar to the previous section, the control transfer method is tested with the adaptive controller. At $t_{LOE} = 15s$, a loss of actuator effort is simulated. Similar to the results obtained in the previous chapter, the results in figure 4.4 show fast recovery from back to the nominal trajectory.



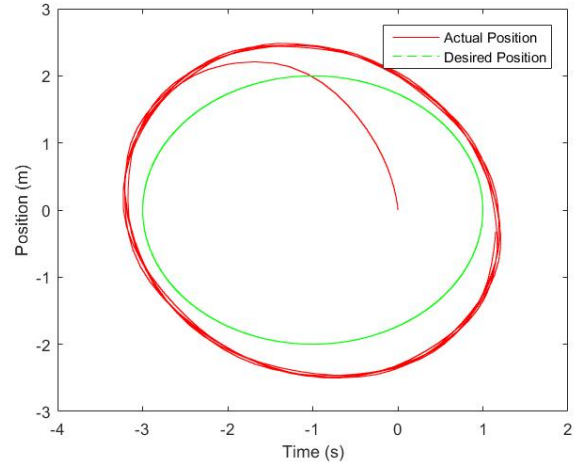
(a) Parrot AR following path with $t_f = 5s$.



(b) Hummingbird following path with $t_f = 5s$.



(c) Parrot AR following path with $t_f = 3s$.



(d) Hummingbird following path with $t_f = 3s$.

Figure 4.3: Both quadrotors traversing a circular trajectory of radius 2m at different speeds. Notice that the behavior of both quadrotor is very similar.

4.4 Conclusion

In this chapter, a method to transfer controllers between quadrotors was introduced. The control transfer method was tested in simulation to transfer two baseline controllers, as described in chapter 3. Results show very similar behavior between the two quadrotors. In the next chapter, theoretical stability analysis

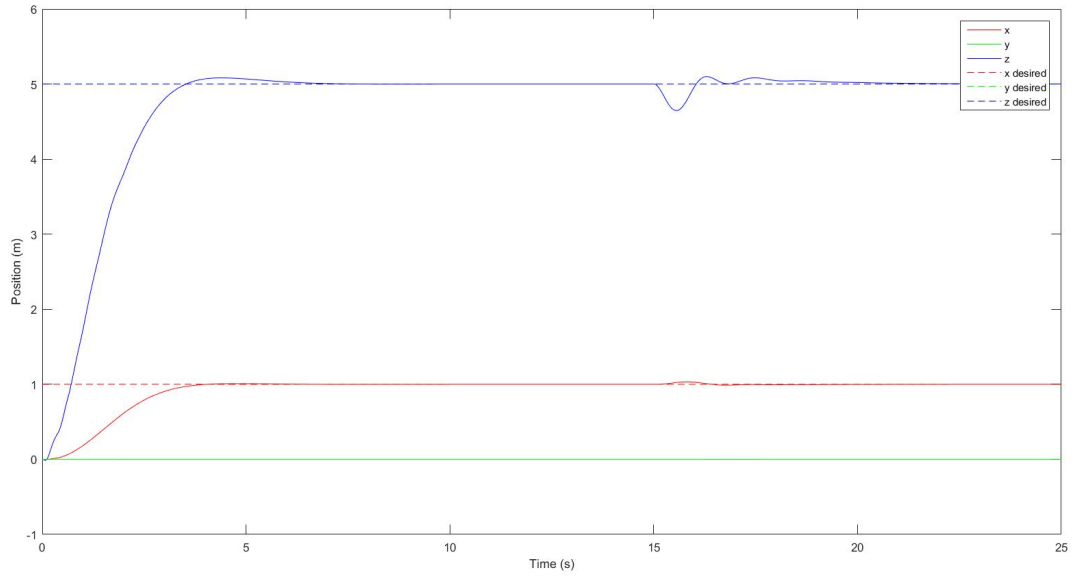


Figure 4.4: Simulation result of transferred adaptive controller for step response with 80% LOE at $t = 15s$. Continuous lines represent actual state signals. Dotted lines represent desired states. Red, green, and blue represent x , y , and z respectively. Notice that the new quadrotor B with the transferred controller is able to recover quickly.

is conducted.

Chapter 5

Stability Analysis

5.1 Introduction

Few baseline controllers were described in the previous sections. In this section, the stability of the quadrotors' attitude dynamics with transferred controllers using our derived method is analyzed. Similar to chapter 4, it is assumed that there is a quadrotor A with a well-tuned controller that achieves a desirable performance. We try to utilize the controller of A to achieve similar performance on a quadrotor B . Recall the attitude control transfer law:

$$M^B = I^B (I^A)^{-1} M^A - I^B (I^A)^{-1} \omega \times I^A \omega + \omega \times I^B \omega \quad (5.1)$$

where $M = [\tau_x, \tau_y, \tau_z]^T$ is the moments vector.

In this chapter we show that in many cases quadrotor B inherits its stability from quadrotor A . In particular, we show that it is safe to map controllers from one quadrotor to another with smaller inertial characteristics.

5.2 Linear Controllers Stability

In this section, we show that systems with transferred **linear** controllers inherit their stability from the original controllers. More specifically, the transfer functions of the two systems A and B are identical.

