# An Adaptive Control Strategy for Neural Network based Optimal Quadcopter Controllers

R. Ferede

Delft University of Technology

TU Delft

# An Adaptive Control Strategy for Neural Network based Optimal Quadcopter Controllers

by

R. Ferede

| Student Name | Student Number |
| --- | --- |
| Robin | 5176603 |

Supervisors: Guido C.H.E. de Croon, Christophe de Wagter, Dario Izzo
Faculty: Faculty of Aerospace Engineering, Delft

TU Delft

# Abstract

Developing optimal controllers for aggressive high speed quadcopter flight remains a major challenge in the field of robotics. Recent work [8, 10, 15] has shown that neural networks trained with supervised learning are a good candidate for real-time optimal quadcopter control. In these methods, the networks (termed G&CNets) are trained using optimal trajectories obtained from a dynamical model of the quadcopter by means of a direct transcription method. A major problem with these methods is the effects of unmodeled dynamics. In this work we identify these effects for G&CNets trained for power optimal full state-to-rpm feedback. We propose an adaptive control strategy to mitigate the effects of unmodeled roll, pitch and yaw moments. Our method works by generating optimal trajectories with constant external moments added to the model and training a network to learn the policy that maps state and external moments to the corresponding optimal rpm command. We demonstrate the effectiveness of our method by performing power-optimal hover-to-hover flights with and without moment feedback. The flight tests show that the inclusion of this moment feedback significantly improves the controller's performance. Additionally we compare the adaptive controller's performance to a time optimal Bang-Bang controller for consecutive waypoint flight and show significantly faster lap times on a 3x4m track.

# Contents

# 1

# Introduction

Nowadays there is an increasing demand for autonomous Unmanned Aerial Vehicles (UAV's) for various military and civilian applications [4]. For many applications such as emergency response, pursuit tasks or racing, it is necessary for the drone to fly as fast as possible. However, developing autonomous systems for aggressive high speed flight remains a major challenge in the field of drone control. In order to push the boundaries of high speed autonomous flight and artificial intelligence, Lockheed Martin organised the Alpha Pilot AI Drone racing Challenge in 2019. This Challenge was won by Team MAVLab from the TU Delft who managed to complete the race track in 12 seconds – 25% faster than the second-place drone. Although this result is impressive, one of the best drone pilots in the world Grabriel "Gab707" Kocher beat the game by finishing the same track in 6 seconds which is twice as fast as Team MAVLab [1]. Decreasing this gap between human and AI flight performance is one of the research objectives of the MAVLab and it is precisely what this thesis is about.

Recent research from the TU Delft and the Advanced Concepts Team from the European Space Agency (ESA) aims to solve this real time optimal control problem by making use of deep neural networks called G&CNets that are trained on a dataset of time optimal trajectories. Our proposal will be a further extension of this G&CNet research covering the future work as described in the conclusion of [8]. In this work we test and implement a G&CNet that takes into account the full 6 degrees of freedom dynamics of the quadcopter. The dynamical model is determined by running system identification experiments on the Parrot Bebop quadcopter in the Cyberzoo at the TU Delft. The biggest obstacle with this approach is the reality gap between the model and the real world. For this reason the identification of unmodeled effects and the analysis of the controller's robustness are essential to our project. The ultimate goal of the research is to use G&CNets to take the quadcopter to its absolute physical limits in terms of aggressive flight. This would not only make a great candidate for drone racing competitions, but also provide a general solution to optimal control problems in robotics where the reality gap plays a big role.

This document is divided into 3 parts. Part 1 contains the scientific paper which present the main results of this thesis. Part 2 will cover a Literature review elaborating on current research followed by a detailed proposal specifying the content of this master thesis. The literature review also contains a preliminary experiment in which the first system identification experiments are presented. Finally in Part 3 additional results are presented that are outside of the scope of the scientific paper.

# Part I

# Scientific Paper

# An Adaptive Control Strategy for Neural Network based Optimal Quadcopter Controllers

Robin Ferede, Guido C.H.E. de Croon, Christophe de Wagter, Dario Izzo
*Delft University of Technology, 2629 HS Delft, The Netherlands*

**Developing optimal controllers for aggressive high speed quadcopter flight remains a major challenge in the field of robotics. Recent work [1–3] has shown that neural networks trained with supervised learning are a good candidate for real-time optimal quadcopter control. In these methods, the networks (termed G&CNets) are trained using optimal trajectories obtained from a dynamical model of the quadcopter by means of a direct transcription method. A major problem with these methods is the effects of unmodeled dynamics. In this work we identify these effects for G&CNets trained for power optimal full state-to-rpm feedback. We propose an adaptive control strategy to mitigate the effects of unmodeled roll, pitch and yaw moments. Our method works by generating optimal trajectories with constant external moments added to the model and training a network to learn the policy that maps state and external moments to the corresponding optimal rpm command. We demonstrate the effectiveness of our method by performing power-optimal hover-to-hover flights with and without moment feedback. The flight tests show that the inclusion of this moment feedback significantly improves the controller's performance. Additionally we compare the adaptive controller's performance to a time optimal Bang-Bang controller for consecutive waypoint flight and show significantly faster lap times on a 3x4m track.**

## I. Introduction

Nowadays there is an increasing demand for autonomous Unmanned Aerial Vehicles (UAV's) for various military and civilian applications [4]. For many applications such as emergency response, pursuit tasks or racing it is necessary for the drone to fly as fast as possible. However, developing autonomous systems for aggressive high speed flight remains a major challenge in the field of drone control.

Most research about high speed autonomous flight focuses on making controllers to follow a reference guidance trajectory [5, 6]. The type of trajectories that are generated determine the aggressiveness and efficiency of the performed manoeuvre. By choosing a parameterization of the trajectory, these methods are only optimal in a restricted sense. Additionally these method require inner loop controllers to track the reference trajectory that often do not take into account the actuator limits. In optimal control theory methods exist to obtain the true optimal trajectory that takes into account the system dynamics and actuator limits. These methods have mostly been developed in the 1950s by Lev Pontryagin and Richard Bellman [7]. Although the theory has been around for a long time, practical implementations remained limited due to the computational power required to obtain the solutions.

Recent research [1–3] from the TU Delft and the Advanced Concepts Team from the European Space Agency (ESA) aims to solve this real-time optimal control problem by making use of deep neural networks called G&CNets that are trained on state action pairs from time optimal trajectories obtained by a direct transcription method. Once trained, the G&CNet can provide the optimal control in real time on-board the quadcopter. With a real flight test this has been demonstrated to work for longitudinal trajectories based on a simplified quadcopter model [1]. In these experiments the G&CNet was used to calculate a thrust command and a pitch acceleration which where tracked by an inner-loop controller. Our proposal will be a further extension of this G&CNet research covering the future work as described in the conclu-

sion of [1]. We will investigate the feasibility of G&CNets trained using a 3 dimensional quadcopter model taking into account drag, aerodynamic effects and actuator delays. This network will then directly calculate the rpm motor commands without using an inner-loop controller. The biggest obstacle with this approach is the reality gap between the model and the real world. In this research we identify this reality gap for power optimal flight and propose an adaptive method to mitigate the effects of unmodeled roll, pitch and yaw moments. Furthermore we will be benchmarking our controller's performance against a time optimal Bang-Bang controller in a consecutive waypoint flight.

The following sections will start with a detailed description of our methodology which elaborates the quadcopter model, control problem, machine learning method and the adaptive control strategy that we use. Additionally a brief explanation of the Bang-Bang controller is included. This is followed by the experimental setup which we use for our flight test. In Results & Discussion we showcase our flight test results in which we identify the unmodeled effects, demonstrate the improved performance of our adaptive method and benchmark our controller's performance for consecutive waypoint fight. Finally in the conclusion we summarize our main contributions and elaborate on further research.

## II. Methodology

### A. Quadcopter model

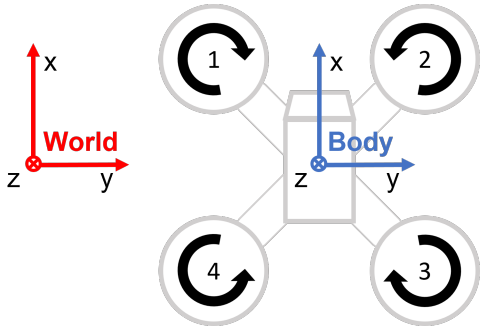



**Figure 1. Axis definition (z-axis points downwards)**

The quadcopter's state and control input are defined by

$$\mathbf{x} = [\mathbf{p}, \mathbf{v}, \lambda, \Omega, \omega]^T \quad \mathbf{u} = [u_1, u_2, u_3, u_4]^T$$

Where $\mathbf{p} = [x, y, z]$ and $\mathbf{v} = [v_x, v_y, v_z]$ are the position and velocity in the world frame, $\mathbf{\Omega} = [p, q, r]$ is the angular velocity in body frame, $\lambda = [\phi, \theta, \psi]$ are the euler angles that describe the orientation of the body frame and $\omega = [\omega_1, \omega_2, \omega_3, \omega_4]$ are the angular velocities of each of the propellers in rpm. The control input $\mathbf{u}$ contains the normalized rpm commands $u_i \in [0, 1]$.

The system dynamics is described by:

$$\begin{aligned} \dot{\mathbf{p}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \mathbf{g} + R(\lambda)\mathbf{F} \\ \dot{\lambda} &= Q(\lambda)\mathbf{\Omega} \\ I\dot{\mathbf{\Omega}} &= -\mathbf{\Omega} \times I\mathbf{\Omega} + \mathbf{M} \\ \dot{\omega} &= ((\omega_{max} - \omega_{min})\mathbf{u} + \omega_{min} - \omega_i)/\tau \end{aligned} \tag{1}$$

Where $\mathbf{g} = [0, 0, g]^T$ is the gravitational acceleration, $I$ is the moment of inertia matrix given by $\text{diag}(I_x, I_y, I_z)$, $\omega_{min}$ and $\omega_{max}$ are the minimum and maximum propeller rpm limits and $\tau$ is the first order delay parameter of the actuator model. Furthermore $R(\lambda)$ is the rotation matrix defined by

$$R(\lambda) = \begin{bmatrix} c_\theta c_\psi & -c_\phi s_\psi + s_\phi s_\theta c_\psi & s_\phi s_\psi + c_\phi s_\theta c_\psi \\ c_\theta s_\psi & c_\phi c_\psi + s_\phi s_\theta s_\psi & -s_\phi c_\psi + c_\phi s_\theta s_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{bmatrix}$$

and $Q(\lambda)$ denotes a transformation between angular velocities and Euler angles given by:

$$Q(\lambda) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \tag{2}$$

$\mathbf{F} = [F_x, F_y, F_z]^T$ is the specific force acting on the quadcopter in the body frame which we model as a function of the body velocities and the propeller rpms

using a thrust and drag model based on [8]:

$$
\begin{aligned}
F_x &= -k_x v_x^B \sum_{i=0}^{4} \omega_i \\
F_y &= -k_y v_y^B \sum_{i=0}^{4} \omega_i \\
F_z &= k_\omega \sum_{i=0}^{4} \omega_i^2 + k_z v_z^B \sum_{i=0}^{4} \omega_i + k_h (v_x^{B2} + v_y^{B2})
\end{aligned} \tag{3}
$$

Similarly, $\mathbf{M} = [M_x, M_y, M_z]^T$ is the moment acting on the quadcopter which we model with the following equations:

$$
\begin{aligned}
M_x &= k_p(\omega_1^2 - \omega_2^2 - \omega_3^2 + \omega_4^2) + k_{pv} v_y^B \\
M_y &= k_q(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2) + k_{qv} v_x^B \\
M_z &= k_{r1}(\omega_1 + \omega_2 + \omega_3 + \omega_4) \\
&\quad + k_{r2}(\dot{\omega}_1 + \dot{\omega}_2 + \dot{\omega}_3 + \dot{\omega}_4) - k_{rr} r
\end{aligned} \tag{4}
$$

See Table 1 for the parameter values identified for our platform.

### B. Optimal control problem

Given a state space $X$ and set of admissible controls $U$, the goal is to find a control trajectory $\mathbf{u} : [0, T] \to U$ that steers the system from an initial state $\mathbf{x}_0$ to some target state $S \subset X$ in time $T$, while minimizing the following cost function:

$$
J(\mathbf{x}, \mathbf{u}, T) = \int_0^T ||\mathbf{u}(t)||^2 dt \tag{5}
$$

The optimal control problem can be formulated as

$$
\begin{aligned}
\underset{\mathbf{u},T}{\text{minimize}} \quad & J(\mathbf{x}, \mathbf{u}, T) \\
\text{subject to} \quad & \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \\
& \mathbf{x}(0) = \mathbf{x}_0 \\
& \mathbf{x}(T) \in S
\end{aligned} \tag{6}
$$

Similar to [1] the control problem is transformed into a Nonlinear Programming (NLP) problem using Hermite Simpson transcription. The trajectories $\mathbf{x}(t), \mathbf{u}(t)$ are discretized into $N + 1$ points with a time step $\Delta t = T/N$ such that $\mathbf{x}_k = \mathbf{x}(k\Delta t)$ and $\mathbf{u}_k = \mathbf{u}(k\Delta t)$ Using the AMPL modeling language with the SNOPT NLP solver, the optimal (discretized) trajectory $\mathbf{x}_0^* \dots \mathbf{x}_N^*$ and $\mathbf{u}_0^* \dots \mathbf{u}_N^*$ can be computed.

### C. Dataset generation and network training

A dataset is created by generating optimal trajectories for a range of initial conditions. From these trajectories, a dataset of state-action pairs can be obtained of the form

$$
(\mathbf{x}_i^*, \mathbf{u}_i^*) \quad i = 0, \dots, N
$$

We use these state action pairs to train a Neural Network to approximate the optimal feedback[1] that maps $\mathbf{x}_i^*$ to $\mathbf{u}_i^*$. In all our experiments we use a neural network with 3 Layers and 120 neurons with a ReLU activation function. Similar to [1] we use the mean squared error loss function:

$$
l = ||N(\mathbf{x}^*) - \mathbf{u}^*||^2
$$

with mini-batch size 256 and a starting learning rate of 1e-3.

### D. Adaptive Method

We modify our model by assuming the existance of some constant external moment $\mathbf{M}_{ext} = [M_{ext,x}, M_{ext,y}, M_{ext,z}]^T$ acting on the system. Furthermore we assume that this moment can be measured onboard the quadcopter in real time. The external moment can thus be considered as part of our state vector

$$
\mathbf{x} = [\mathbf{p}, \mathbf{v}, \lambda, \mathbf{\Omega}, \omega, \mathbf{M}_{ext}]^T
$$

The modified system dynamics becomes:

$$
\begin{aligned}
\dot{\mathbf{p}} &= \mathbf{v} \\
\dot{\mathbf{v}} &= \mathbf{g} + R(\lambda)\mathbf{F} \\
\dot{\lambda} &= Q(\lambda)\mathbf{\Omega} \\
I\dot{\mathbf{\Omega}} &= -\mathbf{\Omega} \times I\mathbf{\Omega} + \mathbf{M} + \mathbf{M}_{ext} \\
\dot{\omega} &= ((\omega_{max} - \omega_{min})\mathbf{u} + \omega_{min} - \omega_i)/\tau \\
\dot{\mathbf{M}}_{ext} &= 0
\end{aligned} \tag{7}
$$

Using the same approach as before, we can now generate optimal trajectories for this system and train a network to approximate the optimal state feedback. Note that we now have to sample the values of $\mathbf{M}_{ext}$ as initial conditions for the trajectories. Additionally,

[1] As discussed in [2]: "the Hamilton-Jacobi-Bellman equations are important here as they imply the existence and uniqueness of an optimal state-feedback $\mathbf{u}^*(\mathbf{x})$ which, in turn, allow to consider universal function approximators such as deep neural networks to represent it." (Sánchez-Sánchez 6)

| $k_x$ | $k_y$ | $k_\omega$ | $k_z$ | $k_h$ | $I_x$ | $I_y$ | $I_z$ |
|---|---|---|---|---|---|---|---|
| 1.08e-05 | 9.65e-06 | 4.36e-08 | 2.79e-05 | 6.26e-02 | 0.000906 | 0.001242 | 0.002054 |
| $k_p$ | $k_{pv}$ | $k_q$ | $k_{qv}$ | $k_{r1}$ | $k_{r2}$ | $k_{rr}$ | $\tau$ |
| 1.41e-09 | -7.97e-03 | 1.22e-09 | 1.29e-02 | 2.57e-06 | 4.11e-07 | 8.13e-04 | 0.06 |

**Table 1. Model parameters for the Parrot Bebop quadcopter. The moments of inertia $I_x, I_y, I_z$ are obtained from [9]. All other parameters have been identified by means of system identification experiments (see Appendix A)**

the neural network will now have 3 extra inputs for $M_{ext,x}, M_{ext,y}, M_{ext,z}$. The obtained controller will now use these extra inputs to optimally compensate for the unmodeled moments (assuming they are constant).

For the onboard implementation we will obtain the values of $\mathbf{M}_{ext}$ by estimating

$$I\dot{\mathbf{\Omega}} + \mathbf{\Omega} \times I\mathbf{\Omega} - \mathbf{M} \tag{8}$$

Since the gyroscope measurements tend to be very noisy, $\dot{\mathbf{\Omega}}$ will be obtained by first filtering $\mathbf{\Omega}$ using a 2nd order 8Hz Butterworth lowpass filter. Similarly, $\mathbf{\Omega} \times I\mathbf{\Omega} - \mathbf{M}$ will also be calculated using filtered state variables. It is important to note that the filtering causes our estimates for $\mathbf{M}_{ext}$ to be slightly delayed. For this reason the method might not be stable in the presence of high frequency modeling errors. Furthermore, the controller's output is based on the assumption of a constant external moment so we can expect our method to only be effective if the modeling errors are in a sufficiently low frequency range.

### E. Bang-Bang Controller

In recent work from the TU Delft an approach for time-optimal model predictive control for quadcopters with limited computational resources has been investigated [10]. The proposed 'bang-bang' controller is derived by simplifying the system dynamics such that Pontryagin's minimum principle can give an analytical solution. The model is simplified to a 2 dimensional model where the altitude is assumed to be constant (regulated by the collective thrust $u_T$). The control input to be determined by the optimal control problem is the desired pitch angle $u_\theta$. The evolution of the horizontal position $x$ is given by the following equation:

$$\ddot{x} = u_T \sin u_\theta$$

From these system dynamics, the following Hamiltonian can be constructed:

$$H(\mathbf{x}, \mathbf{u}, \mathbf{p}) = 1 + p_1\dot{x} + p_2 u_T \sin u_\theta$$

With Pontryagin's minimum principle the optimal control input can be derived:

$$u_\theta^* = \arg\min_{u_\theta} \left[ p_2 u_T \sin u_\theta \right]$$

This means $u_\theta$ is either $\pi/2$ or $-\pi/2$ depending on the sign of $p_2$. Because these pitch angles are infeasable to achieve in practice, maximum pitch and roll angles are defined based on the quadcopter's specifications. Nevertheless, the optimal control for this problem will abruptly switch between a maximal and minimal value. Such a controller is known as a bang-bang controller and it is the foundation of the control algorithm implemented in this research. The advantage of this approach is that the only computation required for the optimal control is the switching time between the minimal and maximal pitch angle. Additionally this method requires very little knowledge of the quadcopter's dynamics.

