Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for
Dynamic Systems and Control
IDSC
Institut für Dynamische Systeme
und Regelungstechnik

Julian Förster

System Identification of the Crazyflie 2.0 Nano Quadrocopter

Bachelor Thesis

Institute for Dynamic Systems and Control
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

Michael Hamer
Prof. Dr. Raffaello D'Andrea

August 2015

IDSC-RD-MDH-04

Preface

The Crazyflie 2.0 (Crazyflie) is the second generation of a nano quadrocopter, developed by the Swedish company Bitcraze AB. It is meant to be a development platform and is therefore open source as well as open hardware.
Currently some research is being undertaken on point-to-point distance measurements using ultra-wideband (UWB) sensors at the Institute for Dynamic Systems and Control (IDSC). Due to its light weight and small size, the Crazyflie was chosen as a demonstration tool for this technology. In the future these quadrocopters should be used to demonstrate the achievements of the research mentioned above at as well as outside of the ETH.
For this purpose the estimation and control algorithms that are already included in Bitcraze's firmware for the Crazyflie have to be replaced. The goal of this bachelor thesis, documented in this report, was to determine the system parameters that will be needed in order to design the new estimation and control algorithms. These are all moments of inertia of the quadrocopter, thrust maps and the transfer function of the motors as well as the drag coefficients of the Crazyflie.

Contents

Abstract

The physical parameters of the Crazyflie 2.0 nano quadrocopter and the experiments that were used to determine them are presented. Firstly, to measure the coefficients of the inertia matrix, the relationship between the moment of inertia and the period of a pendulum was made use of. Secondly, a set of motor parameter mappings between motor input command, produced thrust and torque and the rotor's angular velocities was determined using a force/torque sensor and a laser tachometer. In addition, a transfer function for the motors was identified by applying sinusoidal inputs to the motors. Thirdly, the quadrocopter's drag coefficients that characterize the force acting on the rotating propellers when the quadrocopter is moving in air were determined. For this, air was blown onto the Crazyflie while it was mounted to the force/torque sensor. Finally, the experimental methods for determining the inertia matrix and the motor parameter mappings were verified using appropriate experiments.

Nomenclature

Symbols

$\boldsymbol{I}$	Inertia Matrix	$[\mathrm{kg} \cdot \mathrm{m}^2]$
$\boldsymbol{u}$	Unit Vector	$[-]$
W	Energy	$[\mathrm{J}]$
M	Torque	$[\mathrm{N} \cdot \mathrm{m}]$
g	Earth's gravity	$[\mathrm{N}/\mathrm{kg}]$
t	Time	$[\mathrm{s}]$
φ	Angular position of an axis	$[\mathrm{rad}]$
r	Radius, e.g. of an axis	$[\mathrm{m}]$
d	Diameter	$[\mathrm{m}]$
Ω	Angular velocity	$[\mathrm{rad}/\mathrm{s}]$
N	Number of data points	$[-]$
T	Period	$[s]$
m	Mass	$[kg]$
f	Thrust	$[N]$
$\dot{\theta}$	Angular velocity	$[\mathrm{rad}/\mathrm{s}]$
cmd	Motor input command	$[-]$
RPM	Revolutions per minute	$[1/\mathrm{min}]$
τ	Torque	$[\mathrm{N} \cdot \mathrm{m}]$
R	Rotation matrix	$[-]$
x	Position	$[\mathrm{m}]$

Indicies

x, y, z	Axis of Reference for e.g. a moment of inertia
a	With respect to an axis (e.g. the rotation axis of the swing experiment)
fr	Friction
0	Initial condition
exp	Experimental setup or experimental data
s	Sampling, e.g. T_{s} is the sampling time
pm	Point mass

CF	Crazyflie
TB	Test body
T	Transient

Acronyms and Abbreviations

ETH	Eidgenössische Technische Hochschule
IDSC	Institute for Dynamic Systems and Control
UWB	Ultra Wideband
PMMA	Polymethylmethacrylat (acrylic glass)
eq.	Equation
FFT	Fast Fourier transform
RTT	Round trip time
FMA	Flying machine arena

Chapter 1

Introduction

Drones and robots are gaining more and more importance in everyday life. They are used for search and rescue missions [12], photography from viewpoints that are difficult to reach for humans [14] and countless other applications. Many of them already fly autonomously, however there are still limits making human intervention necessary on occasion. Indoors, autonomous operation in particular is possible but often dependent on expensive and/or complex absolute positioning systems (e.g. computer vision systems) that observe the current position and orientation of moving modules, see [10].

At the IDSC research on a cost efficient and robust alternative to common indoor localization systems is currently being done: using ultra-wideband (UWB) radios [13]. With one radio mounted on each module of a robotic system and a certain number of so-called anchor radios distributed in the space of interest, a measurement of the inter-modular distance as well as the determination of the absolute position of the system becomes possible. To facilitate the research on such a system and to be able to demonstrate its results, a suitable platform was sought. The Crazyflie, developed by the company Bitcraze in Sweden, was chosen from several small sized quadrocopters due to its particularly small size and weight, good flight performance and most importantly because the system is open source and open hardware and therefore easily customizable.

The customization of the Crazyflie for the purpose of a demonstration platform will include replacing both the estimator and flight controller that are already included in the quadrocopter's firmware [7], with a new model based estimator (extended Kalman filter) and controller. The goal of this thesis is to determine all physical parameters of the Crazyflie that will be necessary to implement these new algorithms. That is the drag coefficients that will be used by the estimator to estimate the quadrocopter's velocity based on accelerometer data as well as the inertia matrix and parameters characterizing the motors. These two latter parameters are used by the controller to translate actuating variables for translational and angular velocity into input commands for the Crazyflie's motors.

This thesis is subdivided into three main chapters. Chapter 2 concentrates on the determination of the inertia matrix, Chapter 3 presents everything in connection with the motors and Chapter 4 reveals the details on ascertaining the drag coefficients of the quadrocopter. Every chapter includes details on the application of the parameter, theoretical background, considerations regarding

the choice of experiments, experimental setup and procedure, data analysis, results and the verification of the methods.
Readers that are mainly interested in the numeric results of the experiments can find a sheet summarizing them in Appendix A.

Chapter 2

Inertia

This chapter begins with an explanation of what the inertial properties of the Crazyflie will be used for. Then the theory, that was used as well as the experimental methods that were developed to determine said matrix are described and finally the results are discussed.

2.1 Application

The inertia matrix mainly influences angular accelerations of a body. For this reason it is used by a quadrocopter's on-board controller to calculate the thrust that is necessary in order to achieve a desired angular velocity about a given axis.
As already mentioned earlier, the Crazyflie's estimator and controller is going to be replaced by a more effective and efficient one for the IDSC's desired applications. In order to have a good starting point for the design process, the Crazyflie's inertia matrix, with respect to the mass center, had to be determined as accurately as possible.

2.2 Theoretical Background

The inertia properties of a body can be completely characterized by its inertia matrix [8]:

$$\boldsymbol{I} = \begin{pmatrix} I_{\mathrm{xx}} & -I_{\mathrm{xy}} & -I_{\mathrm{xz}} \\ -I_{\mathrm{yx}} & I_{\mathrm{yy}} & -I_{\mathrm{yz}} \\ -I_{\mathrm{zx}} & -I_{\mathrm{zy}} & I_{\mathrm{zz}} \end{pmatrix} . \tag{2.1}$$

In this matrix, I_{ii} are the body's moments of inertia with respect to the bodyframe axes x, y and z as defined in Figure 2.4 and I_{ij} are the products of inertia. As the inertia matrix is symmetric, that is $I_{\mathrm{ij}} = I_{\mathrm{ji}}$, it has only six *independent* parameters.
All proposed experimental procedures that are described below have something in common: they only allow one to determine a body's moment of inertia around one axis at the time instead of the whole matrix at once. However it is possible to

then determine the whole inertia matrix by solving a linear system of equations that is based on the following equation [8]:

$$I_{\mathrm{Oa}} = I_{\mathrm{xx}} u_{\mathrm{x}}^2 + I_{\mathrm{yy}} u_{\mathrm{y}}^2 + I_{\mathrm{zz}} u_{\mathrm{z}}^2 - 2I_{\mathrm{xy}} u_{\mathrm{x}} u_{\mathrm{y}} - 2I_{\mathrm{yz}} u_{\mathrm{y}} u_{\mathrm{z}} - 2I_{\mathrm{zx}} u_{\mathrm{z}} u_{\mathrm{x}}. \tag{2.2}$$

In this equation I_{Oa} is the moment of inertia of a body about an arbitrary axis, u_{x}, u_{y} and u_{z} are the components of the unit vector that indicates the direction of axis Oa and I_{ij} are the moments and products of inertia of the body. Now measuring the moments of inertia with respect to six different axes and plugging the results as well as the directions of the axes into equation (2.2) leads to a system of six linear equations with six unknowns: the independent components of the body's inertia matrix.
In order to get the inertia matrix with respect to the mass center the measured moments of inertia I_{Oa} all have to be with respect to the mass center. However some of the methods presented below don't allow these to be determined directly. Instead, the parallel axis theorem (eq. (2.3)) [8] can be used to shift the results into the center of mass:

$$\begin{aligned} I_{\mathrm{xx}} &= I_{\mathrm{xx,G}} + m(y_{\mathrm{G}}^2 + z_{\mathrm{G}}^2), \\ I_{\mathrm{yy}} &= I_{\mathrm{yy,G}} + m(x_{\mathrm{G}}^2 + z_{\mathrm{G}}^2), \\ I_{\mathrm{zz}} &= I_{\mathrm{zz,G}} + m(x_{\mathrm{G}}^2 + y_{\mathrm{G}}^2). \end{aligned} \tag{2.3}$$

This theorem allows the calculation of the moment of inertia $I_{\mathrm{ii,G}}$ with respect to an axis that passes the mass center using the moment of inertia I_{ii} with respect to a parallel axis that passes an arbitrary point and the distance between these two axes.
To sum this up, the idea is to measure the Crazyflie's moments of inertia with respect to six different axes that pass arbitrary points and to use the parallel axis theorem as well as the formula to calculate the moment of inertia around an arbitrary axis in order to obtain a linear system of equations. The solutions of this linear system of equations are the six independent components of the inertia matrix.

2.3 Design Considerations

When designing the experiments needed to determine the Crazyflie's inertia matrix, three main criteria had to be taken into account.
Firstly, a body's moments and products of inertia only show their effects when the body is rotating in an accelerated way. You can see this for example in the equation of motion of a pendulum (eq. (2.8)). As long as the angular acceleration $\ddot{\varphi}(t)$ is zero the inertia matrix $\boldsymbol{I}$ does not appear in the equation.
Secondly and as can be shown using the parallel axis theorem (eq. (2.3)) the moments of inertia of a body depend on the axis of the rotation. For this reason the axis of rotation had to be easy to determine.
And finally the motion of the Crazyflie had to be observed during the experiment.
Based on these requirements several experiment designs were developed, considered and some of them rejected. In the following, the most important ones are presented.

2.3.1 Abrupt Stopping

The idea behind this approach was to make use of the kinetic energy W_{kin} that a rotating body has. W_{kin}, which can be calculated using equation (2.4), depends on the body's moment of inertia I_{a} with respect to the rotation axis but also on it's angular velocity ω.

$$W_{\text{kin}} = \frac{1}{2} I_{\text{a}} \omega^2 \tag{2.4}$$

The experiment setup for this design would consist of an axis on which the Crazyflie can be mounted, a motor, an encoder and a force/torque sensor. The Crazyflie is mounted on an axis rotated by the motor with at a constant angular velocity. Then the motor is used to stop the Crazyflie's motion abruptly. During the braking process torques around the spinning axis are detected with the force/torque sensor. From the measured torques the Crazyflie's moment of inertia with respect to the mounting axis can be calculated, taking the braking time into account.
This method was rejected due to its complicated setup and the complex data analysis that would be necessary after the experiments.

2.3.2 Accelerated Rotation

In contrast to the method described in the above section, this method makes use of the effects of the Crazyflie's inertia that show up when the quadrocopter is accelerated around one axis instead of decelerated.
The experiment setup for this experiment consists of a weight, an encoder and again an axis for the Crazyflie. The Crazyflie is mounted to the axis that - this time - has to be pivoted as frictionlessly as possible. The same axis is used to wind up a filament with the weight m_{weight} fixed to its loose end. With a known radius r of the axis, the accelerating moment $m_{\text{weight}} g r$ in equation (2.5) can be calculated easily. Logging the motion and comparing it to the solution of the equation of motion (eq. (2.5)) for this problem allows one to determine the Crazyflie's moment of inertia with respect to the mounting axis.

$$I_{\text{a}} \ddot{\varphi}(t) = m_{\text{weight}} g r - M_{\text{fr}} \dot{\varphi}(t) \tag{2.5}$$

Here, I_{a} is the moment of inertia of all bodies that participate in the spinning motion, g is the earth's gravity and M_{fr} is a constant friction coefficient that has to be determined experimentally. Solving equation (2.5) according to the rules for constant coefficient differential equations and assuming $\varphi(0) = 0$ and $\dot{\varphi}(0) = 0$ leads to the following result, the derivation of which can be found in Appendix B.2.

$$\varphi(t) = \frac{m_{\text{weight}} g r I_{\text{a}}}{M_{fr}^2} \left(e^{-\frac{M_{fr}}{I_{\text{a}}} t} - 1 \right) + \frac{m_{\text{weight}} g r}{M_{fr}} t \tag{2.6}$$

$$\dot{\varphi}(t) = \frac{m_{\text{weight}} g r}{M_{fr}} \left(1 - e^{-\frac{M_{fr}}{I_{\text{a}}} t} \right) \tag{2.7}$$

Interpreting equation (2.7) leads to the conclusion that the dropping mass will accelerate until the axis' angular velocity converges to the steady-state speed $\dot{\varphi}(t \to \infty) = \frac{m_{\text{weight}} g r}{M_{fr}}$. Measuring this velocity using the encoder leads to the desired moment of inertia.
This idea was rejected because it would have required a complicated mount to attach the Crazyflie to the axis.