Linear controllers are designed to operate with perfectly symmetrical quadrotors. This means a diagonal inertia matrix I . Moreover, near hover conditions are assumed to apply. Recall the following assumptions for near hover flight:

$$\begin{aligned}
 \psi &= \psi_0 \\
 \dot{r} &= 0 \\
 \dot{\phi} &= \dot{\theta} = \dot{\psi} = 0 \\
 \cos \phi &\approx \cos \theta \approx 1 \\
 \sin \phi &\approx \phi \\
 \sin \theta &\approx \theta
 \end{aligned} \tag{5.2}$$

Under these conditions, equation 5.1 becomes:

$$M^B = I^B (I^A)^{-1} M^A \tag{5.3}$$

Since I is symmetric, we can write three separate mappings for each dimension:

$$\begin{aligned}
 u_x^B &= \frac{I_x^B}{I_x^A} u_x^A, \\
 u_y^B &= \frac{I_y^B}{I_y^A} u_y^A, \\
 u_z^B &= \frac{I_z^B}{I_z^A} u_z^A
 \end{aligned} \tag{5.4}$$

5.2.1 PD Control Stability

First, consider a quadrotor with a linear PD controller. Because the quadrotor is operating near hover conditions, the roll, pitch, and yaw dynamics can be decoupled. Here we consider the stability of roll dynamics, and similar analysis can be performed for pitch and yaw. Recall the roll dynamics equation:

$$\ddot{\phi} \approx \frac{u_x}{I_x}, \quad (5.5)$$

and the control law for ϕ :

$$u_x = \tau_x = k_{p,\phi}e_\phi + k_{d,\phi}\dot{e}_\phi \quad (5.6)$$

Under near hover conditions, equation 5.6 becomes:

$$u_x = -k_{p,\phi}\phi - k_{d,\phi}\dot{\phi} \quad (5.7)$$

Substituting equation 5.7 in equation 5.5, we get:

$$\ddot{\phi} + \frac{k_{d,\phi}}{I_x}\dot{\phi} + \frac{k_{p,\phi}}{I_x}\phi = 0 \quad (5.8)$$

Alternatively, we can write equation 5.8 as:

$$\ddot{\phi} + 2\xi\omega_n\dot{\phi} + \omega_n^2\phi = 0 \quad (5.9)$$

where:

$$\begin{aligned} k_{d,\phi} &= 2I_x\xi\omega_n, \\ k_{p,\phi} &= I_x\omega_n^2 \end{aligned} \quad (5.10)$$

This non-dimensional approach to tune PD controllers was used in [28]. By selecting a proper damping ratio ξ and a proper natural frequency ω_n , the stability of the quadrotor can be guaranteed.

Now consider the stability of roll dynamics in quadrotor B. From equations 5.4 and 5.7, we get:

$$\ddot{\phi} \approx \frac{u_x^B}{I_x^B} = \frac{1}{I_x^B} \left(\frac{I_x^B}{I_x^A} \right) u_x^A = \frac{1}{I_x^B} \left(\frac{I_x^B}{I_x^A} \right) \left(-k_{p,\phi}^A \phi - k_{d,\phi}^A \dot{\phi} \right) \quad (5.11)$$

Substituting equation 5.10 in 5.11 and simplifying, we get:

$$\ddot{\phi} + 2\xi\omega_n\dot{\phi} + \omega_n^2\phi = 0 \quad (5.12)$$

which means that rolling dynamics of quadrotor B have the same natural frequency and damping ratio as quadrotor A . Hence, quadrotor B has the same stability properties as quadrotor A .

5.2.2 Full State Feedback Stability

Consider the state-space representation of the linearized dynamics of the quadrotor:

$$\dot{X} = AX + Bu \quad (5.13)$$

The values of A and B are shown in equation 5.13. This system is controllable, and the control signal u is computed via a full state feedback controller of the form:

$$u = -Kx, \quad (5.14)$$

where K is a gain matrix. Therefore the closed-loop dynamics equation can be written as:

$$\dot{X} = (A - BK)X = A_{cl}X \quad (5.15)$$

The gain matrix K can be chosen by pole placement. Alternatively, K can be selected by designing an LQR controller and solving its corresponding algebraic Riccati equation.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 & 0 \\ 0 & \frac{1}{I_x} & 0 & 0 \\ 0 & 0 & \frac{1}{I_y} & 0 \\ 0 & 0 & 0 & \frac{1}{I_z} \end{bmatrix} \quad (5.16)$$

Remarkably, the state matrix A is independent of the inertial properties of the quadrotors. Hence, under the control transfer law derived previously, it can be shown that the input matrix B is identical for both quadrotors. Therefore, both quadrotors have the same closed-loop dynamics.

5.3 Nonlinear Control Stability Analysis

In this section, the stability of the nonlinear controller described in [1] is analyzed. The controller laws are:

$$F = -K_p e_p - K_v e_v + mgz_w + \ddot{r}_T \quad (5.17)$$

$$M = -K_R e_R - K_\omega e_\omega \quad (5.18)$$

Equation 5.17 describes the position control, whereas equation 5.18 describes the attitude controller. This controller is a relaxation of the geometric tracking controller derived in [14]. In particular, the attitude controller ignores second-order terms of angular velocities. The original attitude controller has the following structure:

$$M = -K_R e_R - K_\omega e_\omega + \omega \times I \omega - I(\hat{\omega} R^T R_d \omega_d - R^T R_d \dot{\omega}_d) \quad (5.19)$$

The exponential stability and the exponential attractiveness of this system are guaranteed for almost all initial conditions [14]. In [1] demonstrates that even the relaxed controller structure (equations 5.17 and 5.18) is also stable with very large roll and pitch angles.