Using the Parrot Bebop drone in the Cyberzoo at the TU Delft, this method has outperformed a classical PID controller by performing waypoint to waypoint flight. Although the approach in this research is quite different from the proposal of our research, the performed flight tests provide an useful reference to compare performance against.

## III. Experimental Setup

The quadcopter used in our experiment is the Parrot Bebop 1 (Figure 3). The on-board software has

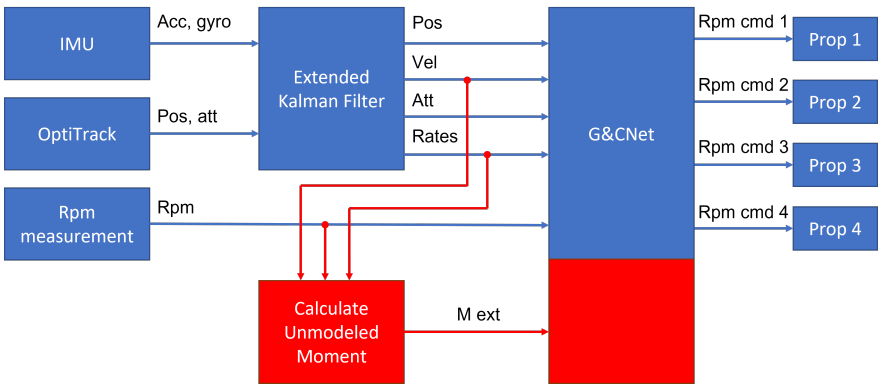



**Figure 2. Control pipeline**

been replaced by the Paparazzi-UAV open-source autopilot project [11]. All computations will run in real-time on the Parrot P7 dual-core CPU Cortex A9 processor. The Parrot Bebop has an MPU650 IMU sensor that will be used to obtain measurements of the specific force and angular velocity along the body axes. Additionally the Bebop has sensors that measure the angular velocities (in rpm) of each of the propellers, which is a necessary requirement for our control method.


**Figure 3. The Parrot Bebop 1 is used as experiment platform. The software is replaced by the Paparazzi UAV open-source autopilot project. Image from [12].**

All flight tests will be performed in The CyberZoo which is a research and test laboratory in the faculty of Aerospace Engineering at the TU Delft. This lab consists of a 10 by 10 meter area surrounded by nets with an OptiTrack motion capture system that can provide position and attitude data in real time. An extended kalman filter is used to fuse the OptiTrack and IMU data in order to obtain an estimate of the position, velocity, attitude and body rates. These state variables are used as an input to the G&CNet along with the rpm measurements. The outputs of the G&CNet will be directly used as rpm commands to the propellers. A diagram of the proposed control pipeline can be seen in Figure 2 where the red part of the diagram shows the calculation of $M_{ext}$ which will be used for the adaptive method. For this calculation the output of the Kalman Filter along with the rpm measurements are used to compute equation 8 as described in section II.D.

## IV. Results & Discussion

### A. Identifying unmodeled effects

*1. Dataset and G&CNet*

Using the system dynamics 1 we generate a dataset of 100,000 power optimal trajectories with a target state defined by:

$$\mathbf{x} = 0 \quad \mathbf{v} = 0 \quad \lambda = 0 \quad \mathbf{\Omega} = 0 \quad \dot{\mathbf{v}} = 0 \quad \dot{\mathbf{\Omega}} = 0 \quad \dot{\omega} = 0$$

The rpm limits are set to:

$$\omega_{min} = 5000, \quad \omega_{max} = 10000$$

and the initial conditions are uniformly sampled from the following intervals:

$$\begin{aligned} x &\in [-5, 5] & y &\in [-5, 5] & z &\in [-1, 1] \\ v_x &\in [-0.5, 0.5] & v_y &\in [-0.5, 0.5] & v_z &\in [-0.5, 0.5] \\ \phi &\in [-40°, 40°] & \theta &\in [-40°, 40°] & \psi &\in [-180°, 180°] \\ p &\in [-1, 1] & q &\in [-1, 1] & r &\in [-1, 1] \end{aligned}$$

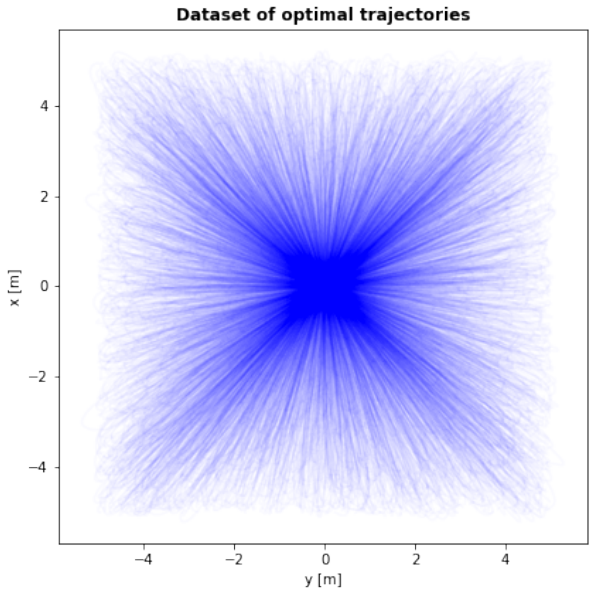



**Figure 4. Visualization of 10,000 trajectories**

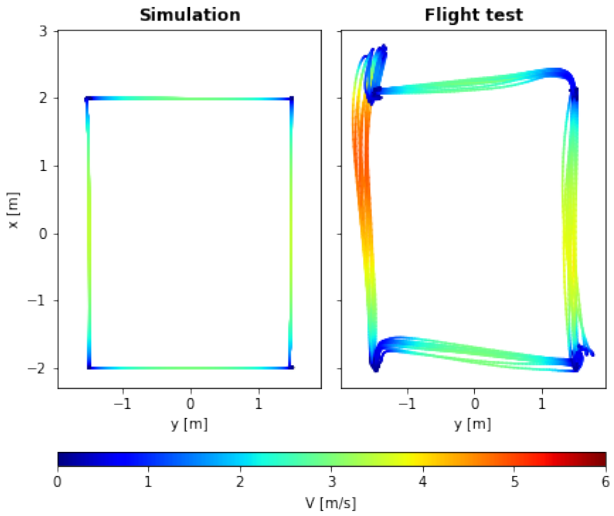



**Figure 5. Trajectory comparison 4 waypoint flight.**

$$\omega_i \in [-\omega_{min}, \omega_{max}] \quad i = 1, \ldots, 4$$

In figure 4 a visualization of this dataset can be seen. We split this dataset into a training set of 90,000 trajectories and a test set of 10,000 trajectories. The G&CNet is trained until a mean squared error of ~0.0003 is obtained on the test set.

*2. Simulation and flight test*

With the trained G&CNet, we simulate the closed loop system dynamics and do a flight test where the drone flies from hover to hover in a 3x4m rectangle. Both in simulation and in the flight test, the drone flies 10 laps in which the target waypoint is switched every 4 seconds. In figure 5 a top down view of the trajectory can be seen for the simulation and the flight test. As expected, in the simulation, the trajectories show significant overlap and the drone consistently arrives at the waypoint without overshoot. In the flight test the trajectories are more spread out and a deviation can be seen in the positive x-direction. The unmodeled effects are especially visible in the forward translation manoeuvre where the drone speeds up too much and overshoots the next waypoint. In figure 6 these forward trajectories are shown both from a top-down view and a sideways view. It can be seen that the drone loses too much altitude causing it to speed up and overshoot.

*3. Unmodeled Moments*

We investigate the unmodeled effects from the forwards translation flight my comparing the measured moments to the modeled moments. The modeled moments is obtained from the moment model from equation 4. The measured moments are calculated from the following equation

$$M_{measured} = I\dot{\mathbf{\Omega}} + \mathbf{\Omega} \times I\mathbf{\Omega}$$

using the filtered (16Hz 2nd order Butterworth non-causal filter) gyroscope measurements. Figure 7 shows these modeled and measured moments for one of the forwards translation trajectories from figure 6. The pitch moment model seems to have the biggest deviations. In figure 8, the difference between the measured pitch moment and the modeled pitch moment $\Delta M_y$ is plotted along with a power spectral density plot. Here it can be seen that in the first 1.3 seconds the unmodeled pitch moment is mostly negative. This modeling error might explain why the drone is diving down so much in the flight test since the negative unmodeled moment could cause the drone to pitch down. Additionally from the power spectral density plot in figure 8 it can be observed that the unmodeled pitch moment is mostly in the low frequency range below 5Hz which is desirable for our adaptive method.

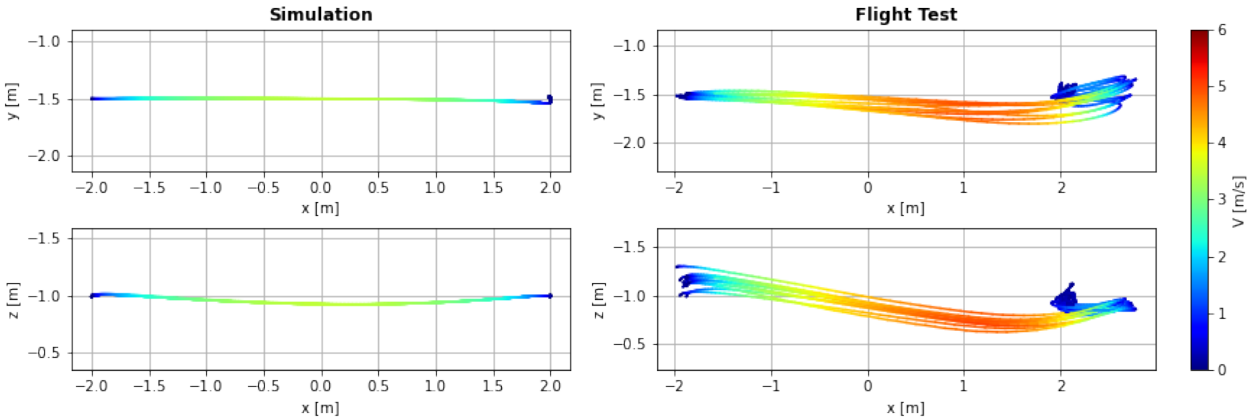



**Figure 6. Trajectory comparison forward translation**

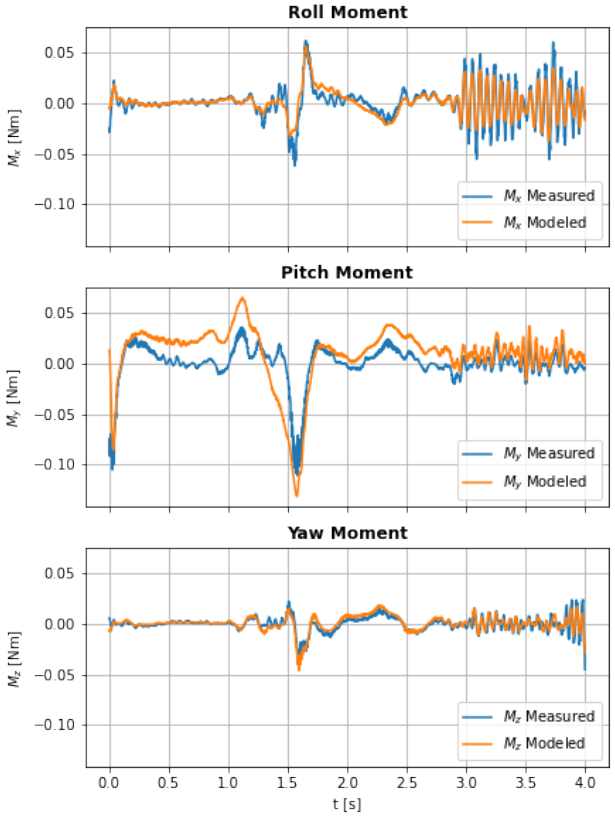



**Figure 7. Measured and modeled moments in forward translation**

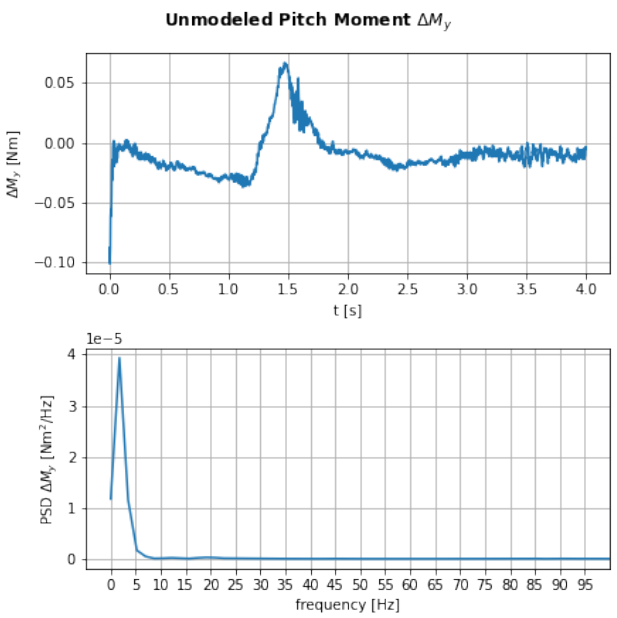



**Figure 8. Unmodeled pitch moment**

## B. Performance Adaptive method

*1. Dataset and G&CNet*

We use the modified system dynamics with external moments from equation 7 to generate another 100,000 power optimal trajectories with the same target state and initial conditions as before, only now we also uniformly sample the external moments from the following intervals:

$$M_{x,ext} \in [-0.04, 0.04]$$
$$M_{y,ext} \in [-0.04, 0.04]$$
$$M_{z,ext} \in [-0.01, 0.01]$$

With the generated dataset we train the adaptive G&CNet with 3 extra $M_{ext}$ inputs to learn the optimal state feedback for the modified system. Again, we train until a mean squared error of ~0.0003 is achieved.

*2. Performance comparison*

With the adaptive G&CNet, we perform a hover to hover flight test using the same 4 waypoints on the 3x4m rectangle and compare the results to the non-adaptive network. In figure 9 a top down view of the adaptive network's trajectory is shown next to the trajectory of the non-adaptive network from the previous flight test. It can be seen that the trajectory no longer deviates towards the positive x-direction and the overshoot in the forward translation manoeuvre is significantly reduced. In figure 10 and 11 the trajectories from the adaptive- and non-adaptive network are compared to the simulated trajectory for a single lap. Here it can be seen that the adaptive network's trajectory remains a lot closer to the simulated trajectory. Also the overshoot in both the y and z axis is significantly less.

*3. Simplified model*

In order to demonstrate the robustness of our adaptive method we now apply our method to a quadcopter model with a slightly simplified pitch and roll moment model:

$$M_x = k_p(\omega_1^2 - \omega_2^2 - \omega_3^2 + \omega_4^2)$$
$$M_y = k_q(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2)$$

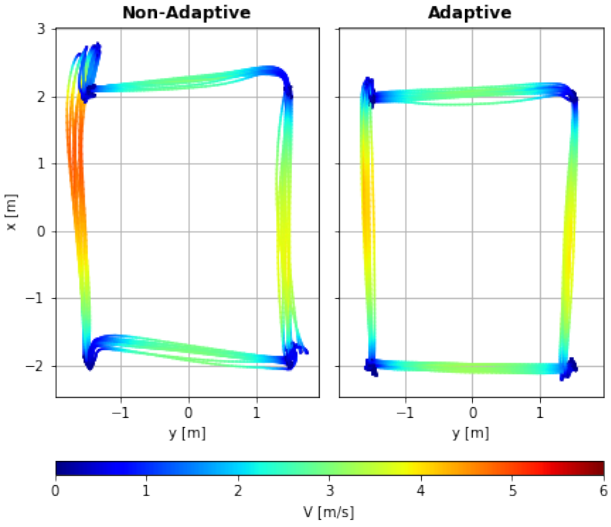



**Figure 9. Trajectory comparison non-adaptive and adaptive**

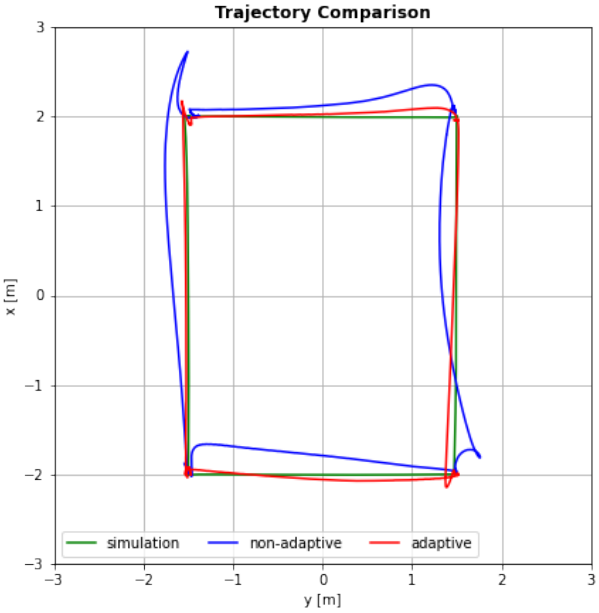



**Figure 10. Trajectory comparison 4 waypoint flight simulation, non-adaptive and adaptive**

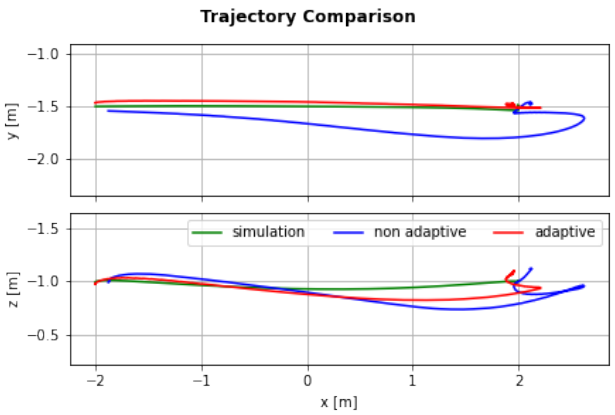



**Figure 11. Trajectory comparison forward translation simulation, non-adaptive and adaptive**

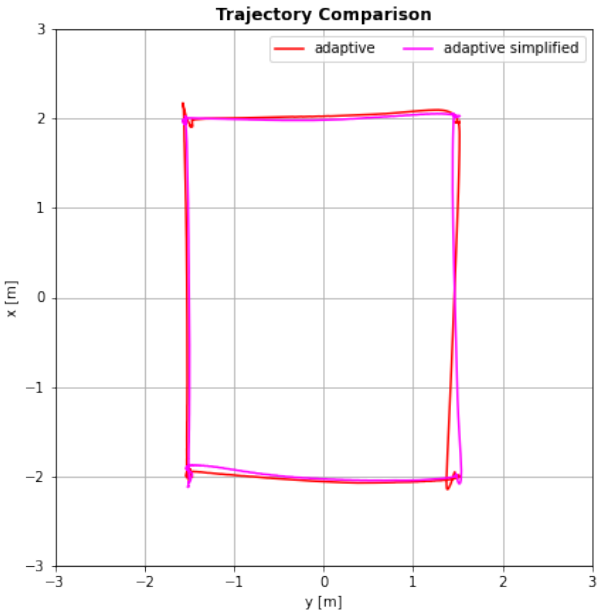



**Figure 12. Trajectory comparison 4 waypoint flight adaptive and adaptive simplified**

Note that the $k_{pv}$ and $k_{qv}$ terms are now left out. With this model we generate a dataset with the same initial conditions and external moments as before. Again we train a G&CNet using the dataset until a mean squared error of $\sim 0.0003$ is reached. We perform the same flight test with our adaptive 'simplified' G&CNet and compare it's performance to the adaptive network. In figure 12 and 13 a trajectory comparison between the two controllers can be seen. Although some differences can be observed between the trajectories, the overall performance is very similar. This similar performance is expected since the moment feedback will still compensate the $k_{pv}v_y^B$ and $k_{qv}v_x^B$ moment terms although they are not explicitly modeled. Moreover these velocity depend moments are in a low frequency range so we can expect the adaptive method to be effective.