2.3.3 Harmonic Swinging

The third idea is based on the principle of harmonic swinging. The period of the swinging motion of a pendulum depends on the moment of inertia of the pendulum with respect to the mounting axis. As a consequence when mounting the Crazyflie to an axis, deflecting it and observing its swinging motion, it becomes possible to calculate its moment of inertia with respect to the mounting axis.

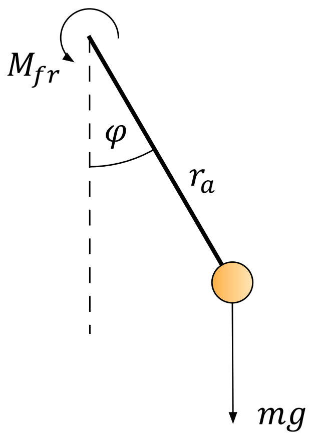


Figure 2.1: Schematic of a pendulum

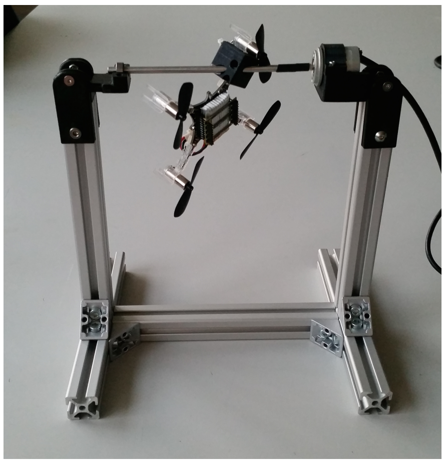
The equation of motion governing the swinging motion is

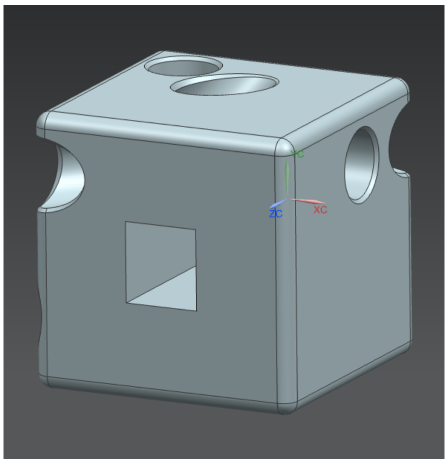
$$I_{\text{a}}\ddot{\varphi}(t) = -mgr \cdot \sin(\varphi(t)) - M_{fr}\dot{\varphi}(t) \tag{2.8}$$

A schematic of a pendulum is depicted in Figure 2.1. Due to its simplicity and good realizability this method was chosen to determine the inertia matrix of the Crazyflie.
In the following Section 2.4 the chosen harmonic swinging experiment is explained in more detail.

2.4 Experiment

This section presents the experiment that was used to determine the inertia matrix, that is experiment design, preparation, experimental procedure and the data analysis that was done after the experiment.


(a) Crazyflie Pendulum



(b) Mounting device that allows to mount the Crazyflie to the axis in six different orientations.

Figure 2.2: Harmonic Swinging Experiment Setup

2.4.1 Experiment Design

Figure 2.2a shows the experiment setup. The frame of the pendulum that was built for the Crazyflie consists of standard aluminium profiles. The axis that is used to mount the Crazyflie has a diameter d of 4 mm and is pivoted on one side using a ball bearing and on the other side with an encoder. The connection between axis and encoder is established with a shrink hose to achieve centering between mounting axis and encoder axis. Custom designed and 3D printed parts are used to keep the ball bearing and the encoder in place.

This setup allows the determination of the moment of inertia of a body mounted to the axis with respect to the rotation axis. However as was already mentioned in Section 2.2, measurements for six different axes are needed in order to determine the whole inertia matrix. So in order to be able to mount the Crazyflie in different orientations and at the same time to the rotational axis, a special mounting device (mounting cube) was designed and 3D printed. This cube is depicted in Figure 2.2b. It is attached to the Crazyflie by removing one motor including motor mount from the quadrocopter, sliding the cube onto the motor mount and reattaching everything to the Crazyflie.

As it can be seen in Figure 2.2b the cube has four holes to hold the axis. Consequently it is already possible to mount the Crazyflie in four different orientations when the cube is mounted to one arm of the quadrocopter. By also mounting the cube to a neighboring arm, it is possible to get two more orientations. All orientations are depicted in Figure 2.3. In addition, the body frame system that was introduced for the Crazyflie is presented in Figure 2.4.

The radial encoder data that reflects the Crazyflie's swinging motion was recorded using a rotary magnetic shaft encoder by RLS (serial number 35R684) and an RLS E201-9S USB encoder interface. To read out the data with a computer a Python script was implemented (see C.1.1). However as minor inconsistencies could be observed in the data when using this script, another script implemented

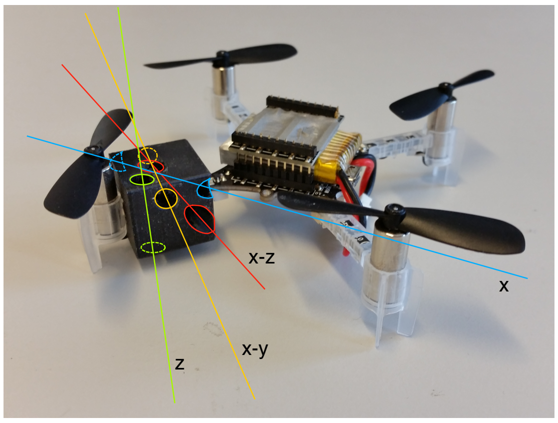


Figure 2.3: Shows how the cube can be used to mount the Crazyflie so that it rotates about the depicted body frame axes. By then mounting the cube to a neighboring arm of the Crazyflie, another two axes of rotation are possible: the y axis instead of the depicted x axis and the y-z axis instead of the depicted x-z axis. With ”x-y axis”, an axis that lies in the x-y plane and has an angle of 45° to the x and the y axes is meant.

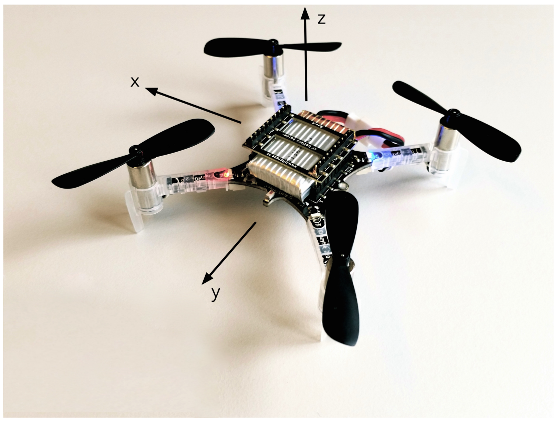


Figure 2.4: Crazyflie together with the body frame that was defined. The x axis corresponds to the pitch, y to the roll and z to the yaw axis of the quadrocopter. This coordinate system is referred to throughout the whole report.

by Michael Hamer was used to record the data that was finally used.

2.4.2 Preparation

Equation of Motion

Recall the pendulum's equation of motion which was already introduced earlier (eq. (2.8))

$$I_a\ddot{\varphi}(t) = -mgr \cdot \sin\left(\varphi(t)\right) - M_{fr}\dot{\varphi}(t). \tag{2.9}$$

This equation can be linearized using the small-angle assumption $\sin(\varphi) \approx \varphi$ for small φ. The linearized equation is

$$I_a\ddot{\varphi}(t) = -mgr \cdot \varphi(t) - M_{fr}\dot{\varphi}(t). \tag{2.10}$$

Solving this equation analytically leads to

$$\varphi(t) = \sqrt{\varphi_0^2 + \left(\frac{\Omega_0 + \delta\varphi_0}{\omega}\right)^2} \cdot e^{-\delta t} \cdot \cos\left(\omega t - \arctan\left(\frac{\Omega_0 + \delta\varphi_0}{\omega\varphi_0}\right)\right) \tag{2.11}$$

where $\varphi(t)$ the deflection angle of the Crazyflie, φ_0 the initial deflection, Ω_0 the initial angular velocity, $\delta = \frac{M_{fr}}{2I_a}$, $\omega_0 = \sqrt{\frac{mgr}{I_a}}$ and $\omega = \sqrt{\omega_0^2 - \delta^2}$. The derivation of this solution can be found in Appendix B.1.

Center of Mass

The Crazyflie's center of mass needs to be known for two calculations: the first is fitting the solution of the equation of motion (eq. (2.11)) to the recorded swinging motion data. The second is calculating the moments of inertia with respect to an axis that passes the center of mass using the parallel axis theorem (eq. (2.3)).
For simplicity, it was assumed that the position of the mass center in the x-y plane matches the geometrical center of the Crazyflie (intersection between straights connecting opposite rotor axes). This assumption makes sense because to achieve an optimal flight performance, a quadrocopter's mass center should be as close to the geometrical center as possible.

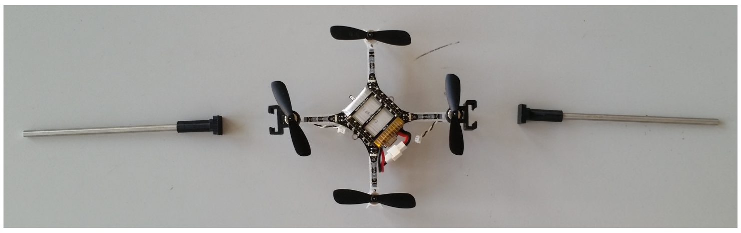


Figure 2.5: Setup that can be used to determine the z position of the Crazyflie's center of mass.

The z position of the mass center was determined using a custom composition that is depicted in Figure 2.5. It consists of two axes and two pairs of 3D printed parts. Two of those parts were attached to two opposite motors of the Crazyflie. The two parts that were left were attached to two axes and then slid into the parts that were attached to the Crazyflie. Subsequently the two axes were fixed on a table so that the Crazyflie could spin freely and the quadrocopter was brought into a position where its z axis pointed horizontally. The z position of the mass center was then determined by sliding the 3D printed parts back and forth until the Crazyflie was balanced and stopped leaving its vertical position. Finally, measuring the position of the 3D printed parts led to the result that the Crazyflie's mass center lies 17.425 mm above the ground when the quadrocopter stands on a flat surface.

Friction

A starting value for the friction coefficient M_{fr} required for the fitting process was determined experimentally. To achieve this, known weights were attached to a fishing line that was wrapped up on the axis of the Crazyflie swing. Subsequently the weight was dropped and the dropping motion was recorded using the encoder. The experiment setup is depicted in Figure 2.6.

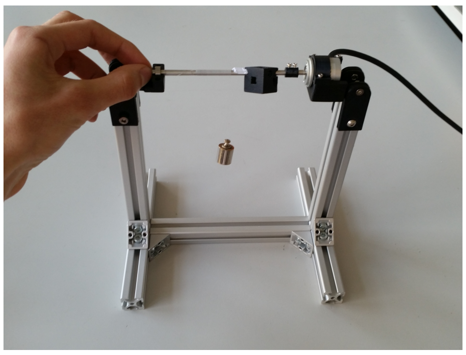


Figure 2.6: Experiment setup used to determine the friction coefficient M_{fr} of the Crazyflie swing. A fishing line is wrapped around the axis and a known weight is attached to its end. The friction coefficient can be determined via the steady state velocity of the dropping weight.

The equations describing this situation were already introduced in Section 2.3.2. The equation of motion is

$$I_{\text{exp}}\ddot{\varphi}(t) = m_{\text{weight}} g r - M_{\text{fr}}\dot{\varphi}(t) \tag{2.12}$$

The solution to this equation which is derived in Appendix B.2 under the assumptions that $\varphi(0) = 0$ and $\dot{\varphi}(0) = 0$ is

$$\varphi(t) = \frac{m_{\text{weight}} g r I_{\text{a}}}{M_{fr}^2}\left(e^{-\frac{M_{fr}}{I_{\text{a}}}t} - 1\right) + \frac{m_{\text{weight}} g r}{M_{fr}} t \tag{2.13}$$

$$\dot{\varphi}(t) = \frac{m_{\text{weight}} g r}{M_{fr}} \left(1 - e^{-\frac{M_{fr}}{I_a} t}\right) \tag{2.14}$$

The steady-state velocity of the spinning axis is $\dot{\varphi}(t \to \infty) = \frac{m_{\text{weight}} g r}{M_{fr}}$. It can easily be converted to the translational velocity of the dropping weight: $v(t) = \dot{\varphi}(t) \cdot r$.

Figure 2.7a shows the recorded encoder data for an example experiment where a weight of $m_{\text{weight}} = 20\,\text{g}$ dropped from approximately one meter. The axes of the diagram were adapted as explained in Section 2.4.4.

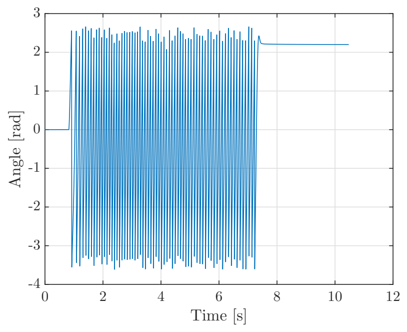
For the next step the data was unwrapped and the derivative was computed using Matlab's `diff` command. Additionally the speed data was filtered with a median filter (orders between 1 and 11) to reject outliers. The results are displayed in the Figures 2.7b and 2.7c.