5.3.1 System B Stability

Substituting equation 5.19 in 5.1. We get:

$$\begin{aligned} M^B = & I^B (I^A)^{-1} \left[-K_R e_R - K_v e_v + \omega \times I^A \omega - I^A (\hat{\omega} R^T R_d \omega_d - R^T R_d \dot{\omega}_d) \right] \\ & - I^B (I^A)^{-1} \omega \times I^A \omega + \omega \times I^B \omega \end{aligned} \quad (5.20)$$

Notice that the third term in the brackets cancels the first term outside the brackets. Expanding and reorganizing terms we get:

$$M^B = I^B (I^A)^{-1} [-K_R e_R - K_v e_v] + \omega \times I^B \omega - I^B (\hat{\omega} R^T R_d \omega_d - R^T R_d \dot{\omega}_d) \quad (5.21)$$

The first term of equation 5.21 becomes:

$$I^B(I^A)^{-1}[-K_R e_R - K_v e_v] = -\underbrace{I^B(I^A)^{-1}K_R}_{K'_R} e_R - \underbrace{I^B(I^A)^{-1}K_v}_{K'_\omega} e_v \quad (5.22)$$

Replacing in equation 5.18, we get:

$$M^B = -K'_R e_R - K'_\omega e_v + \omega \times I^B \omega - I^B(\hat{\omega} R^T R_d \omega_d - R^T R_d \dot{\omega}_d) \quad (5.23)$$

This resulting attitude controller for system B has an identical structure to the one described in equation 5.19. In [14], it is shown that the control law derived for quadrotor B guarantees exponential stability for a large set of initial conditions.

Chapter 6

Generating Polynomial Trajectories Using Deep Learning

6.1 Introduction

Generating dynamically feasible trajectories in real-time is essential for maneuvering quadrotors between an initial and a desired state. Depending on the nature of the generated trajectories, this task might be computationally expensive. Thus infeasible to be executed in real-time. For example, time-optimal trajectories required around one hour to be computed in [20]. While a quadrotor might be able to follow the generated trajectories, there is no guarantee that it can recover from unexpected perturbations. A slight wind gust might hinder the quadrotor from reaching the desired state, and the controller might saturate the actuators while trying to stabilize the quadrotor about the original trajectory. Therefore, it is common practice to rely on real-time trajectories generation methods. To resolve this issue, the condition for time optimality is relaxed. Instead of trying to minimize the total duration between two desired states, an alternative can be

generating smooth trajectories that satisfy minimum jerk [22] or minimum snap objectives [1], [25].

Alternatively, data-driven approaches can be used to learn certain trajectories, and thus, be able to generate them with less computational effort. In [29], an expert driver repeats a set of maneuvers for a remote-controlled (RC) helicopter, and an apprenticeship learning algorithm is used to leverage these expert trials and learn a good control policy. This approach, however, does not generalize to maneuvers other than those executed during the data collection procedure, which is expensive and time-consuming. In [30], quadrotors are controlled using neural networks with two hidden layers that are trained using reinforcement learning, with the unique feature that the neural network directly predicts the control signals. This method is computationally attractive as it only requires $7 \mu s$ to compute a control signal, and the results show impressive maneuvers that the quadrotor can perform well. However, end-to-end deep learning approaches do not provide stability guarantees. The only way to validate such an approach is through statistical methods ([31], [32], and [33]) and extensive testing in real-life scenarios, which can be prohibitively expensive and dangerous. In [34], a supervised learning method is used to generate paths between obstacles in 2D maps. While the feasibility of the results can be checked for collisions, there is no guarantee of dynamic feasibility of the trajectories that will be generated along this path.

In this paper, we address the problem of generating real-time trajectories in 3D space without obstacles. Applications for this problem vary from aerial photography, to benchmarking controllers in indoor environments. Our proposed solution builds upon the recent success of deep learning approaches. We utilize a deep neural network to approximate complex trajectory generation methods,

which are typically nonlinear in terms of the input and constraints. In other words, we try to encode the mapping between the inputs and outputs of deterministic trajectory generation algorithms. The inputs to the neural network are the initial and final desired states, and the outputs of the network are the motion primitives and the final time of the trajectory. Utilizing deep learning has a significant advantage in terms of computational efficiency, which becomes even more pronounced in the presence of a graphics processing unit (GPU) that can handle deep learning models in parallel computation. This enables the simultaneous evaluation of numerous trajectories in a single time step.

For our model of choice, simulation results show that it is possible to compute around 1000 trajectory motion primitives in less than $1\mu s$ on a standard personal computer. Another key advantage of the proposed deep learning method is that its generated trajectories are verifiable, which is not possible with other deep learning approaches, such as deep reinforcement learning. Since the neural network outputs motion primitives, the feasibility of the trajectories can be checked by the same method described in [22].