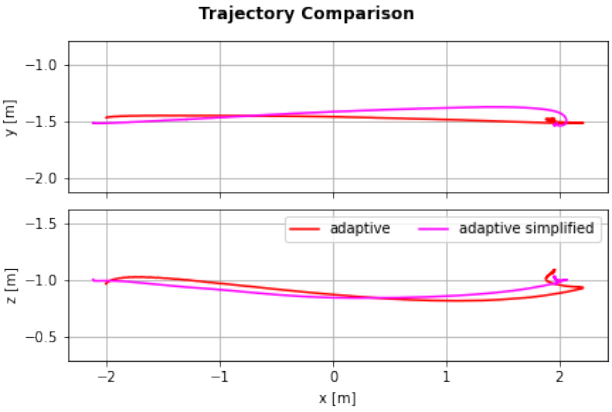



**Figure 13. Trajectory comparison forward translation adaptive and adaptive simplified**

### C. Benchmarking consecutive waypoint flight

*1. Dataset and Network*

For the task of flying through 4 waypoints in a 4x3m rectangle, we will train a G&CNet to reach the waypoint with a forward final velocity in the direction of a 45° yaw angle. Using the modified system dynamics 7 we generate a dataset of 10,000 power optimal

trajectories with a target state given by:

$$
\begin{array}{lll}
x = 0 & y = 0 & z = 0 \\
v_x = V\cos\pi/4 & v_y = V\sin\pi/4 & v_z = 0 \\
\phi = - & \theta = - & \psi = \pi/4 \\
p = 0 & q = 0 & r = 0 \\
\dot{p} = 0 & \dot{q} = 0 & \dot{r} = 0
\end{array}
$$

Where $V > 0$. The rpm limits are set to:

$$\omega_{min} = 3000, \quad \omega_{max} = 12000$$

and the initial conditions are uniformly sampled from the following intervals:

$$
\begin{array}{lll}
x \in [-5,-2] & y \in [-1,1] & z \in [-0.5,0.5] \\
v_x \in [-0.5,5] & v_y \in [-3,3] & v_z \in [-1,1] \\
\phi \in [-40^\circ,40^\circ] & \theta \in [-40^\circ,40^\circ] & \psi \in [-60^\circ,60^\circ] \\
p \in [-1,1] & q \in [-1,1] & r \in [-1,1]
\end{array}
$$

$$\omega_i \in [-\omega_{min}, \omega_{max}] \quad i = 1,\ldots,4$$

In figure 14 a visualization of this dataset can be seen. We split this dataset into a training set of 9000 trajectories and a test set of 1000 trajectories and train until a mean squared error of ~0.0003 is obtained on the test set.

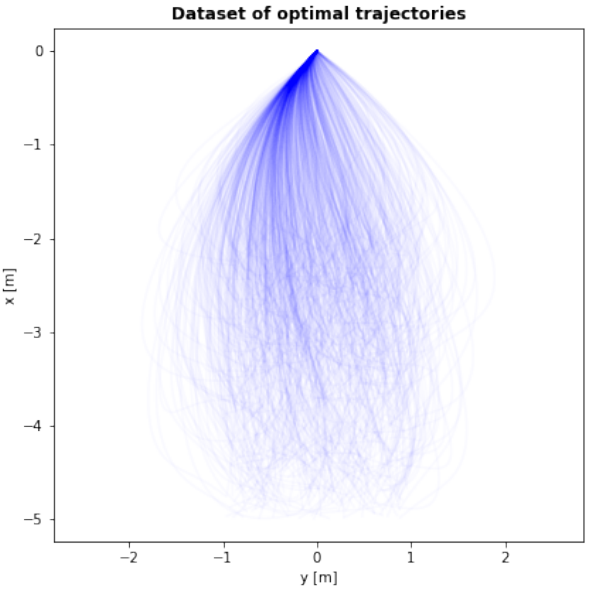



**Figure 14. Dataset free final velocity**

*2. Performance comparison*

With the trained network we perform a flight test where we fly through 4 waypoints in a 3x4m rectangle. The controller switches to the next target waypoint once the drone is within 0.8m from the current target. We compare the performance of our controller to the time optimal bang bang controller from [10] explained in section II.E. Figure 15 shows a top down view of the trajectories from both controllers. It can be seen that the neural network controller has more consistent trajectories each lap while the bang bang controller has more variation each lap. In Figure 16, the lap times and minimum waypoint distances are compared for both controllers. In terms of lap times the G&CNet clearly outperforms the Bang Bang controller with a 0.5 seconds faster average lap time. The minimum waypoint distance for the G&CNet is slightly higher on average by about 5cm.

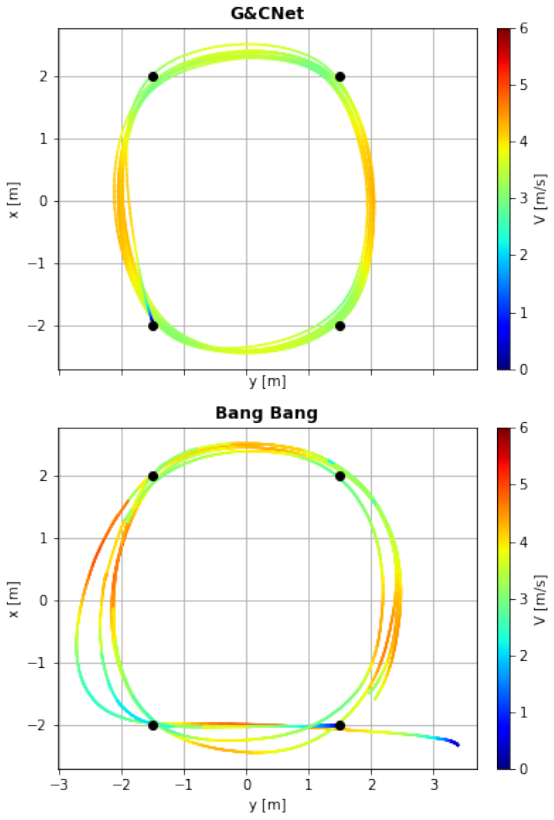



**Figure 15. Trajectory comparison G&CNet and Bang Bang controller**

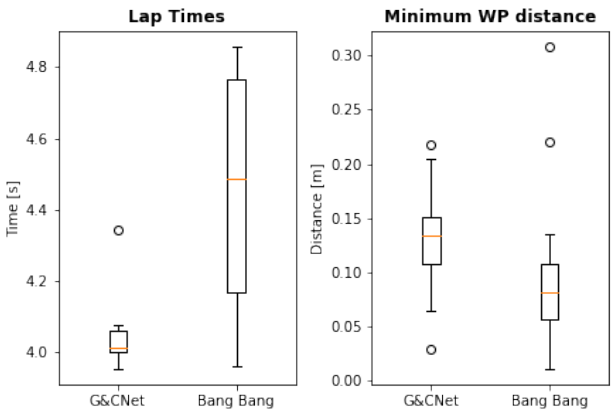



**Figure 16. Lap times and minimum waypoint distances for consecutive waypoint flight**

## V. Conclusion

We have extended the research from [1] by training G&CNets to perform power optimal state to rpm feedback for the 3 dimensional quadcopter model from equation 1. With a flight test we have investigated the real life performance of this G&CNet for hover to hover flight and showed that unmodeled moments significantly influence the flight performance. To mitigate these effects, we implemented an adaptive control strategy that shows a significant improvement in flight performance. Additionally we have shown that the adaptive method applied with a simplified moment model shows similar performance. This implies that less accurate system identification is required for our adaptive method to work. Furthermore we have shown that high speed consecutive waypoint flight can be achieved with our adaptive G&CNet method. We benchmark our results by comparing the performance to a time optimal Bang Bang controller and show 0.5 seconds faster laptimes on a 4x3m track.

Future work can focus on making current G&CNets more time-optimal while retaining their current robustness. A first step would be to investigate whether unmodeled effects or the cost function used in the optimal control problem (Eq. 5) are at the root of the reality gap. The former can be tackled by not only compensating the unmodeled moments, but also errors in thrust, drag forces and actuator delay. The latter can be solved by changing the cost function. In this work the objective of the optimal control problem is purely to minimize power, instead one can formulate a cost function that balances multiple objectives, such as minimizing time and maximizing robustness. Finally, in order to improve the maneuverability of the quadcopter in turns, the G&CNet can be trained on trajectories which account for two consecutive waypoints.

## References


[1] Li, S., Öztürk, E., Wagter, C. D., de Croon, G. C. H. E., and Izzo, D., "Aggressive Online Control of a Quadrotor via Deep Network Representations of Optimality Principles," *CoRR*, Vol. abs/1912.07067, 2019. URL `http://arxiv.org/abs/1912.07067`.

[2] Sánchez-Sánchez, C., and Izzo, D., "Real-time optimal control via Deep Neural Networks: study on landing problems," , 2016. doi: 10.48550/ARXIV.1610.08668, URL `https://arxiv.org/abs/1610.08668`.

[3] Tailor, D., and Izzo, D., "Learning the optimal state-feedback via supervised imitation learning," , 2019. doi: 10.48550/ARXIV.1901.02369, URL `https://arxiv.org/abs/1901.02369`.

[4] Hassanalian, M., and Abdelkefi, A., "Classifications, applications, and design challenges of drones: A review," *Progress in Aerospace Sciences*, Vol. 91, 2017, pp. 99–131.

[5] Kaufmann, E., Loquercio, A., Ranftl, R., Müller, M., Koltun, V., and Scaramuzza, D., "Deep Drone Acrobatics," 2020. doi: 10.48550/ARXIV.2006.05768, URL `https://arxiv.org/abs/2006.05768`.

[6] Faessler, M., Franchi, A., and Scaramuzza, D., "Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate Tracking of High-Speed Trajectories," *IEEE Robotics and Automation Letters*, Vol. 3, No. 2, 2018, pp. 620–626. doi: 10.1109/lra.2017.2776353, URL `https://doi.org/10.1109/lra.2017.2776353`.

[7] Bryson, A., "Optimal control-1950 to 1985," *IEEE Control Systems Magazine*, Vol. 16, No. 3, 1996, pp. 26–33. doi: 10.1109/37.506395.

[8] Svacha, J., Mohta, K., and Kumar, V. R., "Improving quadrotor trajectory tracking by compensating for aerodynamic effects," *2017 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2017, pp. 860–866.

[9] Sun, S., de Visser, C. C., and Chu, Q., "Quadrotor Gray-Box Model Identification from High-Speed Flight Data," *Journal of Aircraft*, Vol. 56, No. 2, 2019, pp. 645–661. doi: 10.2514/1.c035135, URL `https://doi.org/10.2514/1.c035135`.

[10] Westenberger, J., "Time-Optimal Control for Tiny Quadcopters," Master's Thesis, TU Delft Aerospace

Engineering, 3 2021. An optional note.

[11] Gati, B., "Open source autopilot for academic research - The Paparazzi system," *2013 American Control Conference*, IEEE, 2013. doi: 10.1109/acc.2013.6580045, URL `https://doi.org/10.1109/acc.2013.6580045`.

[12] Li, S., Ozo, M. M. O. I., De Wagter, C., and de Croon, G. C. H. E., "Autonomous drone race: A computationally efficient vision-based navigation and control strategy," 2018. doi: 10.48550/ARXIV.1809.05958, URL `https://arxiv.org/abs/1809.05958`.

## A. System Identification

Identification of values in Table 1

### A. Actuator model

In order to fit the actuator model

$$\dot{\omega}_i = (\omega_{ref,i} - \omega_i)/\tau$$

the rpm values from the propellers along with the reference rpm commands need to be logged. For each propeller an array of observed rpm values $\omega_i[k], k = 0, \ldots, N$ along with an array of reference rpm values $\omega_{ref,i}[k], k = 0, \ldots, N$ is logged. Also the numerical derivative $\dot{\omega}_i$ is computed. From these arrays the regression problem is formulated as:

$$\frac{1}{\tau} = \begin{bmatrix} \omega_{ref,i}[0] - \omega_i[0] \\ \vdots \\ \omega_{ref,i}[N] - \omega_i[N] \end{bmatrix}^{\dagger} \begin{bmatrix} \dot{\omega}_i[0] \\ \vdots \\ \dot{\omega}_i[N] \end{bmatrix}$$

Where † denotes the pseudo-inverse of a matrix given by $A^{\dagger} = (A^T A)^{-1} A^T$. Figure 17 shows the regression fits for each of the propellers. This fit was performed using data from an autonomous flight from hover to hover in a 4x4m square.

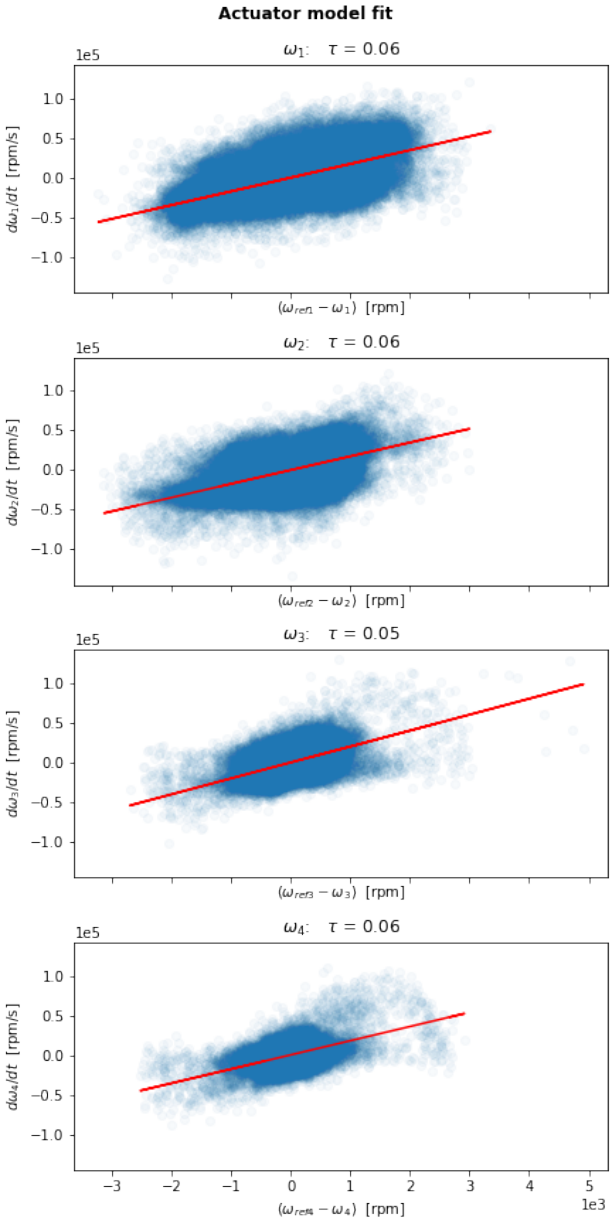



**Figure 17. Actuator model fit**

### B. Thrust and Drag model

For identifying the thrust and drag model, measurement of the specific forces $F_x, F_y, F_y$ are required. These measurements are obtained from the accelerometer which essentially measures this specific force in the quadcopter's body frame. According to our model these accelerometer measurements should satisfy the

following equation:

$$\mathbf{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix}$$
$$= \begin{bmatrix} -k_x v_x^B \sum_{i=0}^{4} \omega_i \\ -k_y v_y^B \sum_{i=0}^{4} \omega_i \\ k_\omega \sum_{i=0}^{4} \omega_i^2 + k_z v_z^B \sum_{i=0}^{4} \omega_i + k_h (v_x^{B2} + v_y^{B2}) \end{bmatrix}$$

Since accelerometers tend to have biases a more accurate model would be:

$$\mathbf{a} = \begin{bmatrix} -k_x v_x^B \sum_{i=0}^{4} \omega_i \\ -k_y v_y^B \sum_{i=0}^{4} \omega_i \\ k_\omega \sum_{i=0}^{4} \omega_i^2 + k_z v_z^B \sum_{i=0}^{4} \omega_i + k_h (v_x^{B2} + v_y^{B2}) \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

Additional measurements required for solving the regression problems are the observed rpm values $\omega_i$ and the body velocities $v_x^B, v_y^B, v_z^B$. The drag regression problems along the x-axis can be formulated as:

$$\begin{bmatrix} \hat{k}_x \\ b_x \end{bmatrix} = \begin{bmatrix} -u[0] \sum \omega_i[0] & 1 \\ \vdots & \vdots \\ -u[N] \sum \omega_i[N] & 1 \end{bmatrix}^\dagger \begin{bmatrix} a_x[0] \\ \vdots \\ a_x[N] \end{bmatrix}$$

Similarly the regression problem for y-component of drag is given by:

$$\begin{bmatrix} \hat{k}_y \\ b_y \end{bmatrix} = \begin{bmatrix} -v[0] \sum \omega_i[0] & 1 \\ \vdots & \vdots \\ -v[N] \sum \omega_i[N] & 1 \end{bmatrix}^\dagger \begin{bmatrix} a_y[0] \\ \vdots \\ a_y[N] \end{bmatrix}$$