In order to get only one value for the angular velocity, the velocity was averaged in a manually chosen interval. The interval was chosen so that the speed data in this interval was as constant as possible (apart from noise).

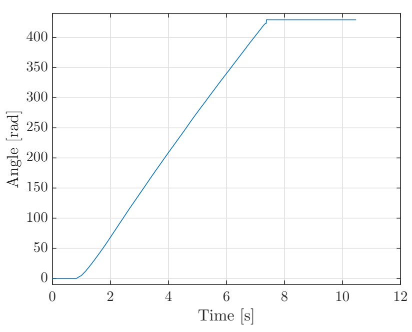
This whole procedure was repeated for different weights. All results are summarized in Table 2.1 and some experiment data is depicted in Figure 2.8.

Table 2.1: Results of the dropping mass experiments for different weights. The speeds were calculated by averaging the measured angular velocity in a manually selected interval. The resulting friction coefficients were calculated using the average speeds.

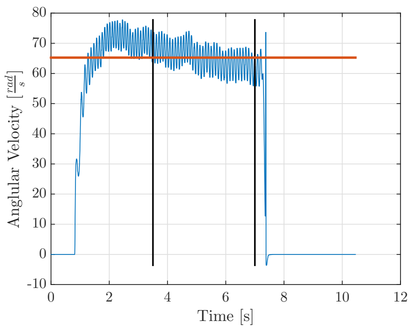
	Index i	Mass m_{weight}	Averaged speed v	Resulting M_{fr}
Unit	-	g	rad/s	$10^5 \cdot \text{kg}\cdot\text{m}^2/\text{rad}\cdot\text{s}$
	1	20	51.97	0.7551
	2	20	57.13	0.6869
	3	20	57.41	0.6835
	4	20	58.13	0.6750
	5	20	66.53	0.5898
	6	30	232.37	0.2533
	7	30	280.48	0.2099
	8	30	271.58	0.2167
	9	30	291.17	0.2022
	10	30	283.76	0.2074
	11	40	498.07	0.1576
	12	40	463.00	0.1695
	13	40	470.78	0.1667
	14	40	513.76	0.1528
	15	50	588.37	0.1667
	16	50	562.57	0.1744
	17	50	603.36	0.1625
	18	50	591.74	0.1658
	19	50	608.88	0.1611



(a) Raw data of an experiment where a mass of 20 g dropped down approximately 1 m. The axes were scaled as described in Section 2.4.4.

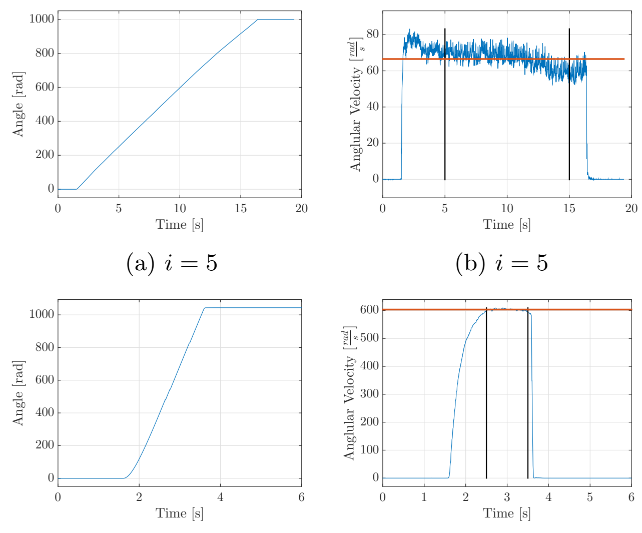


(b) The same data unwrapped.



(c) Angular velocity of the spinning axis calculated from the unwrapped data and filtered with a median filter (order 7). The black vertical lines are manually chosen borders and the orange horizontal line is the average of the speed data between these borders. The borders are chosen so that the speed between them is as constant as possible.

Figure 2.7: Data from the dropping weight experiment. The code that was used to generate these plots can be found in Appendix C.1.2.



(a) $i = 5$ (b) $i = 5$

(c) $i = 17$ (d) $i = 17$

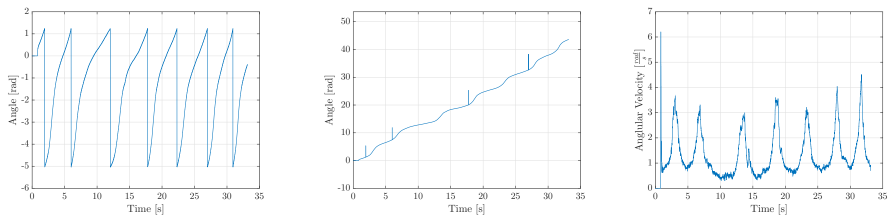
Figure 2.8: Data from selected dropping weight experiments. The indices correspond to the ones in Table 2.1. Left column: unwrapped measurement data, thus showing the position over time. Right column: derivative of unwrapped data and mean value in depicted interval. The code that was used to generate these figures is similar to the one in Appendix C.1.2.

When analyzing the last column of Table 2.1 it becomes clear that the resulting friction coefficient is relatively similar for all experiments except from the ones where the average speed of the dropping weight is low compared to the other experiments ($i = [1, 5]$).

In addition the experiments were also carried out with a weight of $m_{\text{weight}} = 10\,\text{g}$. The data for this experiment can be seen in Figure 2.9. This data was not included in the above table because in this case the speed and therefore also the friction was not constant during one rotation of the axis.

To sum it up, the friction of the experiment setup seems to be influenced by currently inexplicable effects, especially at low angular velocities of the axis. For this reason, the effect of friction at slow speeds on the resulting moment of inertia was further investigated. Details on that can be found in Section 2.4.4.

Another conclusion that can be drawn from the dropping weight experiment concerns the moment of inertia of the experiment setup itself (axis and mounting cube). Equation (2.14) allows to conclude that the stationary speed is reached more slowly when the moment of inertia of the spinning axis is higher. During all dropping weight experiments the mounting cube was attached to the axis. As can be seen in Figure 2.7b the acceleration almost happens instantaneously. For this reason and because of the satisfying results during the verification (Section 2.6) it was decided to neglect the moment of inertia of the mounting cube and the axis for the calculation of the moments of inertia during this experiment.



(a) Raw data of an experiment where a mass of 10g dropped down approximately 1m.

(b) The same data unwrapped.

(c) Derivative of the unwrapped data. A median filter of order 7 was applied to reject outliers.

Figure 2.9: Data from the dropping weight experiment. From the changing speed we can infer that the friction at slow speeds is influenced by nonlinear and unapparent effects. The code that was used to generate these plots can be found in Appendix C.1.2.

2.4.3 Experimental Procedure

Taking measurements using the setup as described in Section 2.4.1 was performed by the following steps:

1. Attach the mounting cube to the Crazyflie.
2. Mount the cube together with the Crazyflie to the swinging axis in a desired orientation. Prevent slipping between cube and axis by fixing the cube using folded paper.
3. Trigger the recording on the computer.
4. Deflect the Crazyflie by hand to an arbitrary angle between 20 and 70 degrees.
5. Let the Crazyflie swing until it stops.
6. Stop the recording on the computer.

2.4.4 Data Analysis

After finishing the steps described above the data had to be post processed and analyzed in order to get the moment of inertia. Figure 2.10 shows the starting point of this procedure. The data that is plotted there was not processed at all.

Preparing the Data

As a first step the units of the axes were adapted: the time axis was changed to start at 0 and the unit was changed to seconds. Additionally the unit of the encoder data axis was changed to radians. To do this the maximal encoder value was determined by turning the encoder axis manually by more than 360° and identifying the maximum which showed to be 4194047. Subsequently the data points were scaled to be between 0 and 2π instead. Also, the mean of the

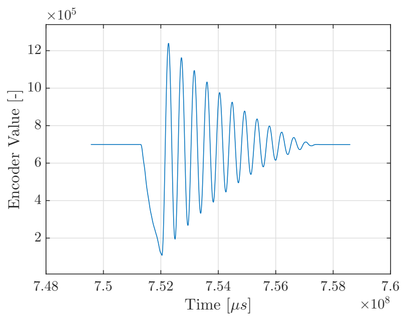


Figure 2.10: Raw data that was recorded with the radial encoder while deflecting the Crazyflie by hand around its x axis and waiting until it stopped swinging. The data in this figure was not post processed in any way.

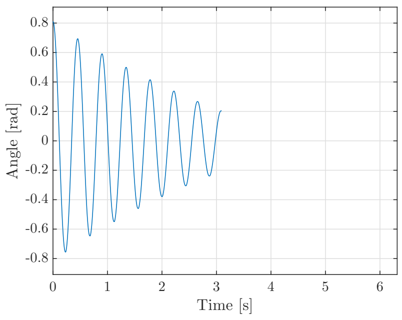


Figure 2.11: Result after changing the units and removing the biases of both axes as well as deleting the parts of the data where the Crazyflie was deflected or where the Crazyflie moves at slow speeds.

first few data points was subtracted from all data points in order to have the motion start at 0 rad.

Secondly, the part where the Crazyflie was deflected by hand was removed by finding the highest peak and deleting all data points before that peak.

Finally, the data where the angle was below approximately 10° was deleted as well. In the example in Figure 2.11 this corresponds to only considering data up to the 8th peak. This was done because the motion of the experiment setup showed an inexplicable nonlinear behavior when moving at small speeds. More information on this can be found in Section 2.4.2. The result of these processing steps can be seen in Figure 2.11.

However as is presented later (Section 2.6), this experiment gives the most accurate results for the moments of inertia when the fit is done for all the available data and not just up to the 8th peak. It is not yet understood why this is the case. Despite this lack of understanding, a fit using all the data was also done for the Crazyflie. This fit was then compared to the result of the fit using data up to the 8th peak.

Fitting the Data

Alternatively to fitting the experiment data, the logarithmic decrement $\ln\left(\frac{\varphi(t)}{\varphi(t+T)}\right) = T\delta$ also could have been used to calculate the moment of inertia from the data. However when calculating the decrement, the results appeared to vary quite a lot. For this reason, fitting the solution of the linearized equation of motion to the recorded data was chosen to determine the moment of inertia.

Linear Fitting The fitting was done using Matlab's function `fminsearch`. The objective function that is based on the solution to the linearized equation of motion of a pendulum (eq. (2.11)) is

$$\sum_{i=1}^{N}\left(\varphi_{\mathrm{exp,i}}-\overbrace{\sqrt{\varphi_0^2+\left(\frac{\Omega_0+\delta\varphi_0}{\omega}\right)^2}\cdot e^{-\delta\cdot iT_{\mathrm{s}}}\cdot\cos\left(\omega\cdot iT_{\mathrm{s}}-\arctan\left(\frac{\Omega_0+\delta\varphi_0}{\omega\varphi_0}\right)\right)}^{\varphi(t)}\right)^2 \tag{2.15}$$

where N is the number of data points, $\varphi_{\mathrm{exp,i}}$ is the i^{th} data point and T_{s} is the sampling time which was $T_{\mathrm{s}} = 0.004\,\mathrm{s}$ in this experiment. All other variables have the same definition as in equation (2.11).
The parameters M_{fr}, I_{a} and Ω_0 were optimized and all other variables were declared as constants. The initial values and the values of the constants as well as their derivations are summarized in Table 2.2.
The Matlab file that was used for the fitting can be found in Appendix C.1.3. In Figure 2.12 an example for a result of such a fitting is depicted.

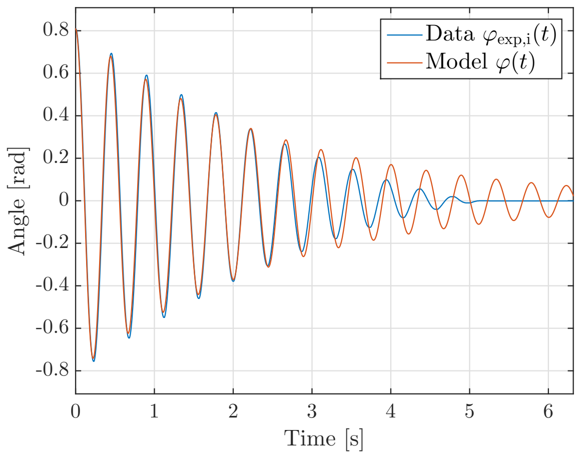


Figure 2.12: Result of fitting the solution of the linearized equation of motion to the data. During this experiment, the Crazyflie was swinging about its x axis.