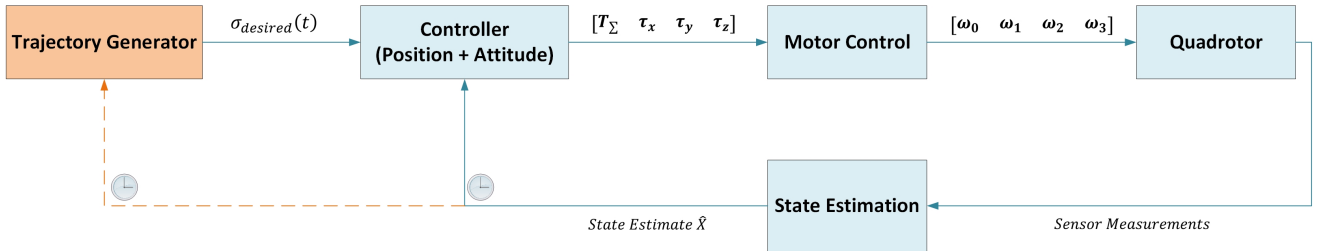


Figure 6.1: Overall block diagram for a quadrotor motion control system. A trajectory generator computes the desired position. The desired position and the current state estimates of the quadrotor are used by the position controller to compute the total thrust and desired orientation. The attitude controller computes the desired torques. The clocks represent different frequencies for the trajectory generator and controller. This chapter is related to the trajectory generation block.

6.2 Trajectory Generation

In this paper, we leverage the work of [24], which describes a method to generate minimum-jerk trajectories between initial and final states for a fixed time interval, T . Formally, we want to generate trajectories such that the L2-norm of the jerk is minimized:

$$\min \frac{1}{T} \int_0^T \|j(t)\|^2 dt \quad (6.1)$$

where the jerk j is the derivative of the acceleration a . This minimization is subject to the constraints:

$$\begin{aligned} 0 &\leq f_{\min} < F < f_{\max}, \\ |\omega| &\leq \omega_{\max}. \end{aligned} \quad (6.2)$$

Moreover, we would like to find the minimum final trajectory time, T , such that we are able to find a solution to the optimization problem in (6.1) without violating the specified constraints. The proposed solution finds decoupled quadratic polynomial trajectories for each axis as:

$$j_i(t) = \frac{1}{2}a_i t^2 + b_i t + c_i, \quad (6.3)$$

where a_i , b_i , and c_i are the motion primitives of each axis. After computing these primitives, we find the acceleration, velocity, and position by integration to obtain:

$$\begin{aligned} a_i(t) &= \frac{1}{6}a_i t^3 + b_i t^2 + c_i t + a_0, \\ v_i(t) &= \frac{1}{24}a_i t^4 + \frac{1}{6}b_i t^3 + c_i t^2 + a_0 t + v_0, \\ p_i(t) &= \frac{1}{120}a_i t^5 + \frac{1}{24}b_i t^4 + \frac{1}{6}c_i t^3 + a_0 t^2 + v_0 t + p_0. \end{aligned} \quad (6.4)$$

We will next show how these motion primitives along with the final time, T , can be efficiently computed using a supervised deep learning approach.

6.3 Data Collection

In order to train a neural network to generate trajectories, training data is first sampled from the proposed method of computing motion primitives in [22]. The input of the training data is a vector composed of the initial and final state:

$$X_i = \begin{bmatrix} r_{t_0} & \dot{r}_{t_0} & \ddot{r}_{t_0} & r_T & \dot{r}_T & \ddot{r}_T \end{bmatrix}. \quad (6.5)$$

The output is a vector of motion primitives along with the optimal final time, T :

$$y_i = \begin{bmatrix} a_x & b_x & c_x & a_y & b_y & c_y & a_z & b_z & c_z & T \end{bmatrix}. \quad (6.6)$$

Note that it is challenging to sample feasible states from this high dimensional input (18 variables). To avoid selecting infeasible data points, we randomize positions of the initial and final states, along with segment final times. For these samples, the velocity and acceleration are set to $[0, 0, 0]^T$, i.e., the quadrotor is at rest. After generating a trajectory between the initial and desired positions, we randomly sample two time instants, t_A and t_B , along this trajectory such that:

$$0 \leq t_A < t_B < T. \quad (6.7)$$

A new set of motion primitives is computed for the new datapoint. To avoid biasing the neural network, we keep the number of states at rest equal to the

number of states in motion. Hence, we randomly choose with probability $p = 0.5$ to have a balanced distribution between states at rest and states in motion, to re-sample the initial and final states.

For each training example, we compute the minimum final time, T , using binary search. We choose an upper-bound $T_{max} = 15s$ and a lower-bound $T_{min} = 0.1s$. The initial guess, T_{guess} , for T is set as the midpoint of T_{max} and T_{min} . We check if the trajectory generated between the selected states is feasible for the current guess of the optimal trajectory time. If the current time is feasible, we set $T_{max} = T_{guess}$. On the other hand, if T_{guess} is not feasible, we set $T_{min} = T_{guess}$. By repeating this process several times, the true optimal time for the trajectory is bounded between T_{max} and T_{min} . The loop is terminated when:

$$T_{max} - T_{min} = \alpha, \quad (6.8)$$

where α is a tolerance value. We sample one million trajectories for training and validation. We also sample another dataset consisting of 100,000 trajectories for testing after training is finished. It is noted here that the gradient descent method described in [25] can be used as an alternative for selecting the final time.