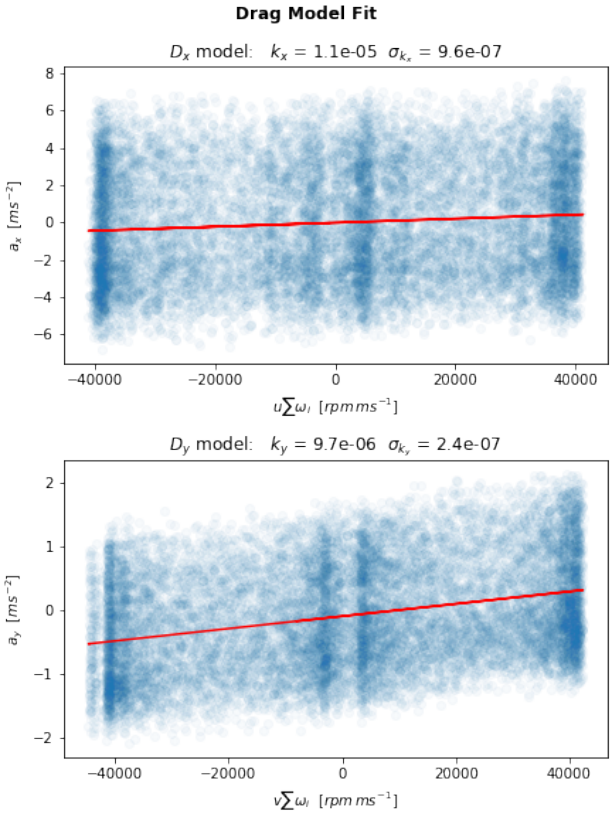



**Figure 18. Drag model fit**

And finally the thrust regression problem is:

$$\begin{bmatrix} \hat{k}_\omega \\ \hat{k}_z \\ \hat{k}_h \\ b_z \end{bmatrix} = \begin{bmatrix} \sum \omega_i[0]^2 & w[0] \sum \omega_i[0] & v_x^B[0]^2 + v_y^B[0]^2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \sum \omega_i[N]^2 & w[N] \sum \omega_i[N] & v_x^B[N]^2 + v_y^B[N]^2 & 1 \end{bmatrix}^\dagger \begin{bmatrix} a_z[0] \\ \vdots \\ a_z[N] \end{bmatrix}$$

The drag model fit was again performed using data from an automatic flight between 4 waypoints where x- and y body velocities up to 1.4 m/s where reached in both the positive and negative direction. The thrust model was identified using a similar but slightly more aggressive automatic flight where speeds up to 5m/s were reached . Figure 18 shows a regression plot of the

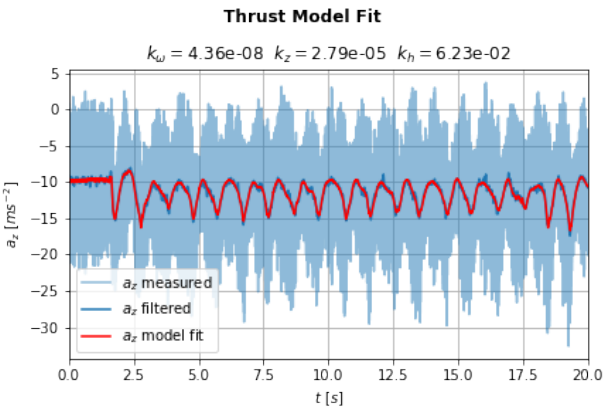



**Figure 19. Thrust model fit**

drag fit and figure 19 shows a plot of the thrust fit compared to the measured z-acceleration. Additionally, sthe filtered (16Hz zero-phase 2d order Butterworth lowpass filter) accelerometer data is shown.

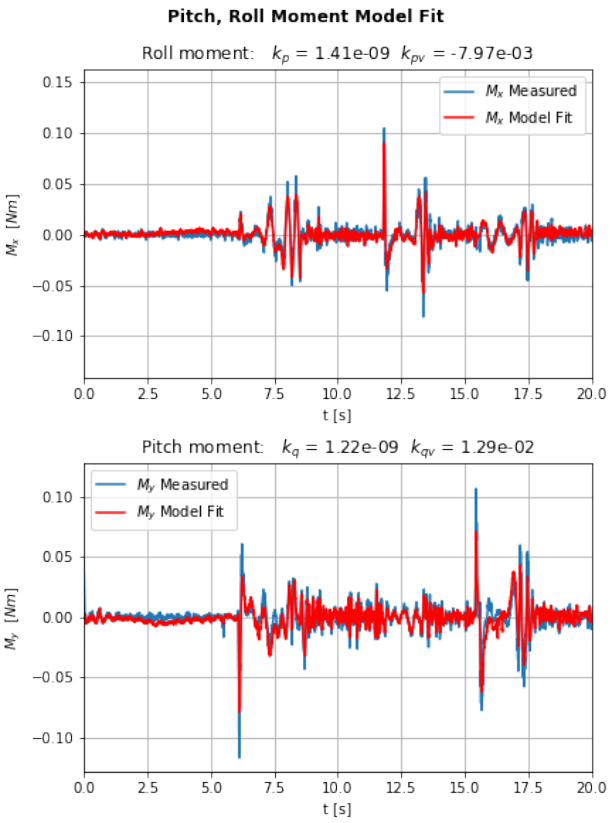



**Figure 20. Pitch and Roll moment model**

### C. Moments model

For the moment model identification, estimates of the moments $M_x, M_y, M_z$ are calculated from gyroscope measurement $\Omega = [p, q, r]^T$ and their derivatives using the following equation:

$$\mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = I\dot{\Omega} + \Omega \times I\Omega = \begin{bmatrix} I_x\dot{p} + (I_z - I_y)qr \\ I_y\dot{q} + (I_x - I_z)pr \\ I_z\dot{r} + (I_y - I_x)pq \end{bmatrix}$$

In order to reduce noise, $p, q$ and $r$ are low passed filtered at 16Hz using a zero-phase 2nd order Butterworth filter. These moments are then used for the following regression problems:

$$\begin{bmatrix} k_p \\ b_p \end{bmatrix} = \begin{bmatrix} \omega_1[0]^2 - \omega_2[0]^2 - \omega 3[0]^2 + \omega_4[0]^2 & 1 \\ \vdots & \vdots \\ \omega_1[N]^2 - \omega_2[N]^2 - \omega 3[N]^2 + \omega_4[N]^2 & 1 \end{bmatrix}^\dagger \begin{bmatrix} M_x[0] \\ \vdots \\ M_x[N] \end{bmatrix}$$

$$\begin{bmatrix} k_q \\ b_q \end{bmatrix} = \begin{bmatrix} \omega_1[0]^2 + \omega_2[0]^2 - \omega 3[0]^2 - \omega_4[0]^2 & 1 \\ \vdots & \vdots \\ \omega_1[N]^2 + \omega_2[N]^2 - \omega 3[N]^2 - \omega_4[N]^2 & 1 \end{bmatrix}^\dagger \begin{bmatrix} M_y[0] \\ \vdots \\ M_y[N] \end{bmatrix}$$

$$\begin{bmatrix} k_{r1} \\ k_{r2} \\ k_{rr} \\ b_r \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^4 (-1)^{i+1}\omega_i[0] & \sum_{i=0}^4 (-1)^{i+1}\dot{\omega}_i[0] & -r[0] & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \sum_{i=0}^4 (-1)^{i+1}\omega_i[N] & \sum_{i=0}^4 (-1)^{i+1}\dot{\omega}_i[N] & -r[N] & 1 \end{bmatrix}^\dagger \begin{bmatrix} M_y[0] \\ \vdots \\ M_y[N] \end{bmatrix}$$

The pitch and roll moment model was identified using flight data from an automatic flight where speeds up

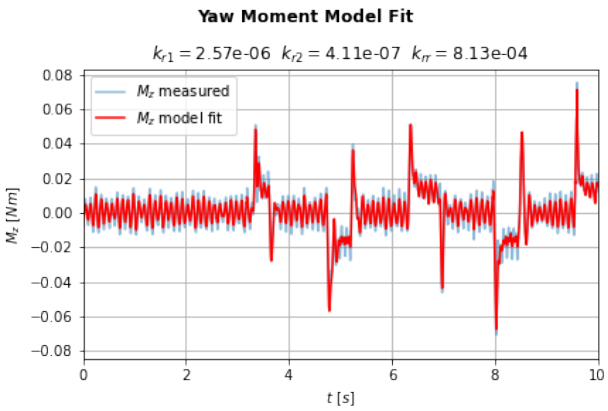



**Figure 21. Yaw moment model**

to 3m/s where reached. The yaw moment model was identified from an automatic flight in hover where yaw commands where given that alternated from 0 to 180 degrees. Figure 20 shows a plot of the pitch and roll moment fit and figure 21 shows a plot of the yaw moment fit.

# Part II

# Literature Review

# 2

# Literature Review

## 2.1. Optimal control theory

One of the main theoretical basis of this research is optimal control theory. This branch of mathematical optimization focuses on finding controls for a dynamical system that optimizes an objective function. The methods for deriving these optimal control policies has mostly been developed in the 1950s by Lev Pontryagin and Richard Bellman [2]. Although the theory has been around for a long time, practical implementations remained limited due to the computational power required to obtain the solutions. This chapter will give a brief introduction of the theory followed by relevant research about optimal control applied to quadcopters

### 2.1.1. Optimal control problem formulation

In optimal control theory the dynamical system is described by the following equation.

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

Here $\mathbf{x} \in R^n$ describes the state vector, $\mathbf{u}$ describes the control vector (that is often part of a restricted set of allowed controls $U \subset R^n$) and $f$ is the function that describes the dynamics of the system. The goal is to find a control $\mathbf{u}(t), t \in [0, T]$ that steers the system from an inital state $\mathbf{x}_0$ to a specific final state $\mathbf{x}_f$ in time $T$, while minimizing a cost function of the form:

$$J(\mathbf{x}, \mathbf{u}, T) = \int_0^T g(\mathbf{x}(t), \mathbf{u}(t))dt \tag{2.1}$$

The optimal control problem can be formulated as

$$\begin{aligned} \underset{\mathbf{u},T}{\text{minimize}} \quad & J(\mathbf{x}, \mathbf{u}, T) \\ \text{subject to} \quad & \dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{x}(T) = \mathbf{x}_f \end{aligned} \tag{2.2}$$

The Hamilton–Jacobi–Bellman (HJB) equation and Pontryagin's maximum principle are the main analytical tools used to solve and analyze such problems.

### 2.1.2. The Hamilton–Jacobi–Bellman equation

For optimal control problems as formulated before, the HJB equation gives a necessary and sufficient condition for optimality. The HJB equation can be derived from Bellman's principle of optimality [2] which results in a partial differential equation of which the solution is the value function of the optimal control problem. The value function denoted by $J^*$ is a real valued function that defines for each value of $\mathbf{x}_0$ the optimal cost to steer the system to $\mathbf{x}_f$.

$$J^*(\mathbf{x}_0) = \min_{\mathbf{u},T} J(\mathbf{x}, \mathbf{u}, T)$$

The Hamilton–Jacobi–Bellman partial differential equation is

$$\min_{\mathbf{u}} \left[ g(\mathbf{x}, \mathbf{u}) + f(\mathbf{x}, \mathbf{u}) \cdot \nabla_{\mathbf{x}} J^* \right] = 0$$

Subject to the terminal condition

$$J^*(\mathbf{x}_f) = 0$$

The solution to these equations is the value function $J^*$ which can be used to obtain the optimal control input $\mathbf{u}^*$. The optimal control policy is then

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} \left[ g(\mathbf{x}, \mathbf{u}) + f(\mathbf{x}, \mathbf{u}) \cdot \nabla_{\mathbf{x}} J^* \right]$$

### 2.1.3. Pontryagin's minimum principle

Unlike the HJB equation, Pontryagin's minimum principle only provides a necessary, but not sufficient condition for optimality. The principle is derived using methods from calculus of variations. It states that any optimal control $\mathbf{u}^*(t)$ and state trajectory $\mathbf{x}^*(t)$ must satisfy a so-called Hamiltonian system. This system is described by a Hamiltonian $H$ defined for all $t \in [0, T]$ by:

$$H(\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}(t)) = g(\mathbf{x}(t), \mathbf{u}(t)) + \mathbf{p}^T(t) f(\mathbf{x}(t), \mathbf{u}(t))$$

where $\mathbf{p}(t)$ are introduced as time varying Lagrange multiplier vector. Pontryagin's minimum principle states that this Hamiltonian should be minimal, meaning for the optimal trajectory $\mathbf{x}^*, \mathbf{u}^*$ along with the corresponding Lagrange multiplier vector $\mathbf{p}^*$ the following should hold:

$$H(\mathbf{x}^*(t), \mathbf{u}^*(t), \mathbf{p}^*(t)) \leq H(\mathbf{x}^*(t), \mathbf{u}, \mathbf{p}^*(t)) \quad \text{for all } \mathbf{u} \in U \tag{2.3}$$

It also defines a boundary value problem for the Lagrange multipliers:

$$\dot{\mathbf{p}}^*(t) = -\nabla_{\mathbf{x}} H(\mathbf{x}^*, \mathbf{u}^*, \mathbf{p}^*) \tag{2.4}$$

$$\mathbf{p}^*(T) = 0 \tag{2.5}$$

These conditions are necessary for $\mathbf{x}^*$ and $\mathbf{u}^*$ to be optimal. In certain cases these conditions are sufficient [9] in which case the optimal control input $\mathbf{u}^*$ can be expressed in terms of $\mathbf{x}$ and $\mathbf{p}$ by calculating:

$$\mathbf{u}^* = \arg\min_{\mathbf{u}} H(\mathbf{x}, \mathbf{u}, \mathbf{p})$$

### 2.1.4. Direct transcription Methods

Direct transcription methods form an alternative to using the HJB equations or Pontryagin's minimum principle to obtain numerical solutions to an optimal control problem. Instead of applying mathematical optimization to the continuous control problem, the control problem is transformed into a Nonlinear Programming (NLP) problem that can be solved with various well-known methods. The trajectories $\mathbf{x}(t), \mathbf{u}(t)$ are discretized into $N + 1$ points with a time step $\Delta t = T/N$ such that $\mathbf{x}_k = \mathbf{x}(k\Delta t)$ and $\mathbf{u}_k = \mathbf{u}(k\Delta t)$ The discrete optimal control problem can then be formulated as:

$$\begin{aligned} \underset{\mathbf{u}, T}{\text{minimize}} \quad & J_d(\mathbf{x}_0, \ldots, \mathbf{x}_N, \mathbf{u}_0, \ldots, \mathbf{u}_N, T) \\ \text{subject to} \quad & \mathbf{x}_{k+1} = f_d(\mathbf{x}_k, \mathbf{u}_k) \\ & \mathbf{x}_0 = \mathbf{x}_0 \\ & \mathbf{x}_N = \mathbf{x}_f \end{aligned} \tag{2.6}$$

Here $J_d$ and $f_d$ are the discrete approximations of the cost function and system dynamics respectively. A popular method of discretization often used for trajectory optimisation is Hermite-Simpson Collocation. In this method, the state is represented by a cubic-Hermite spline (i.e a third degree polynomial defined by its values and first derivatives at the end points). In combination with Simpson's rule for the approximation of integrals, both $J_d$ and $f_d$ can be computed. With this formulation of the optimal control problem, popular NLP solvers like SNOPT can compute the optimal (discretized) trajectory $\mathbf{x}_0^* \ldots \mathbf{x}_N^*$ and $\mathbf{u}_0^* \ldots \mathbf{u}_N^*$.

### 2.1.5. Optimal control applied to quadcopters

In recent work from the TU Delft an approach for time-optimal model predictive control for quadcopters with limited computational resources has been investigated [17]. The proposed 'bang-bang' controller is derived by simplifying the system dynamics such that Pontryagin's minimum principle can give an analytical solution. The model is simplified to a 2 dimensional model where the altitude is assumed to be constant (regulated by the collective thrust $u_T$). The control input to be determined by the optimal control problem is the desired pitch angle $u_\theta$. The evolution of the horizontal position $x$ is given by the following equation:

$$\ddot{x} = u_T \sin u_\theta$$

From these system dynamics, the following Hamiltonian can be constructed:

$$H(\mathbf{x}, \mathbf{u}, \mathbf{p}) = 1 + p_1 \dot{x} + p_2 u_T \sin u_\theta$$

With Pontryagin's minimum principle the optimal control input can be derived:

$$u_\theta^* = \arg\min_{u_\theta} \left[ p_2 u_T \sin u_\theta \right]$$

This means $u_\theta$ is either $\pi/2$ or $-\pi/2$ depending on the sign of $p_2$. Because these pitch angles are infeasable to achieve in practice, maximum pitch and roll angles are defined based on the quadcopter's specifications. Nevertheless, the optimal control for this problem will abruptly switch between a maximal and minimal value. Such a controller is known as a bang-bang controller and it is the foundation of the control algorithm implemented in this research. The advantage of this approach is that the only computation required for the optimal control is the switching time between the minimal and maximal pitch angle. Additionally this method requires very little knowledge of the quadcopter's dynamics.

Using the Parrot Bebop drone in the Cyberzoo at the TU Delft, this method has outperformed a classical PID controller by performing waypoint to waypoint flight. Although the approach in this research is quite different from the proposal of this thesis, the performed flight tests provide an important benchmark to compare performance against.

## 2.2. Machine Learning

Another fundamental topic for this research is Machine Learning (ML): the study of computer algorithms that can improve automatically through experience and by the use of data. More specifically, we will focus on the application of Deep Neural Networks to optimal control problems for achieving high speed autonomous flight. This chapter will start with a brief introduction of the theory of Neural networks followed by recent research covering several approaches for applying ML to quadcopter control.

### 2.2.1. Neural networks

Many different types of neural networks exist, but here we will focus on a specific class of feed-forward neural networks known as a multilayered perceptron. In it's simplest form this a function $f : R^{d_{in}} \rightarrow R^{d_{out}}$ that consists of a composition of affine functions $A_i(x) = W_i x + b_i$ and activation functions $\sigma_i$ such that when $x$ is the input to the neural network, the output can be calculated as follows:

$$\begin{aligned} y_1 &= \sigma_1(W_1 x + b_1) \\ y_2 &= \sigma_2(W_2 y_1 + b_2) \\ &\vdots \\ f(x) &= y_n = \sigma_n(W_n y_{n-1} + b_n) \end{aligned} \tag{2.7}$$

Here the intermediate values $y_i$ are known as the layers of the neural network, $W_i$ is called the weight matrix and has dimensions that match the dimensions of consecutive layers and $b_i$ are called the biases. Popular choises for activation functions $\sigma_i$ are the sigmoid and relu function:

$$\sigma_{\text{sigmoid}}(x) = \frac{1}{1 + \exp(-x)}$$
$$\sigma_{\text{Relu}}(x) = \max(0, x)$$

Multilayered perceptrons have the special property that under a set of mild conditions, they can approximate any continuous function [5]. This result is known as the Universal Approximation Theorem and it motivates the use of Neural Networks for approximating arbitrary functions. The process of learning can be described as systematically changing the weights and biases $W_i, b_i, i = 1, \ldots, n$ in such a way that the neural network more closely approximates a desired function in terms of some performance metric.