Nonlinear Fitting As the maximum angle the Crazyflie had during the swing experiment exceeded 50° from time to time, there was a reason to doubt the validity of the small angle assumption that was made when linearizing the equation of motion. In order to investigate the effects of the mistake that was introduced by this, a fit to the nonlinear equation of motion (eq. (2.9)) was done as well. When fitting the data to the nonlinear equation of motion the first and the second derivative of the data are needed as well. However when numerically taking the derivative of discrete data points, existing noise is amplified [4]. To avoid this effect a different approach was chosen. When fitting a well defined at least two times differentiable function to the data it is possible to analytically take

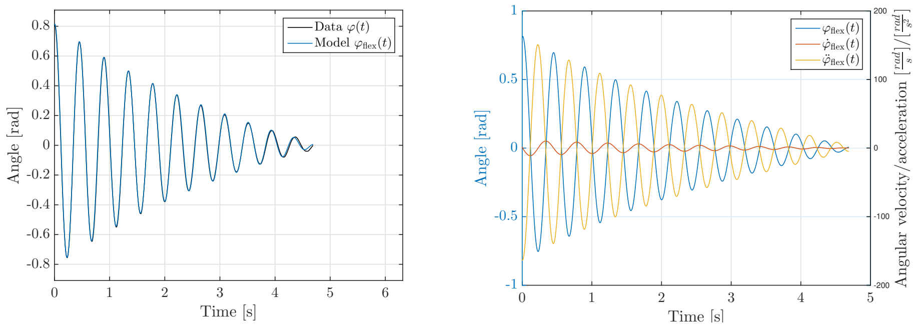
Table 2.2: Initial values for parameters that are fitted using Matlab's `fminsearch` and values of numeric constants that are also needed to perform the fit. In the right column there is an explanation for every value on how it was determined.

	Optimized while fitting?		
Variable	Yes: Initial Value	No: Value	Origin
M_{fr}	$0.1813 \cdot 10^{-5} \mathrm{kg \cdot m^2}/\mathrm{rad \cdot s}$		Mean of all results from Table 2.1 with indices $i = [6, 19]$
I_{a}	$4.48 \cdot 10^{-5} \mathrm{kg} \cdot \mathrm{m}^2$		Moment of inertia $I_{\mathrm{pm}} = mr^2$ of a point mass (same weight as Crazyflie, $m_{\mathrm{CF}} = 28\mathrm{g}$) with distance from axis that equals approximately half the diameter of the Crazyflie ($d_{\mathrm{CF}} = 0.092\,\mathrm{m}$)
Ω_0	$0 \mathrm{rad}/\mathrm{s}$		Not exactly but very close to 0 because Crazyflie is deflected and launched manually. Optimized to improve the fit.
φ_0		$\varphi_{\mathrm{exp},0}$	Is set equal to the first data point
m_{CF}		$28.0\,\mathrm{g}$	Measured using scales (Snowrex EA-3000 with a precision of 0.1g)
g		$9.81\,\mathrm{N}/\mathrm{kg}$	Earth's gravity
r		r_i	The distance between the rotation axis to the mass center depends on the current mounting orientation of the Crazyflie and is calculated with the Matlab script in Appendix C.1.7

the derivative of this function. The results are the desired first two derivatives of the data.
After analyzing the plot of the experiment data the following function was chosen:

$$\varphi_{\text{flex}}(t) = \left(a_1 \cdot t^2 + a_2 \cdot t + a_3 + e^{-a_4 \cdot t}\right) \cdot \cos\left(\left(a_5 + a_6 \cdot t + a_7 \cdot t^2\right) \cdot t + a_8\right). \tag{2.16}$$

Fitting this equation to the measured data (result in Figure 2.13a) and taking the analytical derivative using Wolfram Mathematica led to the first and second derivative of the data (result in Figure 2.13b). Subsequently the nonlinear equation of motion could directly be fitted to the data.



(a) Fitted function compared to the data. The curves are nearly indistinguishable.
(b) Fitted function (left y-axis) with its first and second derivative (right y-axis).

Figure 2.13: Result of fitting the function from equation (2.16) to the data from the swinging experiment (x axis of the Crazyflie in this case) and taking the analytical derivatives.

2.5 Results and Discussion

Linear Fitting The results for the linear fitting are summarized in Table 2.3. Due to geometrical symmetries of the Crazyflie the moments of inertia are expected to be very similar for the x and y axes or for the x-z and x-y axes respectively. This can be observed in the results. Also the expectation that the initial angular velocity is close to zero is met.
However another result is not as expected: the friction coefficient. Looking only at the results from the swing experiment is quite pleasing because the friction coefficient is similar for all measurements. But as soon as one compares these coefficients to the ones determined through the dropping mass experiment (see Section 2.4.2) it becomes clear that the new coefficients are one order of magnitude larger than the predicted ones. Apart from the nonlinear friction effects at small speeds no possible explanation could be found and further investigation is required. However as the results of the verification experiment (Section 2.6) regarding the accuracy of the determined moment of inertia were very encouraging it was decided to abdicate these investigation for the time being.

Table 2.3: This table shows the results of fitting the solution of the *linearized* equation of motion to the measured data. # of peaks says up to which peak the measured data is taken into account for the fitting process. This data was generated using the Matlab script that is shown in Appendix C.1.3.

	Axis	# of peaks	I_{a}	M_{fr}	Ω_0
Unit	-	-	$10^{-5} \cdot \text{kg} \cdot \text{m}^2$	$10^{-5} \cdot {}^{\text{kg}\cdot\text{m}^2}/_{\text{rad}\cdot\text{s}}$	${}^{\text{rad}}/_{\text{s}}$
	x	8	3.144988	2.446468	-0.0644
	x	11	3.119770	2.683766	0.1842
	y	8	3.151127	2.521742	-0.0610
	y	11	3.128159	2.773404	0.1553
	z	8	7.058874	2.701683	-0.0625
	z	16	7.004148	3.214519	-0.0805
	xy	8	5.003777	2.392808	-0.0230
	xy	16	4.940333	2.849633	-0.0784
	xz	8	4.640540	2.560766	-0.0045
	xz	13	4.600706	2.920212	-0.0278
	yz	8	4.780235	2.191634	-0.0334
	yz	15	4.713093	2.684329	-0.0050

Nonlinear Fitting The results for the nonlinear fitting are displayed in Table 2.4. It was expected that these results were more accurate than the ones for the linear fitting because the small angle assumption was not relied on here. But when looking at the resulting moments of inertia and friction coefficients it is obvious that they are (with one exception) nearly exactly equal to the ones that were determined using linear fitting. Consequently it was decided to proceed with the results from the linear fitting.

Table 2.4: This table shows the results of fitting the parameters in the *nonlinear* equation of motion to the measured data. This data was generated using the Matlab script that is shown in Appendix C.1.3.

	Axis	I_{a}	M_{fr}
Unit	-	$10^{-5} \cdot \text{kg} \cdot \text{m}^2$	$10^{-5} \cdot {}^{\text{kg}\cdot\text{m}^2}/_{\text{rad}\cdot\text{s}}$
	x	3.144988	2.446468
	y	3.151127	2.521742
	z	7.058874	2.836767
	xy	4.753588	2.512448
	xz	4.640540	2.560766
	yz	4.541223	2.301216

Inertia Matrix The goal of this experiment was to determine the inertia matrix of the Crazyflie with respect to its mass center. In order to calculate this matrix from the results that are contained in Table 2.3, the moments of inertia with respect to the mounting axes were converted into moments of inertia with respect to parallel axes that pass the mass center using the parallel axis theorem (eq. (2.3)). Subsequently a linear system of equations was set up based on equation (2.2). The solution of this system of equations is a vector containing the six independent components of the inertia matrix. The resulting inertia matrix for all experiments, where the first eight peaks were used for the fit, is

$$\boldsymbol{I}_{\mathbf{CF,1}} = \begin{pmatrix} 16.823890 & 1.224320 & 0.716891 \\ 1.224320 & 16.885278 & 2.083147 \\ 0.716891 & 2.083147 & 29.808912 \end{pmatrix} \cdot 10^{-6}\mathrm{kg}\cdot\mathrm{m}^2. \tag{2.17}$$

The inertia matrix, based on the fits where a higher number of peaks was considered, is

$$\boldsymbol{I}_{\mathbf{CF,2}} = \begin{pmatrix} 16.571710 & 0.830806 & 0.718277 \\ 0.830806 & 16.655602 & 1.800197 \\ 0.718277 & 1.800197 & 29.261652 \end{pmatrix} \cdot 10^{-6}\mathrm{kg}\cdot\mathrm{m}^2. \tag{2.18}$$

All steps described above that were used to obtain the inertia matrices were executed using the Matlab script in Appendix C.1.4.
Both inertia matrices are very similar. However as in the verification process better results were yielded when fitting to the whole data and not just to the first few peaks (see Section 2.6) it was decided to settle on the second inertia matrix:

$$\boldsymbol{I}_{\mathbf{CF}} = \boldsymbol{I}_{\mathbf{CF,2}}. \tag{2.19}$$

There are two reasons why this result for the inertia matrix makes sense.
Firstly, the components on the diagonal (moments of inertia) are significantly larger than the products of inertia. If our assumption that the mass center and the geometrical center of the Crazyflie are identical was correct, the products of inertia would have been zero. This is the case because the x-y, x-z and the y-z planes in the Crazyflie body frame are planes of symmetry [8].
Secondly, the moment of inertia with respect to the z axis is approximately twice as large as the moments of inertia with respect to x and y axis. Why this is expected can be understood intuitively: The motors are the heaviest components whose mass center positions are not even similar to the position of the overall mass center. Assume that d_{CF} is the distance between two opposite motor axes of the Crazyflie. When the Crazyflie is now spinning about its z axis the motors have a distance of $d_{\mathrm{z}} := \frac{d_{\mathrm{CF}}}{2}$ to the common mass center. In contrast when spinning about the x or y axis, the motors have a distance of $d_{\mathrm{x,y}} := \frac{\sqrt{2}}{4} d_{\mathrm{CF}}$ to the common mass center (compare Figure 2.14). It follows that $\frac{d_{\mathrm{z}}}{d_{\mathrm{x,y}}} = \sqrt{2}$. At the same time the moment of inertia is proportional to the square of the distance between a mass point and the rotation axis: $I \propto d^2$. Putting everything together leads to

$$\frac{I_{\mathrm{z}}}{I_{\mathrm{x,y}}} = \left(\frac{d_{\mathrm{z}}}{d_{\mathrm{x,y}}}\right)^2 = 2 \implies I_{\mathrm{z}} = 2 \cdot I_{\mathrm{x,y}} \tag{2.20}$$

which is what can be observed in the result for the inertia matrix (eq. (2.18)).

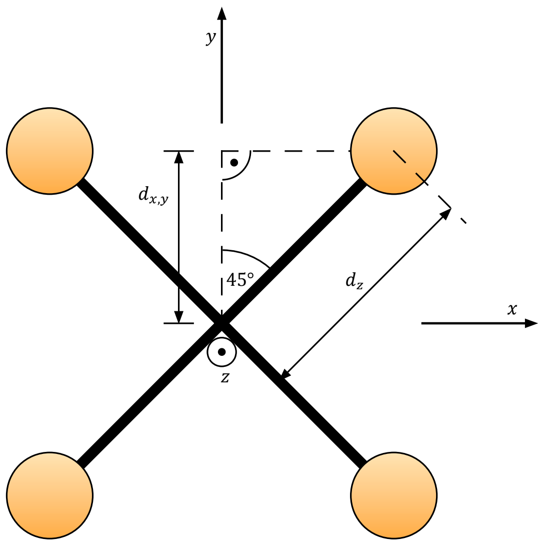


Figure 2.14: Geometrical situation that is used to prove why $I_\mathrm{z} = 2 \cdot I_\mathrm{x,y}$.

2.6 Verification

To verify that the experimental procedure described above yields the correct results for the moment of inertia a test body with known mass ($m_\mathrm{TB} = 17.6\,\mathrm{g}$) and dimensions was built. The idea behind the test body is to measure its moment of inertia in exactly the same way as the Crazyflie's and to then additionally calculate its moment of inertia. Comparing the measured moment of inertia to the calculated one is a promising test for the precision of the experimental method. Figure 2.15 shows a picture of the test body that was used.

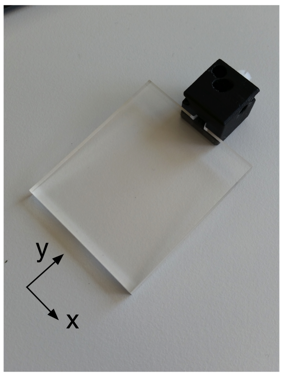


Figure 2.15: Polymethylmethacrylat (PMMA) test body with the mounting cube already attached to its shaft.

Calculation of moments of inertia When calculating the moments of inertia of the test body we make use of the formula for the moment of inertia of a cuboid:

$$
\begin{aligned}
I_{\mathrm{xx,G}} &= m \cdot \left(\frac{b^2}{12} + \frac{c^2}{12} \right) \\
I_{\mathrm{yy,G}} &= m \cdot \left(\frac{a^2}{12} + \frac{c^2}{12} \right) \\
I_{\mathrm{zz,G}} &= m \cdot \left(\frac{a^2}{12} + \frac{b^2}{12} \right)
\end{aligned}
\tag{2.21}
$$

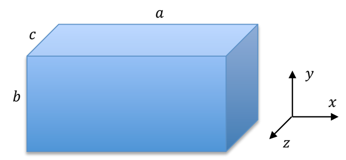


Figure 2.16: Geometrical situation for inertia formulas for cuboids.

The geometrical situation corresponding to this formula is shown in Figure 2.16 and the derivation of the formula can be found in Appendix B.3.
Using this formula as well as the parallel axis theorem (eq. (2.3)) we get the values shown in Table 2.5. The calculation was done using the Matlab script from Appendix C.1.5.

Measurement of moments of inertia The experimental procedure when measuring the moments of inertia of the test body was exactly the same as for the Crazyflie. Only the calculations of the distances from the axes to the body's mass center and the moments of inertia with respect to a mass center axis were a little bit different because of the different geometry. The file that was used to calculate the distances is the second one in Appendix C.1.5 and the file that was used to move the measured moments of inertia to the mass center is in Appendix C.1.6. The results can be found in Table 2.5.
When looking at the results it becomes clear that the measured moments of inertia are more accurate when the fit is done for more data points.