6.4 Model Selection and Training

Before training the machine learning algorithm, we normalize and standardize the training data, which is a particularly important step for the fidelity of the output data given that the scale of the outputs varies by two orders of magnitude. Normalization and standardization help with avoiding imbalanced gradients during training, and they also remove hockey stick learning curves by eliminating

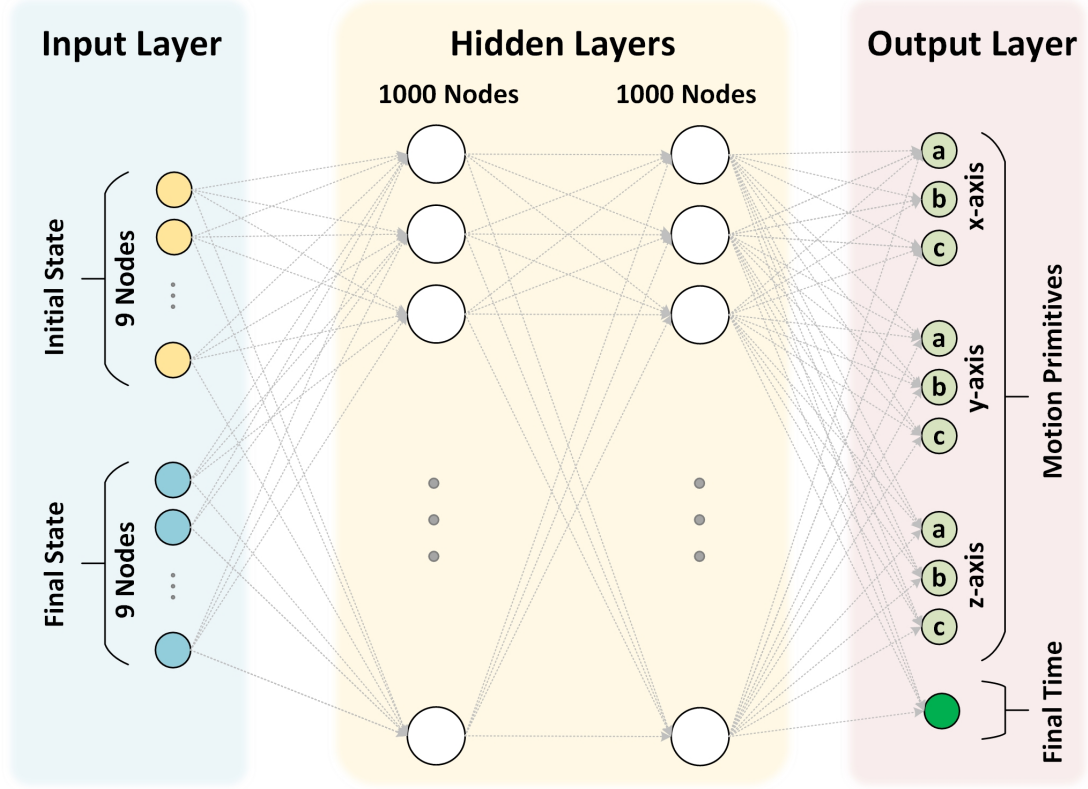


Figure 6.2: Fully connected neural network architecture used for computing motion primitives and final trajectory time. The input layer is formed of the initial and final states described by position, velocity, and acceleration. The two hidden layers have 1000 nodes each. The network has around one million trainable parameters.

the need to learn any biases in the output vector. We train a fully connected neural network with two hidden layers, each consisting of $N_{hidden} = 1000$ hidden nodes, as shown in figure 6.2. While tuning the hyperparameters, it is noticed that increasing the number of hidden layers and the number of nodes per hidden layer results in lower losses and higher accuracy. However, the final values for these parameters are chosen to strike a balance between the accuracy of predictions and the computational power required. This is important especially in the case of a quadrotor, where the computations are executed on-board. The first two layers have rectified linear units ("ReLU"s) [35] as activation functions for

the two hidden layers, and the final layer has a linear activation function. A similar neural network was able to learn complex end-to-end control policies in [36] and [30]. We choose to minimize the mean absolute error (MAE) between the trajectories as the loss function to be minimized:

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - \hat{y}_i| \quad (6.9)$$

We use Adam optimizer [37] with learning rate, $lr = 1 \times 10^{-4}$, and a decay of 2×10^{-6} . Training is carried for 250 epochs, and we perform 5-fold cross validation by randomly partitioning the data into five sets. For five training iterations, we keep one partition for validation and train on the rest of the data. Training is stopped when the training and validation losses consistently (as in equations 6.9) plateau around 0.0028. After cross validation, the neural network is trained again for 250 epochs over the entire training dataset. All hyperparameters are summarized in Table 6.1.

Training Hyperparameters	
Number of hidden layers	2
Number of nodes in hidden layers	1000
1 st and 2 nd layer activations	Relu
Final layer activation	Linear
Batch size	256
Learning rate	10^{-4}
Rate decay	10^{-6}
Number of epochs	250
Optimizer	Adam
Cost function	Mean absolute error

Table 6.1: The final selected training hyperparameters.