### 2.2.2. Supervised learning

In supervised machine learning, training is done by using a dataset which contains input output pairs. These input-output pairs serve as an ideal example of what the network should compute. When training, the neural network's output is compared to the output specified by the training data. This difference between outputs is quantified in terms of a loss function which is optimized by changing the parameters of the neural network according to a gradient based optimization method.

### 2.2.3. Reinforcement learning

In contrast reinforcement learning does not use a dataset, but a simulated environment in which the neural network is optimized based on rewards. In this simulated environment the neural network uses the state of the system to compute an output that determines the action that is taken by the simulated agent. After each action the system moves to a new state and a reward is provided. The goal of the agent is to maximizes the expected cumulative reward. Policy gradient methods are a popular training method for reinforcement learning problems. These methods attempt to estimate the gradient of the cumulative reward with respect to the Neural networks parameters in order to apply gradient based optimization.

### 2.2.4. ML for trajectory generation

In recent research from the Robotics and Perception Group from the University of Zurich [12], reinforcement learning is used to train a neural network to give optimal controls for a quadcopter in a drone racing track. The training is done in a simulated environment where the quadcopter is modeled by a 6 degree-of-freedom rigid body that is acted upon by 4 forces generated by the propellers. A function is defined that calculates a reward based on the progress between the gates, some safety margins and a penalty for gate collisions. Using a policy gradient method called the Proximal Policy Optimization (PPO) algorithm a feed forward neural network is trained with a architecture as seen in figure 2.1. This

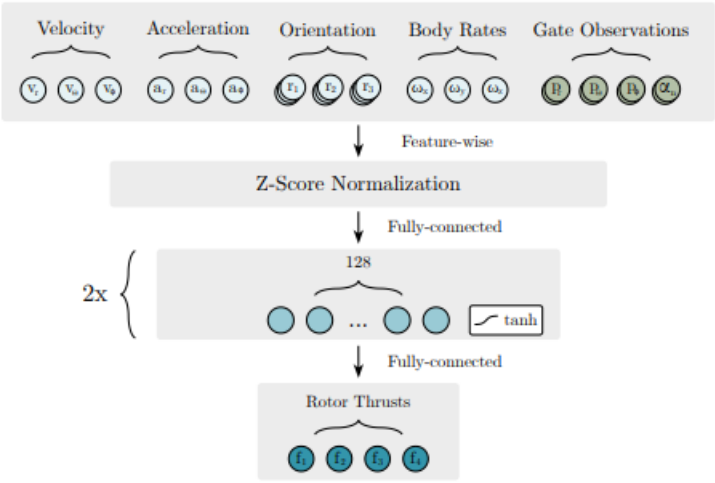



**Figure 2.1:** Illustration of Neural Network from [12]

network takes the velocity, acceleration, attitude and body rates as input along with a series of gate observations described by relative positions and angles. These values are normalized and plugged into a 2 layered neural network with tanh activation function that computes the 4 forces that are used as control input in the simulator. Using randomly sampled race tracks this network is successfully trained. Several existing race tracks (including the Alpha Pilot race track) have been used to verify

the performance in simulation. The study also included a real flight test to validate performance, but the neural network's control output was not directly used. The neural network outputs were used to generate optimal trajectories that were successfully tracked using a model predictive controller on a real drone.

### 2.2.5. ML for aggressive trajectory tracking

In other research from the Robotics and Perception Group [6] a neural network is trained using supervised learning based on a privileged expert to track aggressive trajectories. The approach relies solely on on-board sensing and computation. Figures 2.2 shows a composite image of the quadcopter performing a flip which gives an indication of the type of trajectories considered in this research. In

**Figure 2.2:** The quadcopter performs a Matty Flip. Image from [6]

contrast to [12], trajectory planning is not considered here. The research assumes access to a reference trajectory that specifies an aggressive feasible manoeuvre. The real challenge in this research is that the Neural network controller will only use on-board sensing via the IMU and camera. The privileged expert used for training is a model predictive controller that does have access to all states of the quadcopter. Using a high fidelity simulator, a dataset with optimal control is generated along with camera images processed into 'feature tracks' [1]. Using this dataset a Neural network is trained to produce collective thrust and body rate commands from the processed camera images along with IMU sensor measurements. The trained model could directly be deployed on a real quadcopter and has successfully demonstrated a wide range of acrobatics maneuvers

### 2.2.6. G&CNets

In recent research from ESA's Advanced Concepts Team and the TU delft (the main research line from this proposal) supervised learning is used to train a neural network to give optimal controls. The dataset used for training consists of optimal trajectories generated with a direct transcription method. The goal of this approach is to achieve both guidance and control by a single neural network controller. This method was introduced in the work of ESA's Advanced Concepts Team in 2016 with the goal of finding the optimal control action during a pinpoint landing for several spacecrafts and a quadcopter [10]. This research was continued in collaboration with the TU Delft where the method was tested on a real quadcopter [8]. The dynamic quadcopter model was described by the following state and control input:

$$\mathbf{x} = [x, z, v_x, v_z, \theta, q]^T, \quad \mathbf{u} = [u_1, u_2]^T$$

[1] Feature tracks are an abstraction from camera frames that depend primarily on scene geometry, rather than surface appearance.

where $(x, z)$ is the position $(v_x, v_z)$ the velocity, $\theta$ the pitch angle and $q$ is the pitch rate. The control inputs $u_1$ and $u_2$ represent the left and right throttles respectively. The dynamics are described by:

$$\mathbf{x} = \begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_z} \\ \dot{\theta} \\ \dot{q} \end{bmatrix} = f(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} v_x \\ v_z \\ \left[(u_1 + u_2)\frac{\Delta F}{m} + 2\frac{\underline{F}}{m}\right] \sin\theta - \beta v_x \\ \left[(u_1 + u_2)\frac{\Delta F}{m} + 2\frac{\underline{F}}{m}\right] \cos\theta - g - \beta v_z \\ q \\ \frac{L}{I_{xx}} \Delta F (u_2 - u_1) \end{bmatrix}$$

Where $m$ is the mass, $\beta$ the drag coefficient, $\Delta F$ is the range of thrust and $\underline{F}$ is the minimal thrust. The optimal control problem is formulated as:

$$\begin{aligned} \underset{\mathbf{u}, T}{\text{minimize}} \quad & (1-\epsilon)T + \epsilon \int_0^T u_1(t)^2 + u_2(t)^2 dt \\ \text{subject to} \quad & \mathbf{x} = f(\mathbf{x}, \mathbf{u}) \\ & \mathbf{x}(0) = \mathbf{x}_0 \\ & \mathbf{x}(T) = \mathbf{0} \end{aligned}$$

Where $\epsilon \in [0, 1]$ is a hybridization parameter that can be varied to obtain time optimal control (at $\epsilon = 0$) or power optimal control (at $\epsilon = 1$). Using Hermite Simpson transcription and the SNOPT NLP solver, optimal trajectories were generated for a range of initial condition. Two datasets where generated, one for $\epsilon = 0.5$ and one for $\epsilon = 0.2$. Using supervised learning two neural networks were trained on the state action pairs. In a real world flight test the performance of these networks were compared to a commonly used method DiffG&C. The flight tests showed that even the G&CNet with $\epsilon = 0.5$ was competitive with the DiffG&C while the G&CNet with $\epsilon = 0.2$ even significantly outperformed it.

## 2.3. System identification

In the previous sections we have already seen some examples of quadcopter models. In the work of [17] the model was simplified to such an extent that almost no knowledge of the quadcopter is required to implement the control method. In the research on G&CNets the 2 dimensional model that was used already requires some specific parameters to be estimated like mass, thrust range, moment of inertia and drag coefficients. The estimation of these parameters is crucial for the final performance of the model based controller. More complicated controllers that are based on higher dimensional quadcopter models, require even more parameters to be estimated. This section will focus on the derivation of such a 6 degree-of-freedom quadcopter models covering aerodynamic thrust and drag models along with actuator models.

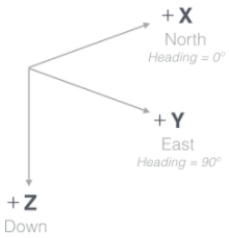
### 2.3.1. Quadcopter dynamic equations

The dynamics of a quadcopter can be modeled using rigid body mechanics. The quadcopter can be described as a rigid body with mass $m$ and moment of inertia matrix $I$. The actuators of the quadcopter along with aerodynamic forces produce a force **F** and moment **M** that can be best described along the axis of the rigid body frame. The axes of the world frame and body frame can be seen in figure 2.3. By using Newton's second law of motion and Euler's equation of rigid body dynamics, the following dynamical system can be derived:

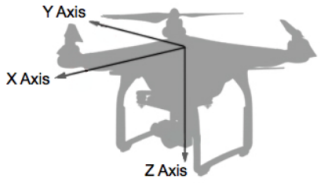
$$\begin{aligned} \ddot{\mathbf{x}} &= \mathbf{g} + \frac{1}{m} R(\lambda) \mathbf{F} \\ I\dot{\Omega} &= -\Omega \times I\Omega + \mathbf{M} \end{aligned}$$

Where $\mathbf{g} = [0, 0, g]^T$, $\mathbf{x} = [x, y, z]^T$ is the position and $\Omega = [p, q, r]^T$ is the angular rate in body frame. The orientation of the quadcopter will be described by the euler angles $\lambda = [\phi, \theta, \psi]^T$ that each correspond to a rotation around one of the axis in the world frame.

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \quad R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.8)$$




**(a)** The axis convention used for the world frame.




**(b)** The axis convention used for the body axes.

**Figure 2.3:** Axis convention used in this sections. Images from dji website

The orientation of the body frame with respect to the world frame is written in terms of these rotation matrices as follows:

$$R(\lambda) = R(\phi,\theta,\psi) = R_z(\psi)R_y(\theta)R_x(\phi)$$
$$= \begin{bmatrix} \cos\theta\cos\psi & -\cos\phi\sin\psi+\sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi+\cos\phi\sin\theta\cos\psi \\ \cos\theta\sin\psi & \cos\phi\cos\psi+\sin\phi\sin\theta\sin\psi & -\sin\phi\cos\psi+\cos\phi\sin\theta\sin\psi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}$$

Altogether the rigid body equations become

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= \mathbf{g} + \frac{1}{m}R(\lambda)\mathbf{F} \\ \dot{\lambda} &= Q(\lambda)\Omega \\ I\dot{\Omega} &= -\Omega \times I\Omega + \mathbf{M} \end{aligned}$$

Here $Q(\lambda)$ denotes a transformation between angular velocities and Euler angles given by

$$Q(\lambda) = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix} \tag{2.9}$$

The rigid body mechanics defined so far hold for any rigid body that has forces and moments acting upon it. The next step is to specify a model to describe how these forces and moment depend on the state variables and actuator inputs.

### 2.3.2. Thrust and drag model

In recent work about improving quadcopter trajectory tracking, a parametric model for thrust and drag is introduced that is shown to be easily identified offline while drastically improving the controller's performance [14]. The thrust $T$ and drag in the $x$ and $y$ body axes $D_x, D_y$ is modeled by the following equations:

$$T = k_\omega \sum_{i=0}^{4}\omega_i^2 + k_z w \sum_{i=0}^{4}\omega_i + k_h(u^2+v^2) \quad D_x = -k_x u \sum_{i=0}^{4}\omega_i \quad D_y = -k_y v \sum_{i=0}^{4}\omega_i$$

In these equations $\omega_i, i = 1,\ldots,4$ are the rpms of the four propellers of the drone. $u, v$ and $w$ are the velocity components along the body $x, y$ and $z$ axis respectively. In this model $k_\omega, k_z, k_h, k_x, k_y$ are the parameters to be identified by performing linear regression on flight data.

### 2.3.3. Moments

Based on this theory a model for the moments can be derived as well. The pitch moment $M_y$ is generated by a difference in thrust between the front and back propellers while the roll moment $M_x$ depends on the thrust difference between the left and right propellers.

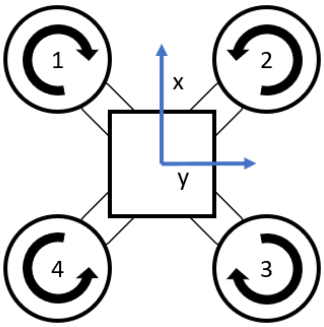



**Figure 2.4:** Propeller numbering and rotation directions

Using the propeller numbering from figure 2.4 the following equations can be derived:

$$M_x = k_p(\omega_1^2 - \omega_2^2 - \omega_3^2 + \omega_4^2)$$
$$M_y = k_q(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2)$$

Where $k_p$ and $k_q$ are new parameters to be identified which should essentially be equal to $k_\omega$ scaled by the horizontal and vertical distances from propeller to center of mass. The yaw moment is unrelated to the thrust model and depends the reaction moment from spinning up the propellers. Using the rotation direction from 2.4, the yaw moment is given by:

$$M_z = k_{r1}(-\omega_1 + \omega_2 - \omega_3 + \omega_4) + k_{r2}(-\dot{\omega_1} + \dot{\omega_2} - \dot{\omega_3} + \dot{\omega_4}) - k_{rr}r$$

### 2.3.4. Actuator model

Since the rotational speed of the propellers cannot be changed instantaneously by the motors, a first order delay actuator model can be used:

$$\dot{\omega_i} = (u_i - \omega_i)/\tau \tag{2.10}$$

Where $\tau$ is the delay constant and $u_i$ is the rpm reference command.

### 2.3.5. Combined model

Combining the models for thrust, drag, moments and actuators into one model gives the following set of dynamics:

$$\begin{aligned}
&\dot{\mathbf{x}} = \mathbf{v} \\
&\dot{\mathbf{v}} = \mathbf{g} + \frac{1}{m}R(\lambda)\begin{bmatrix} -k_x u \sum_{i=0}^{4}\omega_i \\ -k_y v \sum_{i=0}^{4}\omega_i \\ k_\omega \sum_{i=0}^{4}\omega_i^2 + k_z w \sum_{i=0}^{4}\omega_i + k_h(u^2+v^2) \end{bmatrix} \quad \text{where} \quad [u,v,w]^T = R(\lambda)^T\mathbf{v} \\
&\dot{\lambda} = Q(\lambda)\Omega \\
&I\dot{\Omega} = -\Omega \times I\Omega + \begin{bmatrix} k_p(\omega_1^2 - \omega_2^2 - \omega_3^2 + \omega_4^2) \\ k_q(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2) \\ k_{r1}(\omega_1 + \omega_2 + \omega_3 + \omega_4) + k_{r2}(\dot{\omega_1} + \dot{\omega_2} + \dot{\omega_3} + \dot{\omega_4}) - k_{rr}r \end{bmatrix} \\
&\dot{\omega_i} = (u_i - \omega_i)/\tau
\end{aligned} \tag{2.11}$$

Note that now the propeller rpms $\omega_i$ are now part of the state vector, and the control input is the rpm reference command $u_i$.

## 2.4. The Reality gap problem

In the previous section we have seen a detailed description of the quadcopters dynamics in terms of a deterministic equations. This model can be used to estimate how the quadcopter will behave, but

only up to a certain point. Especially in high speed aggressive manoeuvres, additional aerodynamic effects come into play that might be hard or even impossible to model. In our proposal we will aim to achieve exactly these types of manoeuvres using neural network controllers that are based entirely on a certain dynamical model. For this reason, the difference between the proposed model and the real life quadcopter dynamics might be detrimental to the controller's performance. In this section we will go over several approaches to solve this 'reality gap' problem.

### 2.4.1. Advanced modeling

One way to close the reality gap is to improve the model as much as possible. In recent work from the TU Delft [13], free-flight tests with the Parrot Bebop quadcopter are carried out in a large-scale wind tunnel to explore high speed aerodynamic effects. The flight data from this research shows that complex aerodynamic effects can appear that cannot be predicted by state-of-art models such as [14]. To combat these effects, a complex parametric model is identified based on prior knowledge of rotorcraft aerodynamic properties combined with data observations. The model is identified for a speed up to $14m/s$ and shows an 80% reduction of moment model residuals and a 20% reduction of force model residuals compared to a simpler model similar to equation 2.11. The strength of this research is that for their specific platform (Parrot Bebop), the results can be used to improve high speed flight controllers. However, for other platforms this requires redoing the extensive wind tunnel experiments.

### 2.4.2. Stochastic modeling

In recent research from the Robotics and Perception Group at University of Zurich a stochastic modeling approach is proposed to model aerodynamic effects [16]. In this proposal a Gaussian Process is used to predict the difference between the theoretically modeled and observed aerodynamic forces. The parameters of the Gaussian Process are obtained from flight data of a custom made quadcopter reaching speeds up to $14m/s$ and accelerations beyond $4g$. Using a Model Predictive Controller that incorporates the Gaussian process, an up to 70% reduction in trajectory tracking error is achieved. The advantage of this methods is that the model is both accurate, and simple to evaluate. However this method still requires identification of the Gaussian processes for the specific platform.

### 2.4.3. Adaptive inner loop

An alternative solution to the reality gap problem is the work of [11] which proposes an attitude controller that promises high performance nonlinear control without requiring a detailed model of the controlled vehicle. The method only uses a control effectiveness model and uses estimates of the angular accelerations to replace the rest of the model. The strength of this approach is that very little knowledge about the platform's dynamic while still achieving e high performance, disturbance rejection, and adaptiveness properties. A disadvantage is that the method relies on incrementing the actuator rpm commands which makes it difficult to deal with actuator saturations.

## 2.5. Proposal

A promising research to expand upon is the work of about G&CNets. The strength of this research is that the neural network is directly trained on the optimal state action pairs obtained from a direct method. This provides great flexibility in the choice of quadcopter model. A real life experiment has already shown that this method is competitive with a commonly used method by using a G&CNet based on a simplified 2 dimensional quadcopter model. By extending upon this work by using more accurate quadcopter models, it can be expected that even faster aggressive flight can be achieved. In our proposal we aim to achieve this by training a G&CNet that takes into account the full 6 degrees of freedom dynamics of the quadcopter including aerodynamic effects and actuator delays as described in Section 2.3.5. If this project is successful, G&CNets would provide a method to reach a quadcopter's absolute limits in terms of aggressive flight. Additionally, this research fill focus on bridging the reality-gap between simulated and real life flight performance which is a major challenge for machine learning based control methods.

# 3

# Preliminary Experiment

As a preliminary experiment, the parameters $k_x, k_y, k_\omega, k_z, k_h, k_p, k_q, k_{r1}, k_{r2}$ and $\tau$ from the previously described model (equation 2.11) will be identified for the Parrot Bebop quadcopter using the experimental setup that is described in a section 5. Several manual flight tests are conducted to collect data. Using least squares linear regression, the parameters can be found that give the best fit.