Table 2.5: Calculated and experimentally determined values for two different moments of inertia of the test body. The measured moments of inertia for (1) were obtained by fitting data up to the 10^{th} and 13^{th} peak for x and z axis respectively. For (2) the fit was done using data up to the 8^{th} peak for both axes. The numbers in the 'error' lines are relative errors between measured and calculated moments of inertia.

	$I_{\mathrm{xx,G}}$	$I_{\mathrm{zz,G}}$
Calculated	$6.410179 \cdot 10^{-6}\,\mathrm{kg\,m^2}$	$9.860228 \cdot 10^{-6}\,\mathrm{kg\,m^2}$
Measured (1)	$6.428337 \cdot 10^{-6}\,\mathrm{kg\,m^2}$	$9.998638 \cdot 10^{-6}\,\mathrm{kg\,m^2}$
Error (1)	0.2833 %	1.4037 %
Measured (2)	$6.530529 \cdot 10^{-6}\,\mathrm{kg\,m^2}$	$10.426414 \cdot 10^{-6}\,\mathrm{kg\,m^2}$
Error (2)	1.8775 %	5.7421 %

In addition as the error is very small, the test body experiment successfully validates the choice of this experiment for the determination of the inertia matrix of the Crazyflie.

Chapter 3

Motor Parameters

In this chapter, the process of characterizing the Crazyflie's motors is presented. This characterization includes mappings between important operation variables, such as produced thrust and torque and the input command passed to the motors from within the Crazyflie's firmware. In addition, the motor's transfer function was determined.

3.1 Applications

The controller that will be implemented on the Crazyflie will output a value for every motor that represents the thrust that should be produced by this motor. However this value cannot be directly commanded to the motors. The motor driver on the Crazyflie only accepts a 16 bit integer (0 = motors off, 65535 = full thrust) for each motor at a time that subsequently is converted into a PWM signal [7]. In order to make a connection between the set point given in Newton and the 16 bit motor command we need a conversion function.

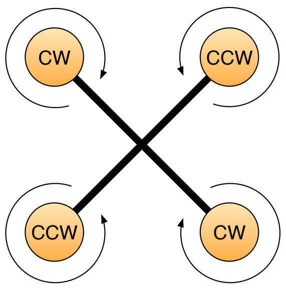


Figure 3.1: Spinning directions of a quadrocopter's rotors. CW = clockwise, CCW = counterclockwise.

Apart from that, the Crazyflie's rotation about its z axis (yaw) is controlled by making use of the torque that is produced by the rotors spinning in air. The vector of the produced torque points exactly in the opposite direction of the rotor's angular velocity vector. In order to prevent the quadrocopter from spinning all the time, this effect has to be abrogated which is done by letting two propellers spin clockwise and the other two counterclockwise (compare Figure

3.1). When the quadrocopter is hovering all propellers have approximately the same speed and therefore generate torques with similar magnitudes that cancel each other out. Now when a rotation about the z axis is desired, the speed of two opposite propellers is reduced and the speed of the two remaining propellers is increased equally so that the total thrust remains the same. The result is a torque that generates the desired rotation. In order to determine the thrust reduction and increase necessary to achieve a specific torque, a mapping between torque and thrust is needed.
Furthermore the rotor's angular velocity can be used to calculate the thrust the rotor currently produces. The following formula introduced in [13] shows the connection between those two values:

$$f = \sum_{i=1}^{4} \kappa \dot{\theta}_i^2 \tag{3.1}$$

In this equation, f is the total thrust produced, κ is a proportionality constant and $\dot{\theta}_i$ is the angular velocity of rotor i. As the Crazyflie features brushed motors, a direct measurement of rotor angular velocity is not possible. Instead the angular velocity has to be estimated. For this, a mapping between input command given to the motors and angular velocity was determined as well.
Finally the motor transfer function was determined. It can be used to predict how fast the motor will adapt to a new velocity and it is planned to be used for simulation purposes.

3.2 Equipment

3.2.1 Load Cell

The produced thrust and torque of the quadrocopter was measured with an ATI industrial automation force/torque sensor Mini40 (load cell) [1]. As is depicted in Figure 3.2 fixing the Crazyflie on the load cell was done using a custom 3D printed mount.

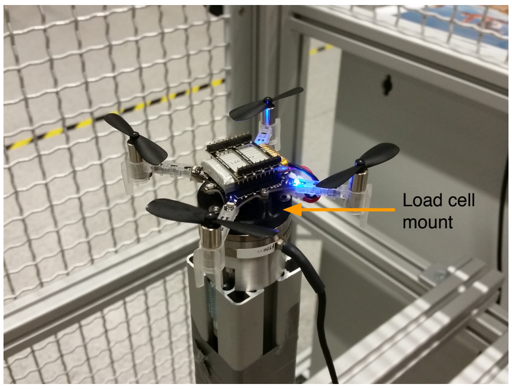


Figure 3.2: The Crazyflie is attached to the load cell using a custom 3D printed mount.

To log the load cell data a Python logger was implemented (Appendix C.2.2). The extreme noise observed when looking at the resulting data really stood out (compare Figure 3.3). However as only steady-state forces and torques were interesting for the mappings, taking the average in the interval in which the motors were turned on gives the required result.

It is also interesting to take the FFT of the load cell data (see Figure 3.4). It reveals that the noise of the load cell data is concentrated at a few distinct frequencies. These frequencies probably correspond to the frequency of the Crazyflie's spinning rotors.

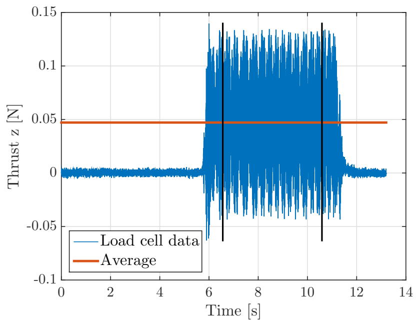


Figure 3.3: Thrust produced by one motor while being turned on with an input command of 28000 (43 % of maximum thrust) and off again. In order to get a single value for the produced thrust the data is averaged in a manually chosen interval.

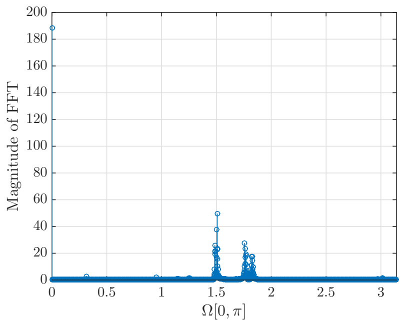


Figure 3.4: FFT of the load cell data from Figure 3.3. The noise is concentrated at very distinct frequencies that most probably correspond to the rotational frequency of the Crazyflie's rotors.

3.2.2 Tachometer

To measure the angular velocities of the rotors a UNI-T UT372 laser tachometer was used. It was mounted below the load cell as depicted in Figure 3.5.

To make the Crazyflie's rotor wings visible for the tachometer, sticky reflective marker tape was applied to both wings of a rotor (Figure 3.6). Sticking tape on solely one wing was not possible because it would have disturbed the balance of the propeller significantly. Even with two prepared wings an increase in vibration during operation could be heard.

Because of this the RPM value that was displayed on the tachometer had to be divided by two. For converting the RPM into an angular velocity ω with unit $^{\text{rad}}/_{\text{s}}$, the formula

$$\omega = 2\pi \cdot \frac{RPM}{2 \cdot 60} = \frac{\pi}{60} \cdot RPM \tag{3.2}$$

was used. In this equation RPM is the value that was read from the tachometer.

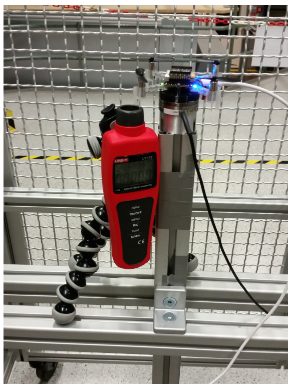


Figure 3.5: Mounting position of the tachometer. It is fixed to the aluminium profiles using a tripod.

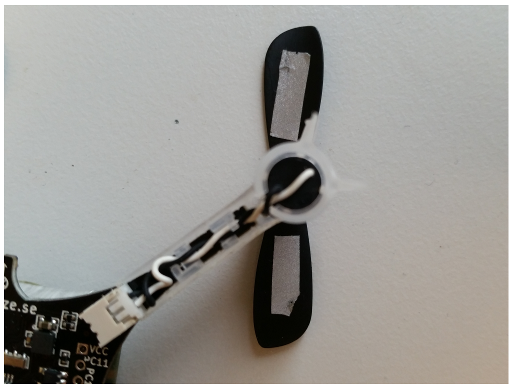


Figure 3.6: Reflective tape beneath the rotors of the Crazyflie. As the tachometer now sees both wings of one propeller the displayed RPM has to be divided by two. Only using one reflective tape was not possible due to balancing issues.

3.3 Mappings

3.3.1 Input Command → Thrust

Experiment Setup and Procedure In order to determine the input command to thrust mapping, a way to provide the Crazyflie with an input command as well as a way to measure the thrust the Crazyflie produces was needed.
For this, the Crazyflie's firmware was extended in order to be able to send constant thrust input commands to the motors. The files that were added can be found in Appendix C.2.1. The commands are transmitted from a computer via the Crazyradio PA and using a Python script that can be found in the same appendix.
The thrust was measured using the load cell as described in Section 3.2.1.

After mounting the Crazyflie to the load cell the following steps were repeated until enough data points were collected.

1. Start logging the load cell data to a file.
2. Send a setpoint from the interval $[0, 65535]$ to two opposite motors of the Crazyflie using the script mentioned above. Wait for approximately 10 s.
3. Send the 0 setpoint to the Crazyflie and stop the logging.
4. Repeat the above steps until all input commands between 0 and 64000 with an increment of 2000 are measured.

The setpoints are sent to two opposite motors of the Crazyflie because this results in torque and thrust values that are twice as high compared to when only one motor is turned on. This increases the signal-to-noise ratio and therefore reduces the measurement error.

Data Processing As described in Section 3.2.1 the average was taken in a manually selected interval in which the Crazyflie's rotors were turned on. Subsequently a polynomial with degree two was fitted to the data. The data processing was done with the Matlab file in Appendix C.2.3 and uses the command `polyfit`.

Results The data processing led to the following function

$$f_{\mathrm{i}} = 2.130295 \cdot 10^{-11} \cdot cmd_{\mathrm{i}}^2 + 1.032633 \cdot 10^{-6} \cdot cmd_{\mathrm{i}} + 5.484560 \cdot 10^{-4} \quad (3.3)$$

where f_{i} is the thrust produced by rotor i and cmd_{i} is the input command passed to motor i.
A plot of this function can be found in Figure 3.7.

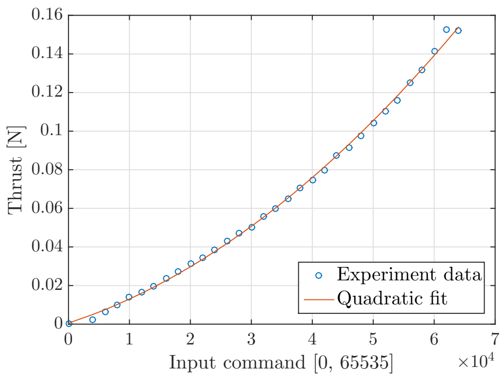


Figure 3.7: Mapping between input command and thrust for one motor together with the data points that were used to compute the mapping.

Verification To verify that the load cell provides the correct values for thrust, a second experiment was undertaken. It included mounting the Crazyflie to scales (see Figure 3.8) and measuring the thrust produced by the Crazyflie via the weight reduction compared to when the motors were turned off. The Crazyflie had to be mounted offset from the scales and even the table because when mounting the Crazyflie centrally over the scales the ground effect of the air stream falsified the measurements. Figure 3.9 shows the results for this verification experiment together with the load cell data and the fit from above. As the verification data is very close to the load cell data it is assumed that the load cell data is correct.
In addition to this, it could be confirmed that the maximum thrust measured for one motor in the load cell is most probably correct by comparing it to a value given by Bitcraze on their homepage. According to them the maximum takeoff weight for the Crazyflie is 42 g [2]. In the experiment described above the maximum thrust that can be produced by one motor was determined to be approximately 0.15 N. From this it follows that the maximum mass that could be kept in hover by four motors is $\frac{4 \cdot 0.15\,\mathrm{N}}{9.81\,\mathrm{N/kg}} = 61.16\,\mathrm{g}$. Note that if the Crazyflie

did indeed weigh 61.16 g, it would only be able to hover at its current height and not ascend. Therefore the maximum thrust that was determined seems to be well proportioned to carry a weight of 42 g as specified by Bitcraze and therefore quite realistic.

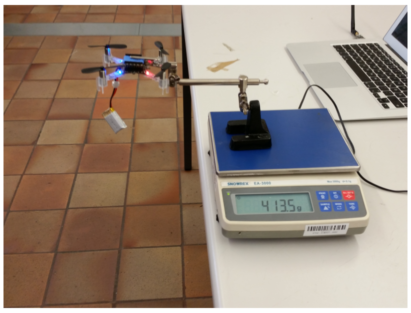


Figure 3.8: Experimental setup that was used to verify that the load cell provides correct values for the thrust produced by the Crazyflie.