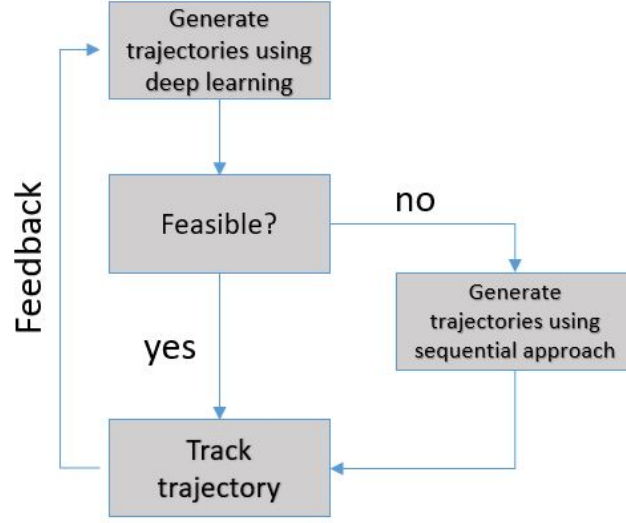


Figure 6.3: Flow chart for real-time feasibility checking of the trajectories generated using deep learning.

6.5 Results

Training the neural network converges to a mean absolute error loss of $MAE = 0.0028$. This loss shows that the error is very small between the true labels and the predicted values. To get a better understanding of the output, we visualize the results by plotting the jerk, acceleration, velocity, and position along each axis for the true and predicted values. A sample of the results is shown in Fig. 6.4. To generate this figure, an initial and final state are selected. In this case, the initial position is $[0, 0, 0]$, and the final position is $[-5, 5, 0]$. Then, motion primitives for the trajectory connecting the two states are computed using the method described in [22]. Additionally, the selected input states are fed to the trained neural network to predict motion primitives. The two sets of motion primitives are used to plot the ground truth trajectory (generated via sequential method in [22]) and predicted trajectory (generated using deep learning). Lines

with stars represent predictions, while continuous lines represent ground truths. The plotted polynomials show extremely close match to the true labels (almost identical) along the entire duration of the trajectory. In fact, the predicted final trajectory time, T , is only about 0.5 ms away from the true label. Also, note that the trajectory generated converges to the desired state at the estimated final time. For the particular example shown in figure 6.4, the final accelerations and velocities are zeros, and the final positions are $< 1mm$ away from the desired positions on each axis. For the rest of 100,000 test trajectories, the test loss is 0.0030, which is very close to training and validation loss. To estimate how long each set of motion primitives and final time requires to be computed, a set of 100,000 trajectories are sequentially tested. The average time to compute a single trajectory on an Nvidia GTX 980 GPU is around $0.4\mu s$. Moreover, each of the 1024 datapoints can be computed in one batch with an average time of $0.64\mu s$.

A final remark is that even though the predictions of the neural network are very accurate, some of the generated trajectories are slightly infeasible. To address this issue, we utilize the predicted final trajectory time, T , as the initial guess for the sequential method described in the data collection section (similar approach in [34]), and we notice that by adding $1ms$ to the final time, almost all generated trajectories ($> 99\%$) become dynamically feasible. This is summarized in figure 6.3.

6.6 Conclusion

A method for generating dynamically feasible polynomial trajectories using supervised learning was presented. After generating millions of trajectories from a well-established trajectory generation method, the data is used to train a deep