## 3.1. Actuators

In order to fit the actuator model, the rpm values from the propellers along with the reference rpm commands need to be logged. For each propeller an array of observed rpm values $\omega_i[k], k = 0, \ldots, N$ along with an array of reference rpm values $u_i[k], k = 0, \ldots, N$ is logged. Also the numerical derivative $\dot{\omega}_i$ is computed. From these arrays the regression problem is formulated as:

$$\begin{bmatrix} \dot{\omega}_i[0] \\ \vdots \\ \dot{\omega}_i[N] \end{bmatrix} = \frac{1}{\tau} \begin{bmatrix} u_i[0] - \omega_i[0] \\ \vdots \\ u_i[N] - \omega_i[N] \end{bmatrix} \quad \rightarrow \quad \frac{1}{\tau} = \begin{bmatrix} u_i[0] - \omega_i[0] \\ \vdots \\ u_i[N] - \omega_i[N] \end{bmatrix}^\dagger \begin{bmatrix} \dot{\omega}_i[0] \\ \vdots \\ \dot{\omega}_i[N] \end{bmatrix}$$

Where $\dagger$ denotes the pseudo-inverse of a matrix given by $A^\dagger = (A^T A)^{-1} A^T$. Figure 3.1 shows the regression fits for each of the propellers. This fit was performed using data from a manual flight with aggressive upwards and downwards movements such that both low and high rpm values are reached. The obtained values for $\tau$ differ slightly per propeller. An average value of $\frac{1}{\tau} = 30.9 \quad \rightarrow \tau = 0.032$ can be used in the model as approximation.

## 3.2. Thrust and Drag

For identifying the thrust and drag model, measurement of the thrust $T$ and drag $D_x, D_y$ forces are required. These measurements are obtained from the accelerometer which essentially measures specific force in the quadcopter's body frame. This specific force consists of all non-gravitational forces acting on the quadcopter divided by it's total mass. According to the model these accelerometer measurements should be equal to the drag and thrust forces, normalized by the mass:

$$\mathbf{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \frac{1}{m} \begin{bmatrix} D_x \\ D_y \\ T \end{bmatrix} = \frac{1}{m} \begin{bmatrix} -k_x u \sum_{i=0}^{4} \omega_i \\ -k_y v \sum_{i=0}^{4} \omega_i \\ k_\omega \sum_{i=0}^{4} \omega_i^2 + k_z w \sum_{i=0}^{4} \omega_i + k_h (u^2 + v^2) \end{bmatrix}$$

Since accelerometers tend to have biases a more accurate model would be:

$$\mathbf{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} = \frac{1}{m} \begin{bmatrix} -k_x u \sum_{i=0}^{4} \omega_i \\ -k_y v \sum_{i=0}^{4} \omega_i \\ k_\omega \sum_{i=0}^{4} \omega_i^2 + k_z w \sum_{i=0}^{4} \omega_i + k_h (u^2 + v^2) \end{bmatrix} + \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

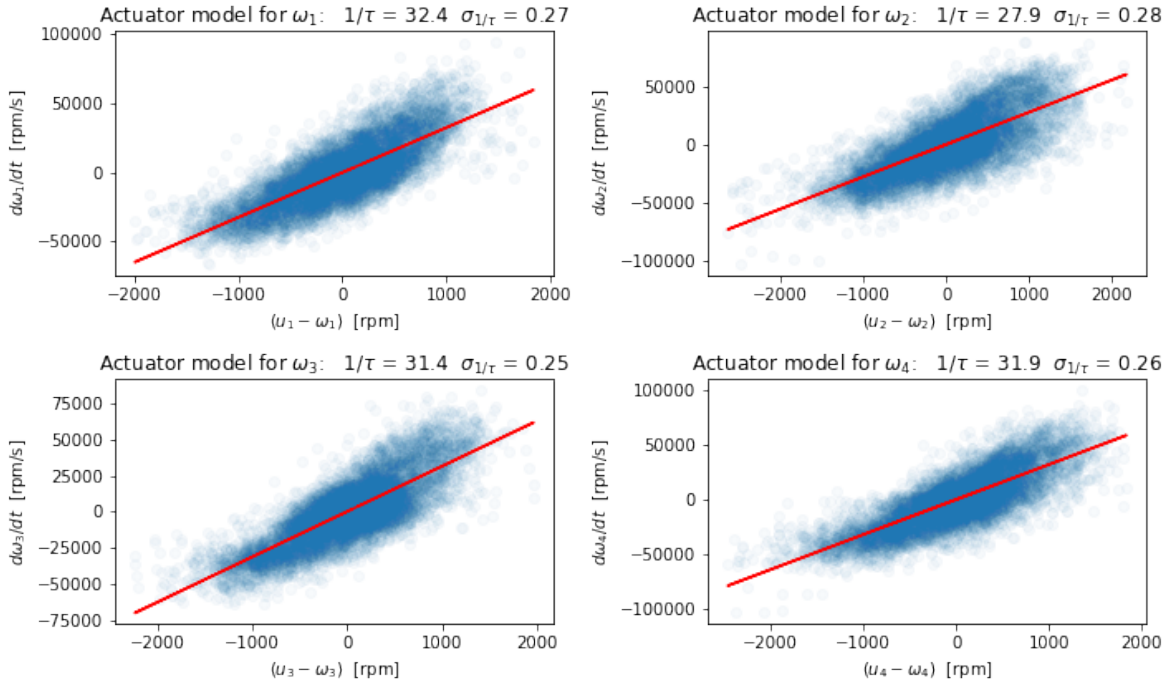



**Figure 3.1:** Regression fits of the actuator model for each of the propellers

For convenience, the following normalized parameters can be defined

$$\hat{k}_x = k_x/m, \quad \hat{k}_y = k_y/m, \quad \hat{k}_\omega = k_\omega/m, \quad \hat{k}_z = k_z/m, \quad \hat{k}_h = k_h/m,$$

such that no mass measurement is required after the parameter identification. Additional measurements required for solving the regression problems are the observed rpm values $\omega_i$ and the body velocities $u, v, w$. The drag regression problems along the x-axis can be formulated as:

$$\begin{bmatrix} a_x[0] \\ \vdots \\ a_x[N] \end{bmatrix} = \begin{bmatrix} -u[0]\sum\omega_i[0] & 1 \\ \vdots & \vdots \\ -u[N]\sum\omega_i[N] & 1 \end{bmatrix} \begin{bmatrix} \hat{k}_x \\ b_x \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \hat{k}_x \\ b_x \end{bmatrix} = \begin{bmatrix} -u[0]\sum\omega_i[0] & 1 \\ \vdots & \vdots \\ -u[N]\sum\omega_i[N] & 1 \end{bmatrix}^\dagger \begin{bmatrix} a_x[0] \\ \vdots \\ a_x[N] \end{bmatrix}$$

Similarly the regression problem for y-component of drag is given by:

$$\begin{bmatrix} a_y[0] \\ \vdots \\ a_y[N] \end{bmatrix} = \begin{bmatrix} -v[0]\sum\omega_i[0] & 1 \\ \vdots & \vdots \\ -v[N]\sum\omega_i[N] & 1 \end{bmatrix} \begin{bmatrix} \hat{k}_y \\ b_y \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \hat{k}_y \\ b_y \end{bmatrix} = \begin{bmatrix} -v[0]\sum\omega_i[0] & 1 \\ \vdots & \vdots \\ -v[N]\sum\omega_i[N] & 1 \end{bmatrix}^\dagger \begin{bmatrix} a_y[0] \\ \vdots \\ a_y[N] \end{bmatrix}$$

And finally the thrust regression problem is:

$$\begin{bmatrix} a_z[0] \\ \vdots \\ a_z[N] \end{bmatrix} = \begin{bmatrix} \sum\omega_i[0]^2 & w[0]\sum\omega_i[0] & u[0]^2+v[0]^2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \sum\omega_i[N]^2 & w[N]\sum\omega_i[N] & u[N]^2+v[N]^2 & 1 \end{bmatrix} \begin{bmatrix} \hat{k}_\omega \\ \hat{k}_z \\ \hat{k}_h \\ b_z \end{bmatrix} \quad \rightarrow$$

$$\begin{bmatrix} \hat{k}_\omega \\ \hat{k}_z \\ \hat{k}_h \\ b_z \end{bmatrix} = \begin{bmatrix} \sum\omega_i[0]^2 & w[0]\sum\omega_i[0] & u[0]^2+v[0]^2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \sum\omega_i[N]^2 & w[N]\sum\omega_i[N] & u[N]^2+v[N]^2 & 1 \end{bmatrix}^\dagger \begin{bmatrix} a_z[0] \\ \vdots \\ a_z[N] \end{bmatrix}$$

The drag model fit was performed using data from an automatic flight between 4 waypoints where high $x, y$ body velocities where reached. The thrust model was identified using the same dataset that was

used for the actuator model identification. The results of the drag and thrust model fit can be seen in table 3.1. Additionally, figure 3.2 shows a plot of the drag fit and figure 3.3 shows a comparison of the thrust model to the measured accelerations.

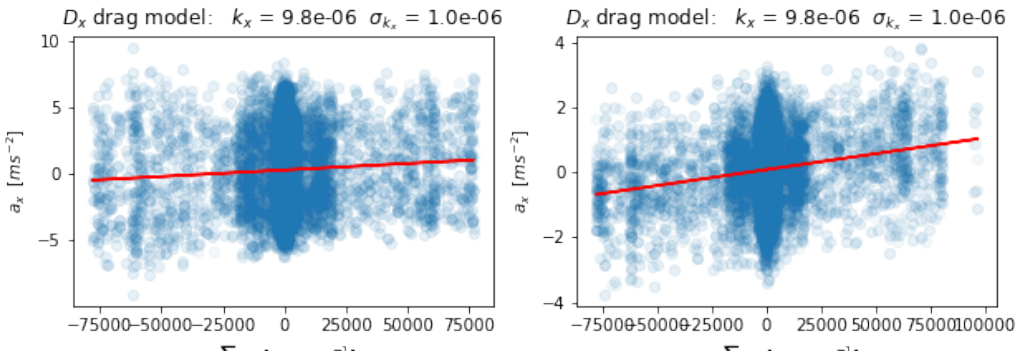



**Figure 3.2:** Regression fits of the $x, y$ drag model

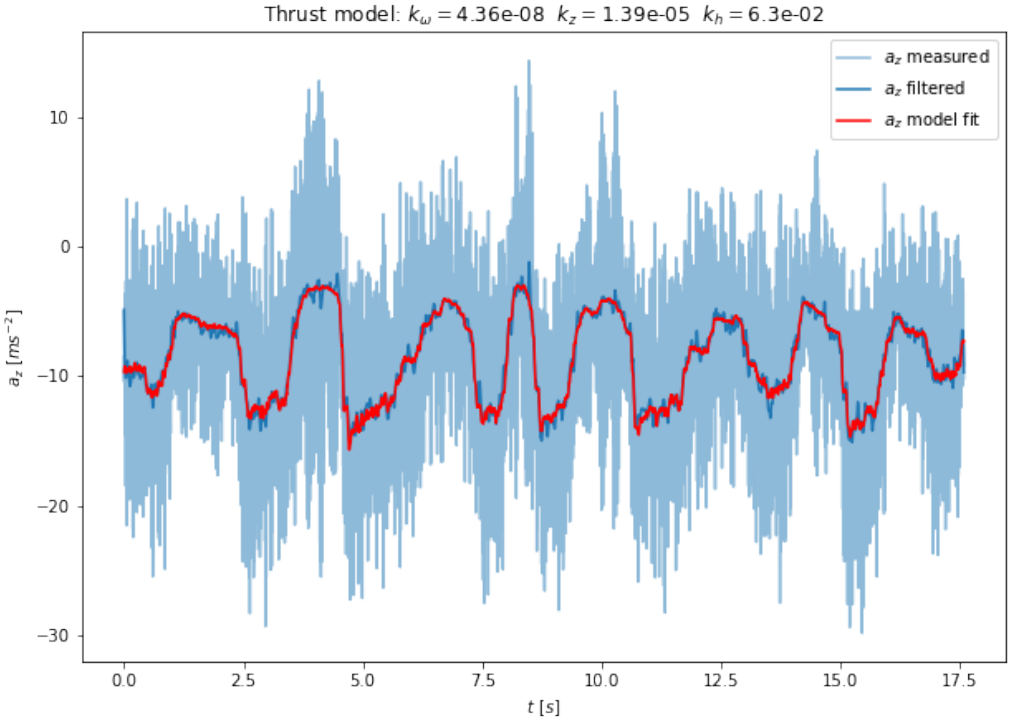



**Figure 3.3:** Thrust model compared to raw and filtered accelerometer measurements.

| Parameter | Value | Standard deviation |
|---|---|---|
| $k_x$ | 9.82e-06 | 1.e-06 |
| $k_y$ | 9.84e-06 | 4.e-07 |
| $k_\omega$ | 4.36e-08 | 9.e-10 |
| $k_z$ | 1.39e-05 | 2.e-06 |
| $k_h$ | 6.3e-02 | 5.e-02 |

**Table 3.1:** Regression fit results for the Drag and Thrust model

## 3.3. Moments

For the moment model identification, estimates of the moments $M_x, M_y, M_Z$ are calculated from gyroscope measurement $\Omega = [p, q, r]^T$ and their derivatives using the following equation:

$$\mathbf{M} = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = I\dot{\Omega} + \Omega \times I\Omega = \begin{bmatrix} I_x\dot{p} + (I_z - I_y)qr \\ I_y\dot{q} + (I_x - I_z)pr \\ I_z\dot{r} + (I_y - I_x)pq \end{bmatrix}$$

In order to reduce noise, $p, q$ and $r$ are low passed filtered at 16Hz using a zero-phase 2nd order Butterworth filter. These moments are then used for the following regression problems:

$$\begin{bmatrix} k_p \\ b_p \end{bmatrix} = \begin{bmatrix} \omega_1[0]^2 - \omega_2[0]^2 - \omega 3[0]^2 + \omega_4[0]^2 & 1 \\ \vdots & \vdots \\ \omega_1[N]^2 - \omega_2[N]^2 - \omega 3[N]^2 + \omega_4[N]^2 & 1 \end{bmatrix}^\dagger \begin{bmatrix} M_x[0] \\ \vdots \\ M_x[N] \end{bmatrix}$$

$$\begin{bmatrix} k_q \\ b_q \end{bmatrix} = \begin{bmatrix} \omega_1[0]^2 + \omega_2[0]^2 - \omega 3[0]^2 - \omega_4[0]^2 & 1 \\ \vdots & \vdots \\ \omega_1[N]^2 + \omega_2[N]^2 - \omega 3[N]^2 - \omega_4[N]^2 & 1 \end{bmatrix}^\dagger \begin{bmatrix} M_y[0] \\ \vdots \\ M_y[N] \end{bmatrix}$$

$$\begin{bmatrix} k_{r1} \\ k_{r2} \\ k_{rr} \\ b_r \end{bmatrix} = \begin{bmatrix} -\omega_1[0] + \omega_2[0] - \omega 3[0] + \omega_4[0] & -\dot{\omega_1}[0] + \dot{\omega_2}[0] - \dot{\omega_3}[0] + \dot{\omega_4}[0] & -r[0] & 1 \\ \vdots & \vdots & \vdots & \vdots \\ -\omega_1[N] + \omega_2[N] - \omega 3[N] + \omega_4[N] & -\dot{\omega_1}[N] + \dot{\omega_2}[N] - \dot{\omega_3}[N] + \dot{\omega_4}[N] & -r[N] & 1 \end{bmatrix}^\dagger \begin{bmatrix} M_y[0] \\ \vdots \\ M_y[N] \end{bmatrix}$$

The fit was performed using data from an automatic flight in hover where yaw commands where given that alternated from $0$ to $180$ degrees. The results of the fit can be seen in table 3.2. Additionally, figure 3.4 shows a plot of the pitch and roll moment fit and figure 3.5 shows a comparison of the yaw moment model to the measured moments.

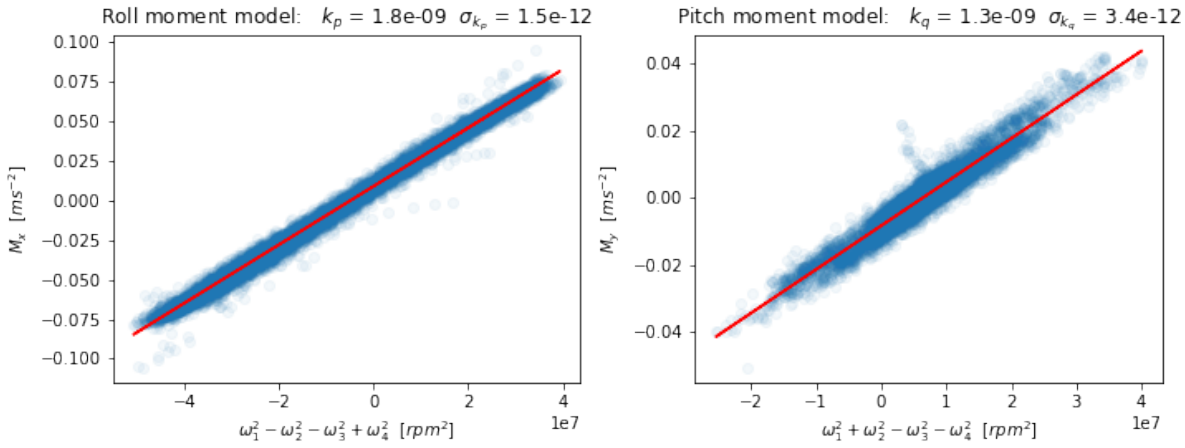



**Figure 3.4:** Caption

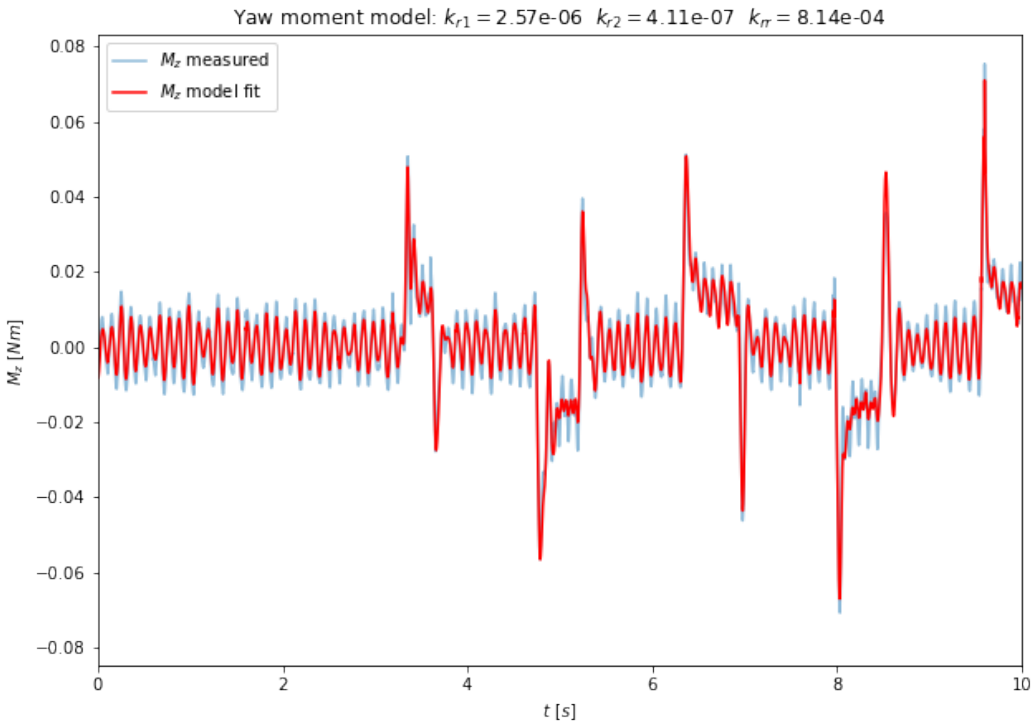



**Figure 3.5:** Yaw moment model compared to the measured moment.

| Parameter | Value | Standard deviation |
|---|---|---|
| $k_p$ | 1.84e-09 | 2.e-12 |
| $k_q$ | 1.30e-09 | 3.e-12 |
| $k_{r1}$ | 2.57e-06 | 1.e-08 |
| $k_{r2}$ | 4.11e-07 | 1.e-09 |
| $k_{rr}$ | 8.14e-04 | 2.e-05 |

**Table 3.2:** Regression fit results for moment model

# 4

# Research Questions

## 4.1. Research Questions

Based on the previous section, the research question of this proposal can be stated as the following:

> **"To what extent can time optimal flight be achieved with a real quadcopter by means of a neural network (G&CNet) trained on optimal state-action pairs providing on-board state to rpm feedback?"**

The research can be split into the following sub-questions:

**How can the reality gap problem be solved?**
Experiments will be needed to validate whether our choice of model is sufficient for the method to be stable. In the case where our method is unstable or has bad performance, the model would need to be re-evaluated. Alternatively, it could be investigated how the unmodeled effects could be compensated for without explicitly modeling them. If a solution to this problem is found, it can be expected that the real life performance of our method is comparable with the simulated performance.