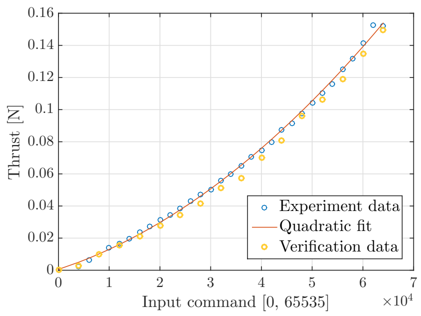


Figure 3.9: Results of the verification experiment compared to the results from the load cell experiment.

3.3.2 Input Command → Angular Velocity

Experiment Setup and Procedure Input commands were transmitted to the Crazyflie as described in Section 3.3.1 and the rotors angular velocity was measured as described in Section 3.2.2.
The following steps were performed until all data points were collected.

1. Send a setpoint from the interval $[0, 65535]$ to the motor with the reflective tape using the script mentioned above.
2. Wait until the display of the tachometer stabilized. Write down the value.
3. Turn the motor off.
4. Repeat the above steps until all input commands between 0 and 64000 with an increment of 2000 are measured.

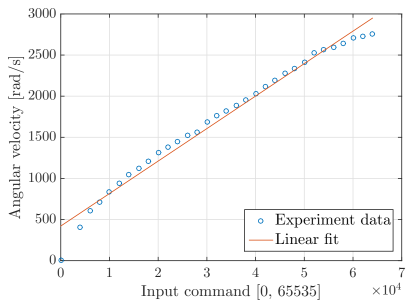
In total two series of measurements were recorded.

Data Analysis From the measured data points the angular velocities were calculated using equation (3.2). The data analysis was done with the Matlab file in Appendix C.2.3. After inspection of the data it was decided to fit a polynomial with degree one to the data.

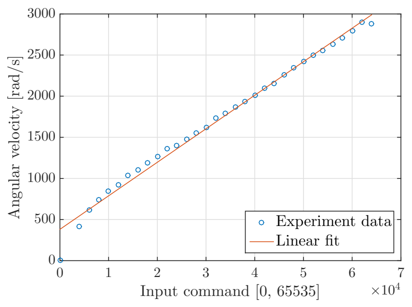
Results The data analysis led to the following two mappings that are displayed in Figure 3.10:

$$\begin{aligned}\theta_{\text{i}} &= 0.03950236 \cdot cmd_{\text{i}} + 420.9420\\ \theta_{\text{i}} &= 0.04076521 \cdot cmd_{\text{i}} + 380.8359\end{aligned} \tag{3.4}$$

The norm of the residuals is equal to 599.6187 for the first series of measurements and 471.0714 for the second one. Therefore the fitting is better for the second series.



(a) First series of measurements.



(b) Second series of measurements.

Figure 3.10: Results and linear fit for the two measurement rows of rotor angular velocity.

As the behavior of the angular velocity is not linear in the input command interval $[0, 1000]$ it is not recommended to use the mapping there. Instead, the angular velocity should be assumed to be zero in this interval.

3.3.3 Thrust $\rightarrow$ Torque

Experiment Setup and Procedure Thrust as well as torque was measured as described in Section 3.2.1 and input commands were again transmitted to the Crazyflie as introduced in 3.3.1.
Also for this mapping, data points corresponding to input commands between 0 and 64000 with an increment of 2000 were collected.
The experiment was done using two opposite propellers in order to get more accurate results.

Data Analysis The data was analyzed using the Matlab script in Appendix C.2.3.

Results The following function resulted from fitting a linear polynomial to the data. It is plotted in Figure 3.11.

$$\tau_{\mathrm{i}} = 0.005964552 \cdot f_{\mathrm{i}} + 1.563383 \cdot 10^{-5} \tag{3.5}$$

In this equation τ_{i} is the torque and f_{i} the thrust, each produced by one rotor.

3.4 Transfer Function

This section describes the process of determining a transfer function between input command given to the motors and produced thrust for the Crazyflie's motors.

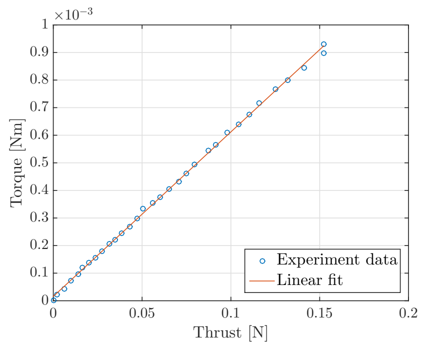


Figure 3.11: Mapping between produced thrust and torque for one motor together with the data points that were used to compute the mapping.

3.4.1 Theoretical Background

As already illustrated in Section 3.2.1 the data recorded using the load cell has a very bad signal-to-noise ratio. When taking measurements for the mappings it was possible to circumnavigate this problem by taking the average over the interval where the Crazyflie's motors were turned on because only steady state forces and torques were needed for the mappings. However when determining the transfer function the steady state signal is not sufficient because it does not contain any information on how fast the motors react.
In order to solve the problem with noise this time a method explained by D'Andrea et al. in [5] is employed. The idea of this method is to input sinusoidal signals of different frequencies into the plant that is to be identified. When a sinusoidal input is applied, the output of the system will also be a sinusoid of the same frequency, which is scaled and shifted. Therefore nearly the whole energy of the response to the input signal is concentrated at one frequency (the input frequency) whereas the energy of the noise is spread over several frequencies. This allows one to consider the system response isolated from the influence of noise.

3.4.2 Experimental Setup and Procedure

The whole experimental procedure and data processing is based on [5]. The application to this context as well as the code implementation is part of this thesis.
For this experiment the thrust produced by the Crazyflie was again measured using the load cell (Section 3.2.1).
However this time as the shift between input and output signal is essential to determine the transfer function, the time scales of the input commands given on the Crazyflie and the forces detected by the load cell had to be synchronized somehow. To achieve this a, script as well as a new Crazyflie firmware module were developed (see Appendix C.2.4). When launched, the script asks the load cell to start streaming data. Every incoming bit of data from the load cell is time stamped by the script. Subsequently the user is able to customize the sinusoidal input sequence (equation (3.6)) that the Crazyflie motors should receive. The variables that can be modified are the motor ID of the motor that should execute

the sequence, the length of the sequence N, the amplitude of the sinusoid A and the frequency determining integer l. The value of B was fixed to 30000.

$$u_{\mathrm{e}}[n] = A \cdot \cos(\Omega_l n) + B,\, \Omega_l = \frac{2\pi l}{N} \tag{3.6}$$

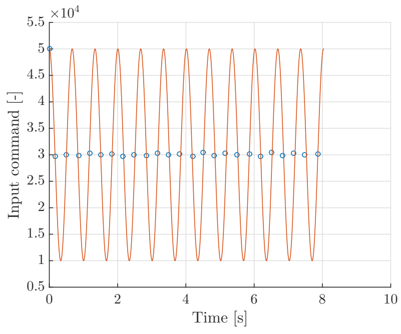
When this is finished, a packet containing these values is sent to the Crazyflie via the Crazyradio PA and the time at which the packet is sent is logged. As soon as the Crazyflie receives the packet it starts generating the input sequence and commanding it to the specified motor(s) with a frequency of 500 Hz. Every time the input sequence crosses the middle value B, a packet is sent from the Crazyflie back to the client who logs it together with a time stamp. Thanks to the time stamps that the client collects for the incoming force and input command data a temporal relationship between input and output of the motors is established.
This procedure was repeated for various values of l. In total two series of measurements were recorded. The parameters for both series can be found in Table 3.1.

Table 3.1: Parameters that were used for the two series of measurements. The increment of l was not constant but increasing.

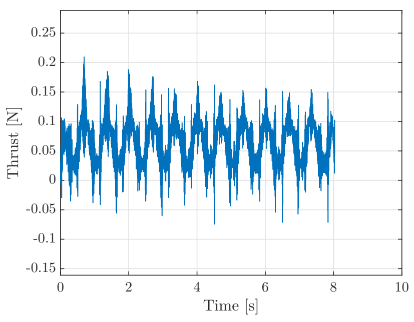
	1^{st} series of measurements	2^{nd} series of measurements
N	4000	8000
A	20000	20000
l	$[0, \ldots, 2000]$	$[0, \ldots, 4000]$
Motor #	1	3
# of measurements	43	40

3.4.3 Data Processing

This section describes the data processing that was done for every measurement (one set of input parameters) using the Matlab script in Appendix C.2.5.
The first processing step is to create a common time vector for the logged load cell and input command data. In addition, the input sequence is reconstructed based on the logging points that were sent back from the Crazyflie, the parameters that were passed to the Crazyflie for this measurement and the general formula for input sequences (equation (3.6)). An example of the data after this step is depicted in Figure 3.12.
Subsequently the FFT of both the input sequence and the load cell data were taken using Matlab. For this step the first N_{T} data points were neglected in order to only consider the part of the output signal without transient. For the first series of measurements, N_{T} was chosen to be 500 and for the second series 1800 because the second series has a larger number of data points N.
In addition, to keep the peaks of the FFT as sharp as possible the FFT was taken on a number of samples N_{FFT} that is as close to a multiple of the number



(a) The blue circles represent the logging packets that come back from the Crazyflie and the orange curve is the input sequence that was reconstruct as part of the post processing.



(b) Data that was logged from the load cell while applying the input sequence.

Figure 3.12: Data that was collected during one measurement in the first series of measurements. For this experiment l was chosen to be 12. The first processing step (synchronizing the time axes) and the reconstruction of the input signal were already performed.

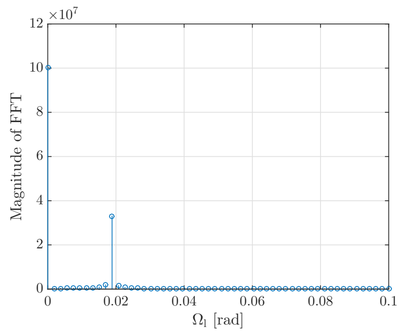
of samples of one period as possible. In [6] it is explained why this works: as long as the condition (adapted from [6])

$$k_0 \frac{2\pi}{N_{\mathrm{FFT}}} = \Omega_0 = \frac{2\pi l}{N} \tag{3.7}$$

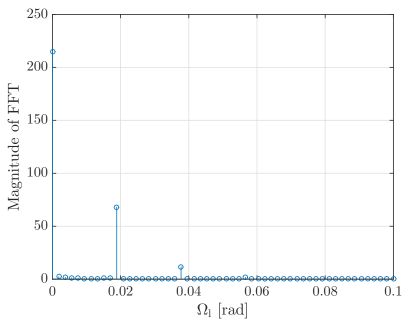
is met for an integer k_0 there exists an FFT coefficient at exactly the frequency of our signal. This FFT coefficient then captures the whole energy of the signal [6]. If the condition is not met, the energy would be spread over all FFT coefficients which is called leakage. Transforming equation (3.7) leads to

$$N_{\mathrm{FFT}} = \underbrace{\frac{N}{l}}_{\text{\# of samples of one period}} \cdot k_0 \tag{3.8}$$

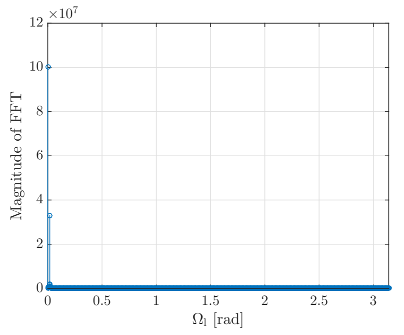
As there does not always exist a $k_0 < l$ such that this equation is satisfied, $\frac{N}{l}$ is rounded to the next integer and k_0 is chosen so that N_{FFT} is as close as possible to $N - N_{\mathrm{T}}$. Like this leakage is not always completely prevented but at least minimized.
The FFTs for our example measurement are depicted in Figure 3.13.
To verify their correctness one can calculate the discrete time frequency that corresponds to the parameters that were chosen for this sample experiment: $\Omega = \frac{2\pi l}{N} = \frac{2\pi \cdot 12}{4000} = 0.01884956\,\mathrm{rad}$. As there exist peaks at this value in the figures 3.13a and 3.13b it is now possible to say that they correctly represent the input/output frequency of our system.
Apart from these peaks there is a peak at $\Omega = 0\,\mathrm{rad}$ in both the input and the output FFT. These peaks are there because the middle value of both the input and output signal is greater than 0.



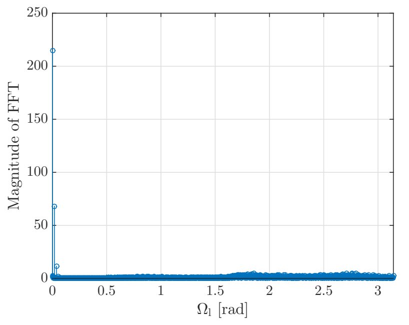
(a) FFT of the *input* command sequence in the interval $n = N_\mathrm{T}, \ldots, N_\mathrm{T} + N_\mathrm{FFT} - 1$ displayed for Ω_l between 0 and 0.1.



(b) FFT of the load cell *output* in the interval $n = N_\mathrm{T}, \ldots, N_\mathrm{T} + N_\mathrm{FFT} - 1$ displayed for Ω_l between 0 and 0.1.



(c) FFT of the *input* command sequence in the interval $n = N_\mathrm{T}, \ldots, N_\mathrm{T} + N_\mathrm{FFT} - 1$ displayed for Ω_l between 0 and π.



(d) FFT of the load cell *output* in the interval $n = N_\mathrm{T}, \ldots, N_\mathrm{T} + N_\mathrm{FFT} - 1$ displayed for Ω_l between 0 and π.

Figure 3.13: FFTs of the data depicted in Figure 3.12.

Furthermore there are two peaks visible in Figure 3.13b that belong to the harmonics of the output frequency and many peaks in Figure 3.13d on the right that correspond to high frequency noise of the load cell signal.