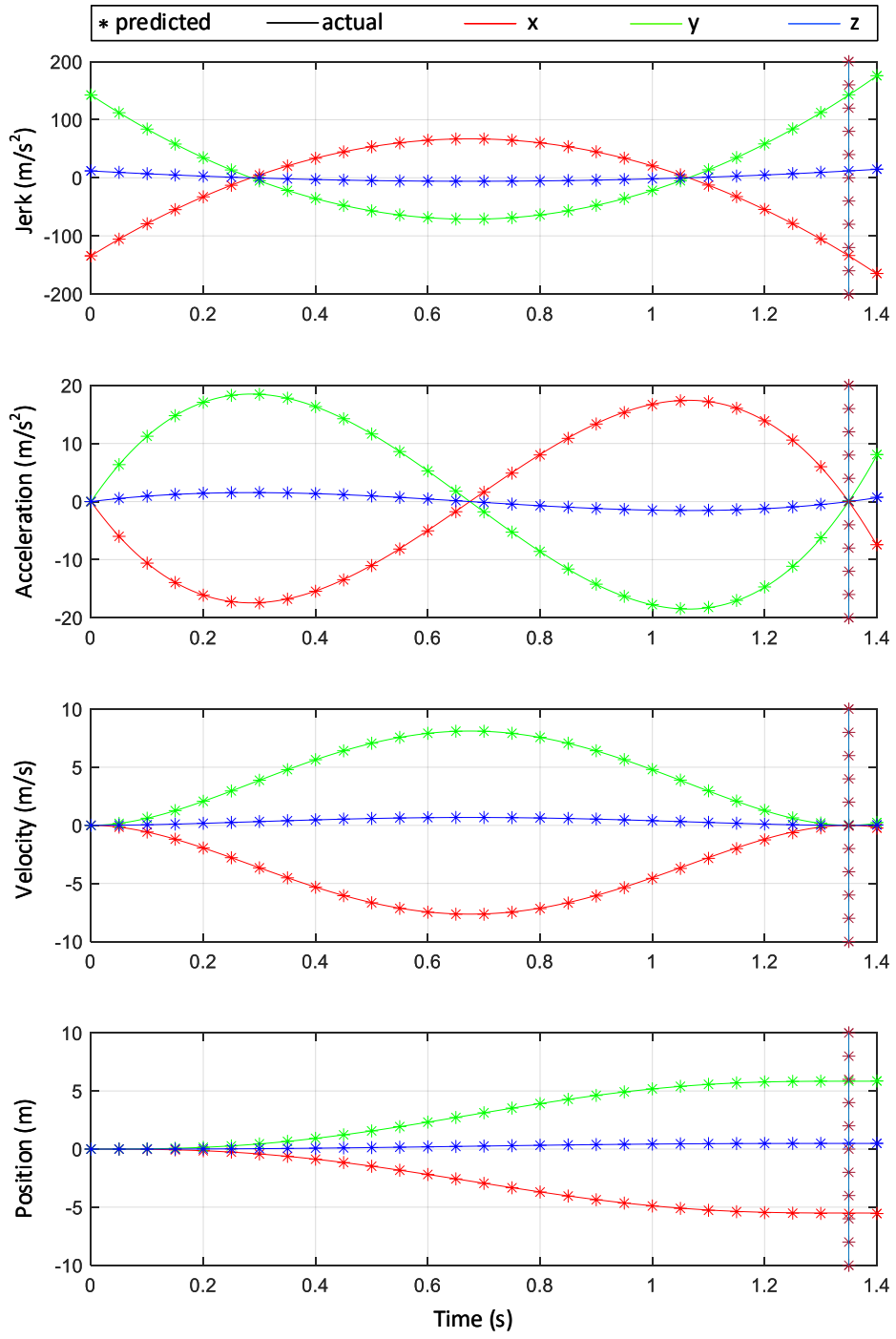


Figure 6.4: Actual vs predicted motion primitives. The vertical line on 1.37 seconds represents the predicted final time of the trajectories.

neural network to predict motion primitives along with minimum final trajectory time per segment. Results show very accurate predictions of motion primitives as well as the final time. One of the main advantages of using our method is that it enables us to check the feasibility of the trajectories being generated, since it directly computes motion primitives. This is a property that is missing from most other deep learning approaches, such as deep reinforcement learning. Since the neural network was able to learn a complex trajectory generation method, we can try training similar networks for predicting more complex trajectories. A good alternative is time optimal trajectories, which takes a long time to compute, and therefore, cannot be used for real-time applications. On the other hand, the current method suffers from its inability to incorporate obstacles and corridor constraints. Properly representing cluttered environments in the training data might allow this method to be extended to new applications, such as trajectory generation for fleets of quadrotors.

Chapter 7

Conclusion

A method for transferring control policies between different quadrotors was proposed. We showed that to guarantee identical performance between different quadrotors, a simple non-linear transformation between the control signals of the two quadrotors is sufficient. This transformation depends on the weight and inertia matrix of both quadrotors. While this method to transfer controllers between one quadrotor to another is derived from a simplified model ignores aerodynamic forces such as drag, it is possible to derive another control policy.

Moreover, the problem of efficiently generating dynamically feasible trajectories for quadrotors is also studied in chapter 6. A supervised learning approach is used to train a deep neural network with two hidden layers. The training data is generated from a well-known trajectory generation method that minimizes jerk given a fixed interval time. More than a million dynamically feasible trajectories between two random points in 3D space are generated and used as training data. The input of the neural network is a vector composed of initial and desired states, along with the final trajectory time. The output of the neural network generates the motion primitives of the trajectories, as well as the duration or final time

of a segment. Simulation results show very fast dynamically feasible trajectory generation by the proposed deep learning algorithm. This encourages us to use a deep learning approach method for generating time-optimal trajectories, which are computationally prohibitive to generate using sequential methods. Finally, both control transfer and trajectory generation methods should be tested in a realistic scenario to further validate our work.

Appendix A

Abbreviations

UAV	Unmanned Aerial Vehicles
IMU	Inertial Measurement Unit
PID	Proportional-Integral-Derivative
LQR	Linear Quadratic Regulator
TVLQR	Time Varying Linear Quadratic Regulator
UAV	Unmanned Aerial Vehicle
MRAC	Model Reference Adaptive Control
RLS	Recursive Least Squares
SGD	Stochastic Gradient Decent
ReLU	Rectified Linear Unit
LOE	Loss Of Effort
SPD	Symmetric Positive Definite

Bibliography

- [1] D. Mellinger and V. Kumar, “Minimum snap trajectory generation and control for quadrotors,” in *2011 IEEE International Conference on Robotics and Automation*, pp. 2520–2525, IEEE, 2011.
- [2] G. T. Martin, “Modelling and control of the parrot ar. drone,” *The UNSW Canberra at ADFA Journal of Undergraduate Engineering Research*, vol. 5, no. 1, 2012.
- [3] R. Mahony, C. Kumar, and P. Vijay, “Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor,” *IEEE Robotics and Automation Magazine*, vol. 19, no. 3, p. 20–32, 2012.
- [4] V. K. Sara Tang, “Autonomous flight,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, pp. 29–52, 5 2018.
- [5] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar, “The grasp multiple micro-uav testbed,” *IEEE Robotics & Automation Magazine*, vol. 17, no. 3, pp. 56–65, 2010.
- [6] D. Mellinger, N. Michael, and V. Kumar, “Trajectory generation and control for precise aggressive maneuvers with quadrotors,” *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 664–674, 2012.