**Can the network be evaluated on the quadcopter in real time?**
The work of [8] shows the successful implementation of a Neural Network that can provide real time optimal control on-board the Parrot Bebop quadcopter. But it remains to be seen how far this method can be pushed for bigger neural networks based no more complicated models. It needs to be investigated at what frequency the neural networks output can be computed and what frequency is sufficient for stable flight. The answer of this question will determine whether our method is feasible with the computational restriction of the Parrot Bebop.

**To what extend can time-optimality be achieved in terms of the power/time-optimal cost function?**
Our approach will use a cost function that is a hybrid between power- and time-optimality. The aim of this question is to find out how far we can push our method towards time-optimality ($\epsilon = 0$). The amount by which this is possible is mostly constraint by the reality gap problem. With the faster more time-optimal trajectories we can expect more influence from unmodeled effects that could de-stabilize the quadcopter. For this reason the answer to this question will depend on the solution that is found for question 1. The answer to this sub-question will give insight into the performance of our proposed method which will provide the most direct answer to the main research question.

## 4.2. Research Objective

Our research will cover the investigation of a machine learning based control method that aims to achieve time optimal flight. Therefore the main research objective of this thesis is:

> **"To achieve time optimal waypoint to waypoint flight with a quadcopter by means of a G&CNet providing on-board state to rpm feedback."**

This is an ambitious goal that we should not expect to achieve fully. However by attempting to achieve this goal, we will find the limits of our proposed method which answers our research question. The process necessary for achieving our research objective can be summarized by the next 4 sub-goals:

**Sub-goal 1: System identification and Simulation**
The first step necessary to achieve our research objective is to identify the model parameters of the quadcopter that will be used for our experiments. This will be done by gathering data from manual quadcopter flights. Based on the identified model, optimal trajectories can be generated using Hermite Simpson transcription with an NLP solver on a powerful computer. These trajectories will be used to train a neural network to approximate the optimal policy. To avoid crashes in the initial test, it seems safest to start with slow hover to hover trajectories that are power optimal. By simulating the system dynamic with the trained G&CNet controller it can be determined whether the performance is sufficient for a first test flight.

**Sub-goal 2: The first G&CNet flight tests**
When the first sub-goal is achieved, a G&CNet is successfully trained and should be ready to be deployed on our quadcopter. In order to perform the first G&CNet flight test, a practical implementation of the neural network should be developed that can run on the quadcopter's hardware. In the development process, the answer to our second research question should become clear. Once the implementation is working properly, the first flights can be performed and analyzed. By flying the hover-to-hover trajectories that the G&CNet is trained on, the performance of the method can be evaluated. Flight logs will have to be compared to the simulated trajectories to investigate any unmodeled effects.

**Sub-goal 3: Reality-Gap analysis**
This sub-goal covers the analysis of the fight data of the initial G&CNet flight tests. It can be expected that the first flight tests show results that are quite different compared to the simulations. If this is the case, it needs to be investigated what exactly this difference is and what is causing it. Ideally this goal is achieved by providing a solution to the reality gap problem in some way. This could be done by making changes to the quadcopter model, introducing some stochasticity to the problem, or by using some adaptive method as described in section 2.4. The exact method that will be used to achieve this goal is not determined yet which makes this goal the most difficult to achieve.

**Sub-goal 4: Implement improvements and final tests**
The final sub-goal involves the implementation of the improvements proposed in the previous steps. These improved G&CNets are expected to fly near time-optimal tarjectories. By iteratively changing the $\epsilon$ parameter in the optimal control problem, faster and faster G&CNets can be trained. By performing flight tests for all of these networks, the limits of the proposed method can be found. The fastest succesfully operating G&CNet can then be compared to other state-of-the-art flight controller for a final evaluation.

# 5

# Experimental Set-up

The experiments in this project can be divided into two categories. The first category consists of performing the computations and simulations necessary to obtain a G&CNet controller. The other category covers all flight test experiments that will determine the real performance of the controller. In this research we will continually switch between simulation experiments and real life flight tests in order to iteratively improve the G&CNet controller. In this section we will go over the technical details of each of our experimental setups.

### 5.0.1. Computation and Simulation

Similar to [8], the optimal control problem will be transformed into an NLP problem by using Hermite Simpson transcritption. The AMPL modelling language will be used to specify the NLP problem which can then be solved via SNOPT, an SQP NLP solver. Using AMPLPY (a python interface for the AMPL interpreter), a set of initial conditions can be sampled for which the optimal solution can be computed. From these optimal trajectories, a dataset of state-control pairs can be obtained. This dataset will be generated using parallel computing on a remote server. Using the python library PyTorch, neural networks will be trained to learn the control policy by a stochastic gradient descent method using only a portion of the dataset. The rest of the dataset can be used for validation. The performance of the trained neural network will then be investigated by simulating the closed loop system dynamics in Python. The simulated trajectory can then be compared with a trajectory from the dataset with the same initial condition. The simulation can also be modified to include external forces or moments to investigate the robustness of the method.

### 5.0.2. Flight tests

The quadcopter used in our experiment is the Parrot Bebop 1 (Figure 5.1). The on-board software has been replaced by the Paparazzi-UAV open-source autopilot project [3]. All computations will run in real-time on the Parrot P7 dual-core CPU Cortex A9 processor. Based on previous research [8] it is expected that this hardware is sufficient for our computational requirements, but experiments will have to show if there are any limitations. The Parrot Bebop has an MPU650 IMU sensor that will be used to obtain measurements of the specific force and angular velocity along the body axes. Additionally the Bebop has sensors that measure the angular velocities (in rpm) of each of the propellers, which is a necessary requirement for our control method.

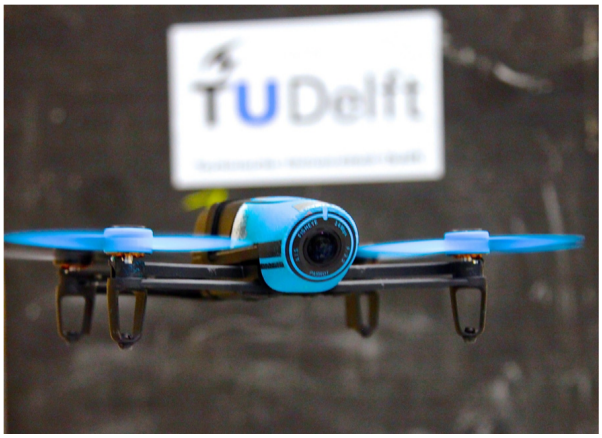



**Figure 5.1:** The Parrot Bebop 1 is used as experiment platform. The software is replaced bythe Paparazzi UAV open-source autopilot project. Image from [7].

All flight tests will be performed in The CyberZoo which is a research and test laboratory in the faculty of Aerospace Engineering at the TU Delft. This lab consists of a 10 by 10 meter area surrounded by nets with an OptiTrack motion capture system that can provide position and attitude data in real time. Our control pipeline will use OptiTrack's data in combination with the Bebop's IMU in order to obtain the most accurate state estimation possible. A diagram of the proposed control pipeline can be seen in Figure 5.2.

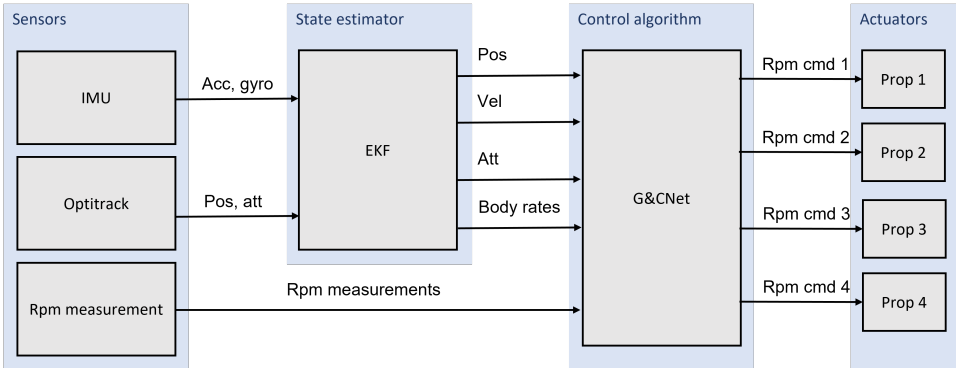



**Figure 5.2:** Proposed control pipeline: An extended kalman filter is used to fuse the OptiTrack and IMU data in order to obtain an estimate of the position, velocity, attitude and body rates. These state variables are used as an input to the G&CNet along with the rpm measurements. The outputs of the G&CNet will be directly used as rpm commands to the propellers.

Since our control method is based on the assumption that the G&CNet has access to perfect state information, the accuracy of our state estimator should be as high as possible. Inaccurate state estimates could limit the controller's performance or in the worst case even cause a crash. Additionally the accuracy of the state estimates are important for the analysis of the flight data and the identification of the model parameters.

# 6

# Project Planning

The thesis project can best be structured by following the previously described sub-goals as a guideline. The first things to be done are the initial system identification experiments and Neural network simulations along with setting up the required software. If this is achieved, the first G&CNet flight test will be performed which is the second sub-goal. The G&CNet implementation can be developed in parallel with the system identification experiments to save time. Once the first G&CNet experiments are finished, an extensive analysis is required in an attempt to solve the reality gap problem. This is the most uncertain phase of our project because the problems that could arise are still uncertain. The phase is ideally completed by making a proposal that could improve the performance from our flight test. At this point in the project it would be a perfect time to plan a midterm meeting to evaluate the proposal and see if our goals are still feasible. The research will continue by implementing the new proposals in our final experiments. These experiments would be the main results of this thesis that will answer our main research question. In the final phase of this research the results from our experiments will be worked out in the thesis report. The thesis is completed with a Green-light review followed by the final defense.

The project planning is presented in a detailed Gantt chart in figure 6.1. This chart covers all the previously discussed tasks planned over the period from April 2021 to March 2022.

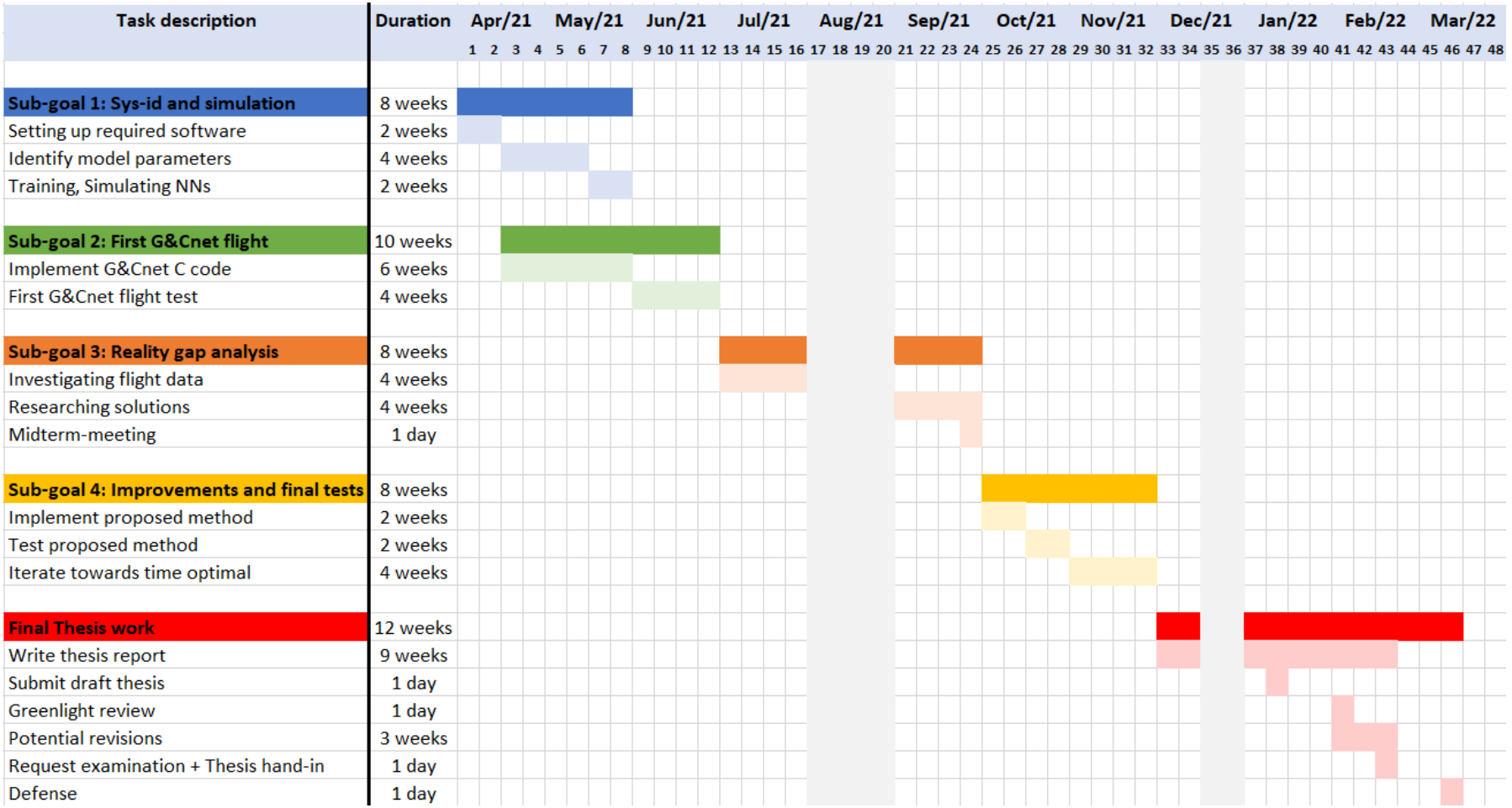



**Figure 6.1:** Gantt Chart

# 7

# Results, Outcome and Relevance

In our proposed research, we will perform flight test experiments to determine the performance of our machine learning based controller. The goal of our controller is to achieve time-optimal flight. An ideal outcome of these experiments would be a successful execution of a time optimal trajectory using our experimental setup. Such a result could be validated by direct comparison with simulated trajectories. However, as we have discussed before, the reality gap problem along with the inherent limitations of our experimental setup make 100% time optimal flight extremely challenging. A more obtainable outcome would be to outperform a previous attempt at time optimal control that used the same experimental setup like [8] or [17]. Such a comparison would demonstrate the strengths of our proposed method. On the other hand, the experiments could also provide insight into the limitations of our control method. The final outcome of our proposed research will contribute to solving the challenge of real-time optimal control. It will help decrease the gap between human and AI flight performance and push the quad-copter to its physical limits. This is especially relevant for future technologies that rely on high speed aggressive flight of UAVs.

# Part III

# Additional Results

# 8

# Extended Kalman Filter Implementation

In the flight tests of this thesis an extended kalman filter is used to fuse the OptiTrack and IMU data in order to obtain an estimate of the position, velocity, attitude and body rates. In this chapter the implementation of this filter will be explained.

## 8.1. State Transition and Observation Model

The following equations describe the state transition and observation model:

**State, inputs and measurements**

$$\mathbf{x} = (x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, l_x, l_y, l_z, l_p, l_q, l_r)$$
$$\mathbf{u} = (a_{x_m}, a_{y_m}, a_{z_m}, p_m, q_m, r_m)$$
$$\mathbf{z} = (x_m, y_m, z_m, \phi_m, \theta_m, \psi_m)$$

Where **u** is the accelerometer and gyro measurement from the IMU and **z** is the position and attitude obtained from Optitrack. $l_x, l_y, l_z, l_p, l_q, l_r$ are the biases corresponding to the accelerometer and gyroscope. The process noise is given by

$$\mathbf{w} = (w_x, w_y, w_z, w_p, w_q, w_r)$$

which is equal to the IMU noise in our case. The measurement noise is gievn by

$$\mathbf{v} = (v_x, v_y, v_z, v_\phi, v_\theta, v_\psi)$$

which is equal to the Optitrack noise in our case.