The next processing step is to compute a frequency response estimate using the FFT of the input signal at the input frequency $U_e[l]$ and the FFT of the output signal at the input frequency $Y_m[l]$ according to [5]

$$\widehat{H}(\Omega_l) = \frac{Y_m[l]}{U_e[l]}. \tag{3.9}$$

In order to finally get the transfer function

$$H(z) = \frac{\sum_{k=0}^{B-1} b_k z^{-k}}{1 + \sum_{k=1}^{A-1} a_k z^{-k}} \tag{3.10}$$

[5] explains to make use of the frequency response

$$H(\Omega) = \frac{\sum_{k=0}^{B-1} b_k e^{-j\Omega k}}{1 + \sum_{k=1}^{A-1} a_k e^{-j\Omega k}}. \tag{3.11}$$

After multiplying this equation with the denominator, a least squares problem can be formulated considering all measurements. The goal of the least squares problem is to minimize the error

$$\boldsymbol{e} = \boldsymbol{G} - \boldsymbol{H} \cdot \boldsymbol{\Theta}. \tag{3.12}$$

In this equation, $\boldsymbol{G}$ and $\boldsymbol{H}$ are composed using the frequency response and $\boldsymbol{\Theta}$ contains the unknown parameters. According to [5], the solution to the problem is

$$\boldsymbol{\Theta} = (\boldsymbol{F}^T \boldsymbol{F})^{-1} \boldsymbol{F}^T \boldsymbol{G}. \tag{3.13}$$

All least squares problems that occurred during this work were solved using this equation.
When determining the transfer function, the least squares problem was weighted: due to the relatively low sampling rate on the Crazyflie the higher frequency input sequences were considered to be generated less accurately. Therefore measurements with $l > 15$ for the first series of measurements and $l > 50$ for the second series were weighted with 0.5 while measurements with lower values of l were weighted with 2 for the first series and 2.5 for the second one. Values with $l > 100$ for the first series and $l > 200$ for the second one (both corresponding to a continuous time frequency of 12.5 Hz) were completely ignored. The solution to a weighted least squares problem is [5]

$$\boldsymbol{\Theta} = (\boldsymbol{F}^T \boldsymbol{W}^T \boldsymbol{W} \boldsymbol{F})^{-1} \boldsymbol{F}^T \boldsymbol{W}^T \boldsymbol{W} \boldsymbol{G}. \tag{3.14}$$

The results of the least squares problem are the coefficients of the transfer function.

3.4.4 Results and Discussion

The first series of measurements led to the transfer function $H_1(z)$ while the second series of experiments resulted in the transfer function $H_2(z)$.

$$H_1(z) = \frac{6.0705967 \cdot 10^{-8}}{1 - 0.9745210 \cdot z^{-1}} \tag{3.15}$$

$$H_2(z) = \frac{7.2345374 \cdot 10^{-8}}{1 - 0.9695404 \cdot z^{-1}} \tag{3.16}$$

Figures 3.14 and 3.15 depict the frequency response estimates as well as the fitted frequency responses for both series of measurements.

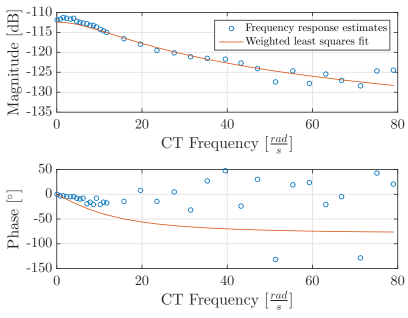


Figure 3.14: Result of the first series of measurements.

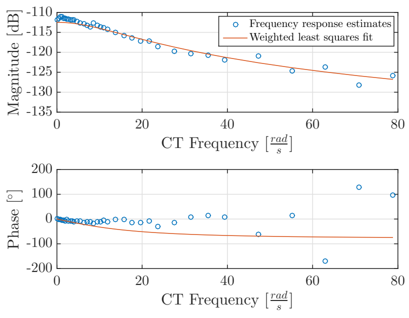


Figure 3.15: Result for the second series of measurements.

When looking at these figures what stands out is that the frequency response estimates vary a lot, especially at high frequencies and especially the phase. One of the reasons for this might be that the time scale synchronization between the input and output signal is not perfect because currently the latency of the signals is completely omitted.
The latency of the connection between Python client and Crazyflie was determined. For this a simple ping test was implemented as a Python script running on a computer and a new part of the Crazyflie's firmware (firmware module) (both in Appendix C.2.6). From the Python script packets are sent to the Crazyflie. Immediately after receiving a packet the Crazyflie returns an empty packet. The Python script logs the time between sending and receiving a packet. The results are summarized in Table 3.2. The average latency over all experiments is 57.1030 ms. As the input signal is generated onboard of the Crazyflie half of this value for the latency would have to be subtracted from all time stamps of logging packets coming in from the Crazyflie.
The latency of the load cell is more difficult to determine than the latency for the Crazyflie. For this reason it was not determined. As both latencies would have been subtracted from the corresponding time scales if they had been known, only taking the Crazyflie latency into account would have worsened the situation compared to when latencies are ignored. Therefore it was decided to not incorporate either latency into the calculation.
As the phase for the data from the second series of measurements exhibits slightly less outliers than the phase of the data from the first series, the second transfer function approximates the data more accurately. For this reason, it is recommended to use the second transfer function:

$$H(z) = H_2(z). \tag{3.17}$$

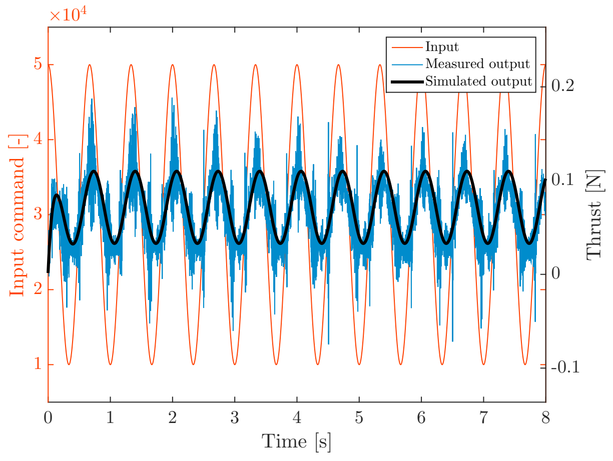
3.4.5 Verification

The transfer functions were verified by simulating them using Matlab's `lsim` command and comparing the simulated output to the measured one. Figure 3.16 demonstrates the result of this for the transfer function that was yielded from the first series of measurements. The first series was chosen because it is

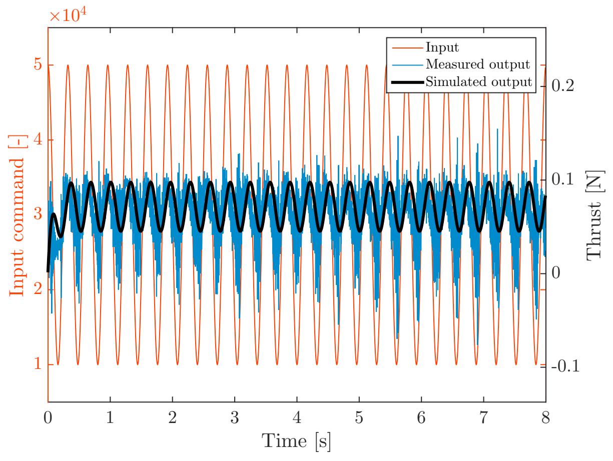
Table 3.2: Results for the ping test that was conducted to determine the latency of the connection between a script running on a computer and the Crazyflie. The table contains the times that were measured from the client, that is full roundtrip times (RTT).

Index	Circumstances	# packets	Mean RTT [ms]	Standard deviation [ms]
1	CF just started up	1000	58.2088	16.2416
2	2^{nd} experiment after startup	1000	56.9923	15.8395
3	3^{rd} experiment after startup	1000	57.2185	16.1054
4	CF just started up	500	58.1413	16.0421
5	2^{nd} experiment after startup	500	55.9051	15.8487
3	3^{rd} experiment after startup	500	56.1523	15.6315

interesting to see how the transfer function that is assumed to be less accurate performs.
For $l = 12$ the transfer function corresponds to the measured data very well. However for $l = 25$ a distinct shift between the simulated and the measured course can be seen. This error can most probably be accounted to the missing latency compensation.



(a) $l = 12$



(b) $l = 25$

Figure 3.16: Verification of the motor transfer function that was done by simulating its time domain response to an input ($N = 4000$, $A = 20000$) and comparing it to the measured output corresponding to the same input.

Chapter 4

Drag Coefficients

This chapter describes the process of determining the coefficients that represent the drag that the spinning propellers of the Crazyflie cause in an air flow.

4.1 Application and Theoretical Background

Application According to [13] the translational dynamics of a quadrocopter can be described by the equation

$$\begin{aligned} m\ddot{\boldsymbol{x}} &= \boldsymbol{R}(f\boldsymbol{e_3} + \boldsymbol{f_a}) + m\boldsymbol{g} \\ \dot{\boldsymbol{R}} &= \boldsymbol{R}[\![\boldsymbol{\omega}\times]\!] \end{aligned} \tag{4.1}$$

where $\boldsymbol{x}$ is the quadrocopter's position in an inertial reference frame, $\boldsymbol{R}$ is the rotation of the body frame with respect to the inertial frame, f is the total thrust produced by the rotors and $\boldsymbol{\omega}$ is the quadrocopter's angular velocity in the body frame. $\boldsymbol{f_a}$ denotes (also according to [13]) all aerodynamic forces apart from f and can be calculated as

$$\boldsymbol{f_a} = \boldsymbol{K_{aero}}\dot{\theta}_\Sigma \boldsymbol{R^{-1}}\dot{\boldsymbol{x}} \tag{4.2}$$

with $\boldsymbol{K_{aero}} = diag(\kappa_\perp, \kappa_\perp, \kappa_\parallel)$ being the constant drag coefficients and $\dot{\theta}_\Sigma = \sum_{i=1}^{4}|\dot{\theta}_i|$ the sum of the angular velocities of all rotors.
Under the assumption that there is no wind, equation (4.2) can be used to estimate the current vehicle speed in the body frame solely based on accelerometer data. When transforming equation (4.1) it becomes more clear why this works:

$$\boldsymbol{f_a} = m \underbrace{\boldsymbol{R^{-1}}(\ddot{\boldsymbol{x}} - \boldsymbol{g})}_{\text{accelerations in body frame}} - f\boldsymbol{e_3} \tag{4.3}$$

The quadrocopter's accelerometer measures all accelerations in the body frame as highlighted in the equation above. The vehicle mass m is known and the produced thrust can be calculated with the formula $f = \sum_{i=1}^{4} \kappa\dot{\theta}_i^2$ [13] if the angular velocities of the rotors can be measured. If the angular velocities are unknown f could for example be determined using a mapping from thrust to input command (Section 3.3.1). Subsequently the result for $\boldsymbol{f_a}$ can be plugged into the following equation, a transformed version of equation (4.2):

$$\boldsymbol{R^{-1}\dot{x}} = \frac{1}{\dot{\theta}_\Sigma}\boldsymbol{K_{aero}^{-1}f_a} \tag{4.4}$$

which gives as a result the desired speed of the quadrocopter in the body frame $\boldsymbol{R^{-1}\dot{x}}$.

Theoretical Background This section explains the origin of the drag force $\boldsymbol{f_a}$.
There exists a large number of aerodynamic effects that cause forces on a spinning rotor. However according to [11], apart from the one that is responsible for the produced thrust, there are two major effects: induced drag and blade flapping. Both of them appear when the quadrocopter is moving in the air and both of them can cause the total thrust to have a component that is parallel to the rotor plane.

Induced drag occurs as soon as a rotor wing is moving in the air. As explained in [9] it is caused because there is a higher pressure under the wing than above the wing which leads to the formation of vortices at the wing tips. These vortices lead to air flowing down behind the wing with velocity w. This effect is called downwash and it influences the effective incident velocity U_e as is shown in Figure 4.1. According to the lift theorem introduced by Kutta and Zhukovsky the lift force L of a wing is proportional and perpendicular to U_e [9]. As U_e has a vertical component downwards, L will have a horizontal component backwards. This component is called induced drag.

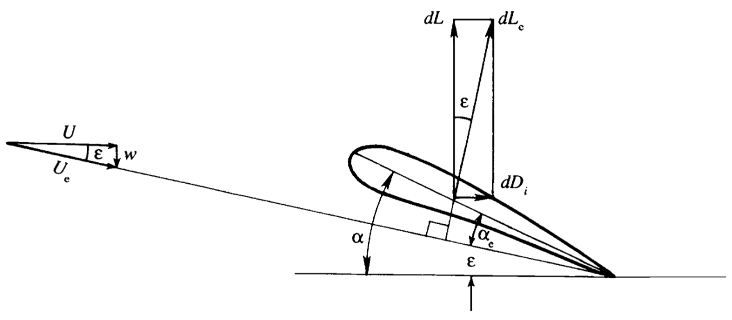


Figure 4.1: From Kundu, Cohen et al. [9] (p. 712). Relationships between wind velocity U, downwash w, effective incident velocity U_e, and effectively generated lift L_e.

Now for a vehicle that has rotating wings such as a quadrocopter, these induced drag components have equal magnitudes for all points on the circumference of a rotor [11] (see Figure 4.2). Therefore no resulting force but only a torque around the rotor axis is generated (note that this torque is used to control the yaw angle of a quadrocopter). But as soon the quadrocopter moves in one direction the advancing blade of the motor feels a higher relative air speed than the retreating blade. As the induced drag is proportional to the lift which is proportional to the relative air speed, the induced drag will increase for the advancing blade and decrease for the retreating blade (see Figure 4.3). As a consequence there will be a resulting force opposing the direction of travel.