- [7] F. Sabatino, “Quadrotor control: Modeling, nonlinear control design, and simulation,” Master’s thesis, KTH Royal Institute of Technology, 2015.
- [8] H. Huang, G. M. Hoffmann, S. L. Waslander, and C. J. Tomlin, “Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering,” in *2009 IEEE international conference on robotics and automation*, pp. 3277–3282, IEEE, 2009.
- [9] G. Hoffmann, H. Huang, S. Waslander, and C. Tomlin, “Quadrotor helicopter flight dynamics and control: Theory and experiment,” in *AIAA guidance, navigation and control conference and exhibit*, p. 6461, 2007.
- [10] S. Bouabdallah, “Design and control of quadrotors with application to autonomous flying,” tech. rep., Epfl, 2007.
- [11] I. D. Cowling, J. F. Whidborne, and A. K. Cooke, “Optimal trajectory planning and lqr control for a quadrotor uav,” in *International Conference on Control*, 2006.
- [12] K. J. Åström and B. Wittenmark, *Adaptive control*. Courier Corporation, 2013.
- [13] Z. T. Dydek, A. M. Annaswamy, and E. Lavretsky, “Adaptive control of quadrotor uavs: A design trade study with flight evaluations,” *IEEE Transactions on control systems technology*, vol. 21, no. 4, pp. 1400–1406, 2012.
- [14] 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)*, pp. 5420–5425, IEEE, 2010.

- [15] V. Kumar and N. Michael, “Opportunities and challenges with autonomous micro aerial vehicles,” *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1279–1291, 2012.
- [16] B. Landry, R. Deits, P. R. Florence, and R. Tedrake, “Aggressive quadrotor flight through cluttered environments using mixed integer programming,” in *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 1469–1475, IEEE, 2016.
- [17] A. Coates, P. Abbeel, and A. Y. Ng, “Learning for control from multiple demonstrations,” in *Proceedings of the 25th international conference on Machine learning*, pp. 144–151, 2008.
- [18] S. Lupashin, A. Schöllig, M. Sherback, and R. D’Andrea, “A simple learning strategy for high-speed quadcopter multi-flips,” in *2010 IEEE international conference on robotics and automation*, pp. 1642–1648, IEEE, 2010.
- [19] G. Chowdhary, T. Wu, M. Cutler, N. K. Ure, and J. How, “Experimental results of concurrent learning adaptive controllers,” in *AIAA Guidance, Navigation, and Control Conference*, p. 4551, 2012.
- [20] M. Hehn, R. Ritz, and R. D’Andrea, “Performance benchmarking of quadrotor systems using time-optimal control,” *Autonomous Robots*, vol. 33, no. 1-2, pp. 69–88, 2012.
- [21] M. Hehn and R. D’Andrea, “Real-time trajectory generation for quadcopters,” *IEEE Transactions on Robotics*, vol. 31, no. 4, pp. 877–892, 2015.
- [22] M. Hehn and R. D’Andrea, “Quadcopter trajectory generation and control,” *IFAC proceedings Volumes*, vol. 44, no. 1, pp. 1485–1491, 2011.

- [23] M. Hehn and R. D’Andrea, “Quadrocopter trajectory generation and control,” *IFAC proceedings Volumes*, vol. 44, no. 1, pp. 1485–1491, 2011.
- [24] M. W. Mueller, M. Hehn, and R. D’Andrea, “A computationally efficient motion primitive for quadrocopter trajectory generation,” *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1294–1310, 2015.
- [25] C. Richter, A. Bry, and N. Roy, “Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments,” in *Robotics Research*, pp. 649–666, Springer, 2016.
- [26] L. Ljung, *System Identification: Theory for the User*. USA: Prentice-Hall, Inc., 1986.
- [27] B. Landry, “Planning and control for quadrotor flight through cluttered environments,” Master’s thesis, Massachusetts Institute of Technology, 2015.
- [28] D. W. Mellinger, *Trajectory generation and control for quadrotors*. PhD thesis, University of Pennsylvania, 2012.
- [29] 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.
- [30] 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, pp. 2096–2103, 2017.
- [31] S. Webb, T. Rainforth, Y. W. Teh, and M. P. Kumar, “A statistical approach to assessing neural network robustness,” *arXiv preprint arXiv:1811.07209*, 2018.

- [32] M. Althoff and J. M. Dolan, “Online verification of automated road vehicles using reachability analysis,” *IEEE Transactions on Robotics*, vol. 30, no. 4, pp. 903–918, 2014.
- [33] M. O’Kelly, A. Sinha, H. Namkoong, R. Tedrake, and J. C. Duchi, “Scalable end-to-end autonomous vehicle testing via rare-event simulation,” in *Advances in Neural Information Processing Systems*, pp. 9827–9838, 2018.
- [34] T. Watanabe and E. N. Johnson, “Trajectory generation using deep neural network,” in *2018 AIAA Information Systems-AIAA Infotech @ Aerospace*, 2018.
- [35] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.
- [36] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” *arXiv preprint arXiv:1509.02971*, 2015.
- [37] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.