**State transition equation**

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

$$\begin{aligned}
\dot{x} &= v_x \\
\dot{y} &= v_y \\
\dot{z} &= v_z \\
\dot{v_x} &= (s_\phi s_\psi + s_\theta c_\phi c_\psi)(a_{z_m} - l_z - w_z) + (s_\phi s_\theta c_\psi - s_\psi c_\phi)(a_{y_m} - l_y - w_y) + (a_{x_m} - l_x - w_x) c_\psi c_\theta \\
\dot{v_y} &= (-s_\phi c_\psi + s_\psi s_\theta c_\phi)(a_{z_m} - l_z - w_z) + (s_\phi s_\psi s_\theta + c_\phi c_\psi)(a_{y_m} - l_y - w_y) + (a_{x_m} - l_x - w_x) s_\psi c_\theta \\
\dot{v_z} &= -(a_{x_m} - l_x - w_x) s_\theta + (a_{y_m} - l_y - w_y) s_\phi c_\theta + (a_{z_m} - l_z - w_z) c_\phi c_\theta + 9.81 \\
\dot{\phi} &= -l_p + p_m - w_p + (-l_q + q_m - w_q) s_\phi t_\theta + (-l_r + r_m - w_r) c_\phi t_\theta \\
\dot{\theta} &= (-l_q + q_m - w_q) c_\phi - (-l_r + r_m - w_r) s_\phi \\
\dot{\psi} &= \frac{(-l_q + q_m - w_q) s_\phi}{c_\theta} + \frac{(-l_r + r_m - w_r) c_\phi}{c_\theta} \\
\dot{l_x} &= 0 \quad \dot{l_y} = 0 \quad \dot{l_z} = 0 \quad \dot{l_p} = 0 \quad \dot{l_q} = 0 \quad \dot{l_r} = 0
\end{aligned}$$

With $s_x = \sin x \quad c_x = \cos x \quad t_x = \tan x$

**State observation equation**

$$\mathbf{z} = h(\mathbf{x}) + \mathbf{v}$$

$$\begin{aligned} x_m &= x + v_x \\ y_m &= y + v_y \\ z_m &= z + v_z \\ \phi_m &= \phi + v_\phi \\ \theta_m &= \theta + v_\theta \\ \psi_m &= \psi + v_\psi \end{aligned}$$

## 8.2. Algorithm

The Extended Kalman Filter will be applied to the state transition and observation model. The algorithm consists of the following steps:

**Step 0: initial values**
An initial estimate for the state $\mathbf{x}_{0,0}$ and the state covariance matrix $P_{0,0}$ needs to be chosen. Also the noise co-variance matrices $Q$ and $R$ need to be known. The Kalman filter is initialized after the fist Optitrack message is received. The initial states estimate will be obtained from the first position and attitude measurement:

$$\mathbf{x}_{0,0} = (x_m, y_m, z_m, 0, 0, 0, \phi_m, \theta_m, \psi_m, 0, 0, 0, 0, 0, 0)$$

The state covariance is initialized as

$$P_{0,0} = \mathsf{diag}(1, \ldots, 1)$$

The noise co-variance matrices are set to

$$\begin{aligned} Q &= \mathsf{diag}(0.5, 0.5, 0.5, 0.01, 0.01, 0.01) \\ R &= \mathsf{diag}(0.001, 0.001, 0.001, 0.1, 0.1, 0.1) \end{aligned}$$

**Step 1: One step ahead prediction**
Predicts the next state $\mathbf{x}_{k+1,k}$ and measurement $\mathbf{z}_{k+1,k}$ based previous state estimate $\mathbf{x}_{k1,k}$ by integrating the state transit equation

$$\begin{aligned} \mathbf{x}_{k+1,k} &= \int_{t_k}^{t_{k+1}} f(\mathbf{x}_{k,k}, \mathbf{u}_k, \mathbf{0}) dx \approx f(\mathbf{x}_{k,k}, \mathbf{u}_k, \mathbf{0})(t_{k+1} - t_k) \\ \mathbf{z}_{k+1,k} &= h(\mathbf{x}_{k+1,k}) \end{aligned}$$

**Step 2: Calculate $F_x$, $H_x$ and $G$**
The following jacobians need to be calculated

$$F_x = \frac{\partial f}{\partial \mathbf{x}}(\mathbf{x}_{k+1,k}, \mathbf{u}_k, \mathbf{0}) \quad H_x = \frac{\partial h}{\partial \mathbf{x}}(\mathbf{x}_{k+1,k}) \quad L = \frac{\partial f}{\partial \mathbf{w}}(\mathbf{x}_{k+1,k}, \mathbf{u}_k, \mathbf{0})$$

The expression for the jacobians are obtained using SymPy's jacobian method on the functions $f$ and $g$ expressed symbolically.

**Step 3: Discretize $F_x$ and $G$**
The matrices $F_x$ and $G$ are transformed to the discrete form using:

$$\Phi = e^{F_x \Delta t} \approx I + F_x \Delta t \quad \Gamma = \int_0^{\Delta t} e^{A\tau} L d\tau \approx L \Delta t$$

Where $\Delta t = t_{k+1} - t_k$

**Step 4: Predict covariance matrix**
A prediction of the covariance matrix is made by calculating

$$P_{k+1,k} = \Phi P_{k,k} \Phi^T + \Gamma Q \Gamma^T;$$

**Step 5: Kalman gain calculation**
Calculate Kalman gain

$$K = P_{k+1,k} H_x^T (H_x P_{k+1,k} H_x^T + R)^{-1}$$

**Step 6: Measurement update**
The estimated state is calculated based on the obtained measurement and the predicted state:

$$\mathbf{x}_{k+1,k+1} = \mathbf{x}_{k+1,k} + K(z_k - z_{k+1,k})$$

**Step 7: Covariance matrix update**
The estimated covariance matrix is calculated

$$P_{k+1,k+1} = (I - KH_x) P_{k+1,k}$$

However, for numericall stability the following is used:

$$P_{k+1,k+1} = (I_n - KH_x) P_{k+1,k} (I_n - KH_x)^T + KRK^T$$

With these steps, an estimate for each of the state variables will be obtained

$$\mathbf{x} = (x, y, z, v_x, v_y, v_z, \phi, \theta, \psi, l_x, l_y, l_z, l_p, l_q, l_r)$$

# 9

# Time Optimal Control

In the main results of this thesis only power optimal flight is considered. The reason for this choice is that power optimal trajectories tend to be slower and less aggressive making them easier to imitate or the G&CNets. Additionally, we expect that power optimal G&CNets are more robust to modeling errors compared to the time optimal variant. In this section we will investigate this claim in order to find out how far our G&CNet method can be pushed towards true time optimal control. A full investigation of methods to obtain 'true' time optimal control is out of scope for this project. For this reason, this chapter can be viewed as a potential starting point for further research.

We will train a G&CNet that minimizes a hybrid cost function taking into account both time- and power optimally given by:

$$J(\mathbf{x}, \mathbf{u}, T) = 0.5T + 0.5 \int_0^T ||\mathbf{u}(t)||^2 dt$$

Using similar initial conditions and final state constrains as in the scientific paper, the network will be trained to learn the '50% time optimal' state feedback. Unlike the G&CNet from the paper, here we use the following rpm limits.

$$\omega_{min} = 3000 \quad \omega_{max} = 9800$$

The controller switches to the next target waypoint once the drone is within 0.5m from the current target. Additionally we are also using the adaptive moment feedback method with the same control pipeline as described in the paper. In figure 9.1 a top down and sideways view can be seen of the performed flight test. After 2 successful laps, the drone passe through waypoint 2, loses altitude and crashes. In figure 9.2, the pitch and roll angles during this flight are plotted. Here it can be seen that around $t = 8.0s$, the drone starts to oscillate and destabilize in the roll axis which is followed by an increasing pitch angle as the drone crashes down. In Figure 9.3, the actuator rpms are plotted as well as the reference rpm commands. As expected it can be seen that the actuators are often saturated during the flight. Towards the end it can be seen that all actuators are saturated. Additionally it can be noted the rpms sometimes overshoot the actuator limits which should not be possible according to the model. It is hard to say what exactly is causing the drone to crash in this flight test, but it is clear that by increasing time optimality we are somehow losing robustness. There are various methods that could be used to mitigate these instability issues. A simple solution would be to switch the target waypoint earlier such that the controller would not have to make any aggressive adjustments to approach the waypoint more closely. This will however decrease the controller's accuracy. Alternatively, the robustness could also be increased by extending our adaptive method to also compensate for unmodeled effects in the actuator, thrust and drag model.

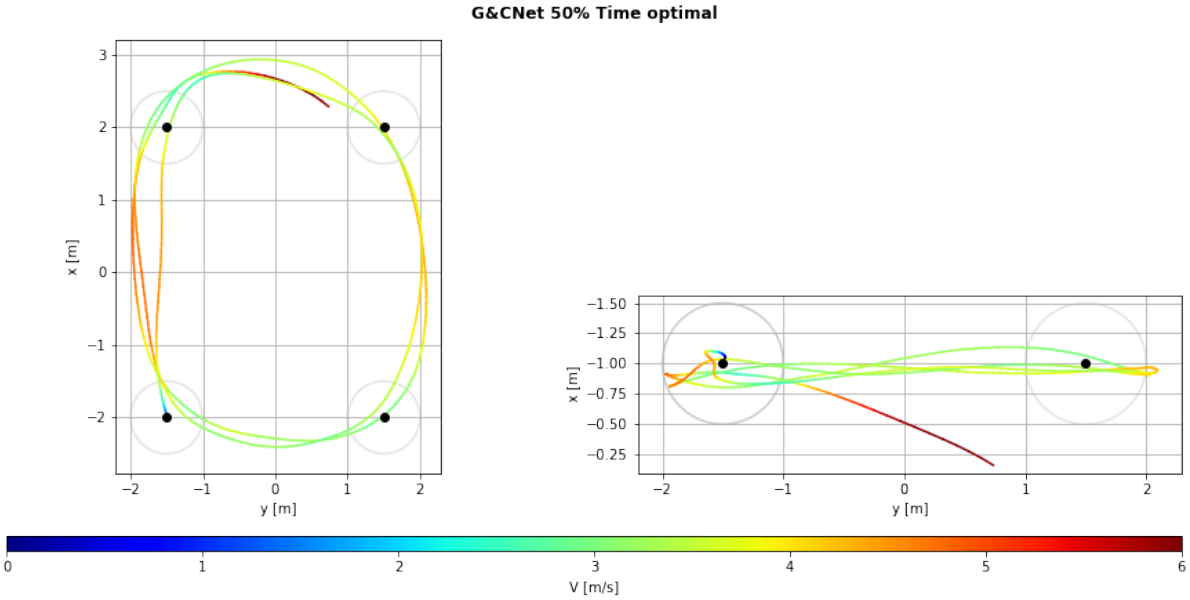



**Figure 9.1:** 50% time optimal G&CNet flying in a 4x3m track. After 2 successful laps, the drone Crashes around waypoint 3.

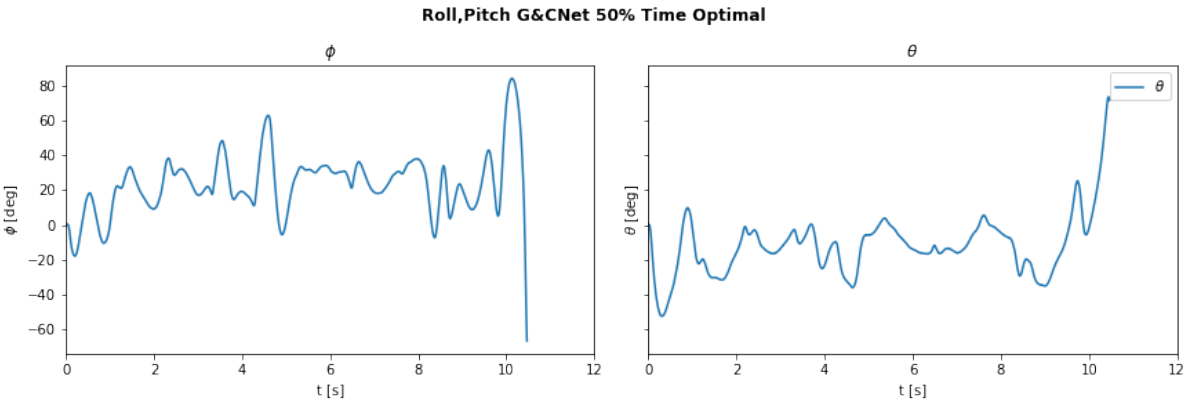



**Figure 9.2:** Roll and Pitch angles during 50% time optimal flight

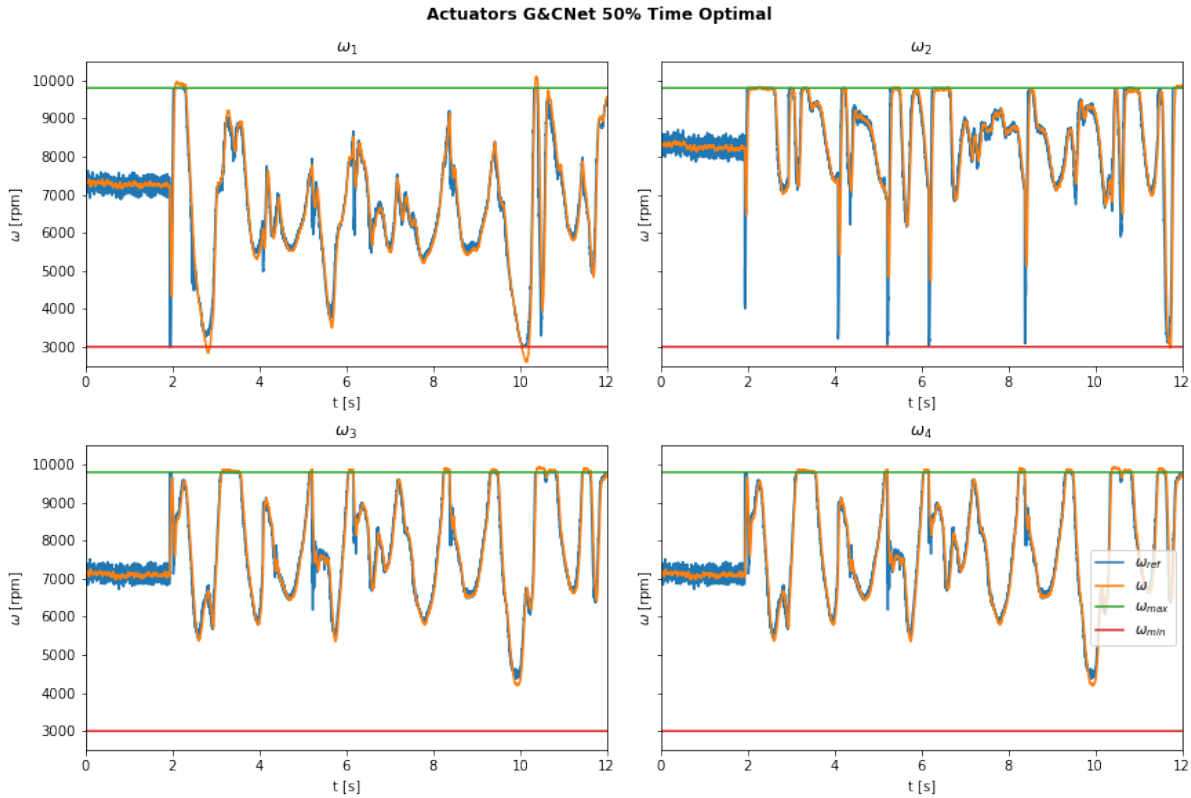



**Figure 9.3:** Angular velocities (in rpm) of each of the 4 actuator during 50% time optimal flight

# References


[1] Ilona van den Brink and Christophe de Wagter. *Mavlab World Champion in Airr Autonomous Drone Race 2019*. Dec. 2019. URL: https://www.tudelft.nl/en/2019/tu-delft/mavlab-world-champion-in-airr-autonomous-drone-race-2019/.

[2] A.E. Bryson. "Optimal control-1950 to 1985". In: *IEEE Control Systems Magazine* 16.3 (1996), pp. 26–33. DOI: 10.1109/37.506395.

[3] Balazs Gati. "Open source autopilot for academic research - The Paparazzi system". In: *2013 American Control Conference*. IEEE, June 2013. DOI: 10.1109/acc.2013.6580045. URL: https://doi.org/10.1109/acc.2013.6580045.

[4] Mostafa Hassanalian and Abdessattar Abdelkefi. "Classifications, applications, and design challenges of drones: A review". In: *Progress in Aerospace Sciences* 91 (2017), pp. 99–131.

[5] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[6] Elia Kaufmann et al. "Deep Drone Acrobatics". In: (2020). DOI: 10.48550/ARXIV.2006.05768. URL: https://arxiv.org/abs/2006.05768.

[7] S. Li et al. "Autonomous drone race: A computationally efficient vision-based navigation and control strategy". In: (2018). DOI: 10.48550/ARXIV.1809.05958. URL: https://arxiv.org/abs/1809.05958.

[8] Shuo Li et al. "Aggressive Online Control of a Quadrotor via Deep Network Representations of Optimality Principles". In: *CoRR* abs/1912.07067 (2019). arXiv: 1912.07067. URL: http://arxiv.org/abs/1912.07067.

[9] O. L. Mangasarian. "Sufficient Conditions for the Optimal Control of Nonlinear Systems". In: *SIAM Journal on Control* 4.1 (1966), pp. 139–152. DOI: 10.1137/0304013. eprint: https://doi.org/10.1137/0304013. URL: https://doi.org/10.1137/0304013.

[10] Carlos Sánchez-Sánchez and Dario Izzo. *Real-time optimal control via Deep Neural Networks: study on landing problems*. 2016. DOI: 10.48550/ARXIV.1610.08668. URL: https://arxiv.org/abs/1610.08668.

[11] Ewoud J. J. Smeur, Qiping Chu, and Guido C. H. E. de Croon. "Adaptive Incremental Nonlinear Dynamic Inversion for Attitude Control of Micro Air Vehicles". In: *Journal of Guidance, Control, and Dynamics* 39.3 (Mar. 2016), pp. 450–461. DOI: 10.2514/1.g001490. URL: https://doi.org/10.2514/1.g001490.

[12] Yunlong Song et al. *Autonomous Drone Racing with Deep Reinforcement Learning*. 2021. DOI: 10.48550/ARXIV.2103.08624. URL: https://arxiv.org/abs/2103.08624.

[13] Sihao Sun, Coen C. de Visser, and Qiping Chu. "Quadrotor Gray-Box Model Identification from High-Speed Flight Data". In: *Journal of Aircraft* 56.2 (Mar. 2019), pp. 645–661. DOI: 10.2514/1.c035135. URL: https://doi.org/10.2514/1.c035135.

[14] James Svacha, Kartik Mohta, and Vijay R. Kumar. "Improving quadrotor trajectory tracking by compensating for aerodynamic effects". In: *2017 International Conference on Unmanned Aircraft Systems (ICUAS)* (2017), pp. 860–866.

[15] Dharmesh Tailor and Dario Izzo. *Learning the optimal state-feedback via supervised imitation learning*. 2019. DOI: 10.48550/ARXIV.1901.02369. URL: https://arxiv.org/abs/1901.02369.

[16] Guillem Torrente et al. "Data-Driven MPC for Quadrotors". In: *IEEE Robotics and Automation Letters* (2021).

[17] Jelle Westenberger. “Time-Optimal Control for Tiny Quadcopters”. An optional note. MA thesis. TU Delft Aerospace Engineering, Mar. 2021.