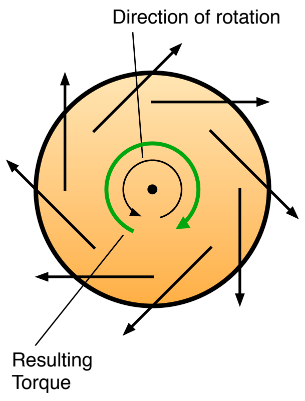


Figure 4.2: Spinning rotor during hover. The induced drag is equally spread around the circumference. There is no resulting drag force but only a torque about the rotor axis.

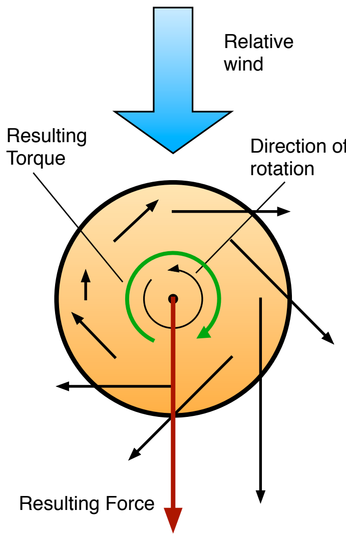


Figure 4.3: Spinning rotor while quadrocopter is translating. Induced drag is increased for the advancing and decreased for the retreating blade, leading not only to the torque that is felt during hover but also to a resulting force pointing opposite to the direction of travel.

Whereas the phenomenon of induced drag is based on the fact that wings are rather rigid, blade flapping only can occur when the rotor wings have a certain degree of flexibility [11]. Again this effect is based on the fact that the advancing blade has a higher tip speed than the retreating one. Due to this higher speed the lift force on the advancing blade is also increased whereas the lift force of the retreating blade is decreased. The result is a torque on the rotor that points in the opposite direction of the vehicle's velocity (for counterclockwise spinning rotors). However because the rotor spins at a high angular velocity and therefore has a high angular momentum, it acts like a gyroscope. For this reason, the attacking torque shows its effect in a direction that is rotated by 90° with respect to the original one [11]. Consequently the rotor tilts backwards. As the lift force is perpendicular to the rotor plane, it is also tilted backwards and now has a drag component. This effect is partially extenuated because, due to its up flapping, the advancing blade has a reduced angle of attack which decreases the lift and therefore the torque, which is responsible for the flapping. This results in an established equilibrium [11]. The geometrical situation for this effect is depicted in Figure 4.4.

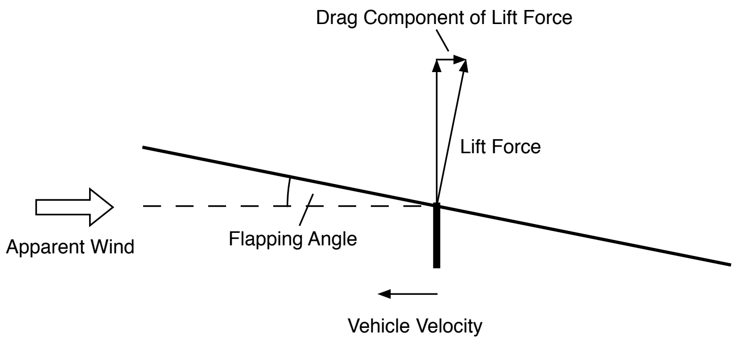


Figure 4.4: Adapted from Mahony, Kumar and Corke [11]. The apparent wind induces a torque on the rotor plane which causes it to tilt away from the apparent wind. The component of the lift force that points in the opposite direction of the vehicle velocity is the drag that is caused by blade flapping.

4.2 Equipment

During this experiment for determining the drag coefficients, apart from measuring forces, using the load cell (Section 3.2.1), and rotor angular velocities, using the laser tachometer (Section 3.2.2), wind speeds had to be measured frequently. For this a hand-handled optical vane anemometer was used. It can measure wind velocities of up to 20 m/s with a varying resolution between 0.05 m/s for slow wind speeds and 0.5 m/s for faster wind speeds. The device is depicted in Figure 4.5.

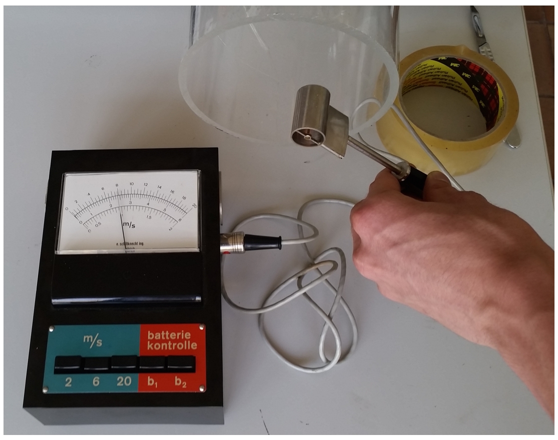


Figure 4.5: Anemometer that was used to determine wind velocities.

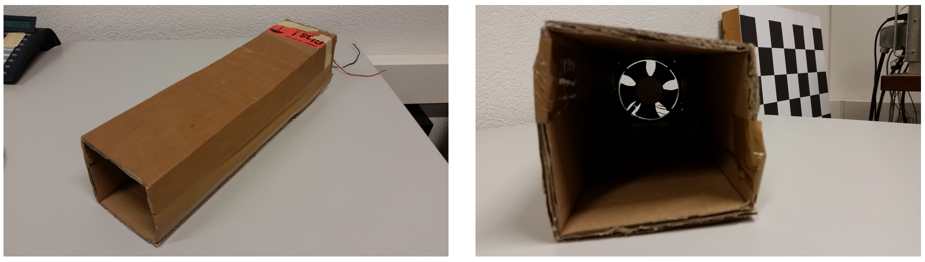
4.3 Design Considerations

4.3.1 Computer Vision

The first approach to determine the drag coefficients $\kappa_\perp$ and $\kappa_\parallel$ experimentally involved making use of the IDSC's flying machine arena (FMA), more specifically its global sensing system Vicon that is described in more detail in [10]. With the Vicon system it is possible to track the position and orientation of an object very accurately. Therefore the drag coefficients of a quadrocopter can be determined by keeping track of position $\boldsymbol{x}$ and its derivatives as well as the rotation $\boldsymbol{R}$ and its derivative with Vicon while also logging f and $\dot{\theta}_\Sigma$ onboard during a quadrocopter's random flight in the FMA. Subsequently, equations (4.1) and (4.2) can be used to formulate a least squares problem of which the drag coefficients result.
However this is unfortunately not possible for the Crazyflie. Due to its small size only one marker with the sufficient area (circle with a diameter of at least 2 cm) could be attached to the Crazyflie's battery. Yet for keeping track of the orientation, Vicon requires a minimum of three markers. More markers could have been mounted to the Crazyflie but not without also mounting a construction to attach them. Such a construction would have influenced the Crazyflie's dynamics, especially the drag, significantly which disqualified this method for the purpose of determining the drag coefficients.

4.3.2 Wind Generator

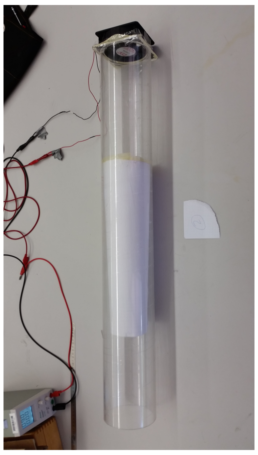
The idea that was developed as an alternative was to keep the Crazyflie stationary while simulating the quadrocopter's translation through the air by blowing air onto it and measuring all attacking forces using the load cell (see Section 3.2.1).
In order to prove that this method would work in principle a rapid prototype was built from carton. It is depicted in Figure 4.6 and features a 12V DC powered fan. With this fan the wind speeds across the opening varied between 2.25 and 4.25 $^{\mathrm{m}}/_{\mathrm{s}}$. However, it was still possible to determine drag coefficients with a correct sign.


(a) View from the outside.

(b) View of the opening where the generated wind comes out. The fan is well recognizable.

Figure 4.6: Paper prototype for a wind generator that was built using standard carton, scotch tape and a fan.

Nonetheless in order to get more accurate results the design was improved iteratively. Firstly while the fan was still used the carton was replaced by a PMMA pipe with a diameter of approximately 15 cm. The problem with this design was that the speed of the air flow at the exit still varied a lot. Especially in the center, where the speed was about half as fast as it was along the circumference. This was most probably due to the large motor housing in the center of the fan. In order to generate a more homogenous flow, firstly only a funnel with decreasing diameter was added based on [3] (Figure 4.7a). But because the flow was still very inhomogeneous - most likely due to turbulence - a honeycomb was introduced as well. The final composition of the wind generator is depicted in Figure 4.7b. The next section contains more information on the experimental design and procedure.


(a) Paper funnel inside the tube to increase the flow speed in the center of the opening.

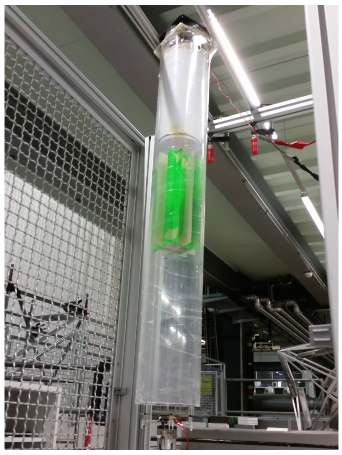

(b) Additional honey comb to reduce turbulance and to make the flow as homogenous as possible.

Figure 4.7: PMMA tube with fan and two different configurations inside.

4.4 Experiment

4.4.1 Experiment Design

Figure 4.8 shows the final experimental setup while the geometrical situation is depicted in Figure 4.9. The Crazyflie was mounted to the load cell and supplied by a constant 3.7 V DC power supply in order to have constant conditions for all experiments. The anemometer and the tachometer were in place to take measurements. The wind generator was mounted to the load cell cage and powered by a second power supply with up 12 V DC. This voltage was used to control the velocity of the air flow. What angle wind was blown onto the Crazyflie from could be varied by fixing the wind generator to the cage in different heights using cable ties. For this experiment the angles 0°, 48.4° and 77.9° with respect to a vertical position were chosen because fixing the generator in these positions was convenient.

The load cell data was recorded using the logger in Appendix C.2.2 and input commands were given to the Crazyflie's motors using the code in Appendix C.2.1.

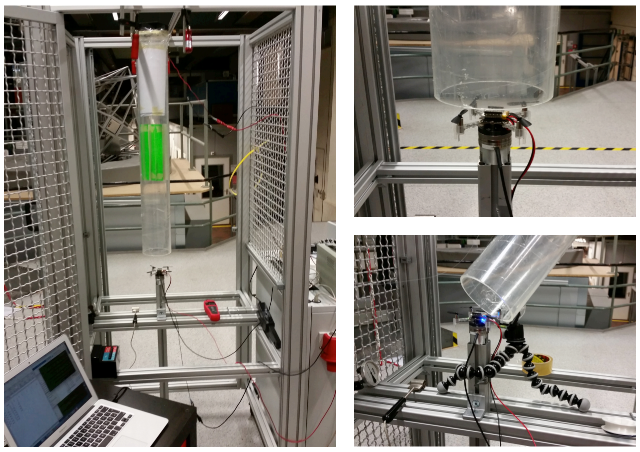


Figure 4.8: Experimental setup that was used to determine the drag coefficient of the Crazyflie. The quadrocopter is mounted to the load cell and powered using a 3.7 V DC power supply. The wind generator can be mounted to the load cell cage in 3 different angles. The tachometer for measuring rotor angular velocity while wind is blown onto the Crazyflie and the anemometer used to measure the flow speed around the Crazyflie are prepared.

4.4.2 Experimental Procedure

After several iterations on the experimental procedure it was decided to do a total of 112 experiments: seven different input commands to the Crazyflie motors (0, 10000, 20000, 30000, 37300, 40000 and 50000), three wind angles and for every wind angle five fan voltages (5 V, 6 V, 8 V, 10 V and 12 V). In addition, one experiment for every input command was conducted without wind blowing onto the Crazyflie.
During the experiments the following procedure was followed.

1. Mount the wind generator at one of the three angles.
   (a) Set a wind speed by adjusting the voltage.
   (b) Measure the wind speed around the Crazyflie.
      i. Set a motor input command for all motors on the Crazyflie.
      ii. Measure one motor's angular velocity.
      iii. Start the load cell measurement and wait for about 10 s.
      iv. Stop the load cell measurement and turn the motors off.
      v. Start again from i. until all input commands were measured.
   (c) Start again from (a) until all wind speeds were measured.
2. Start again from 1. until all measurements for all three angles were taken.

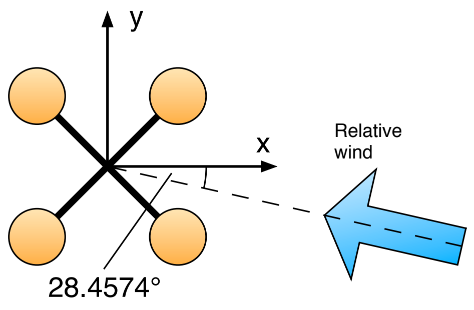


Figure 4.9: Geometrical situation depicting the direction the wind is blowing from in the x-y plane. The angle was determined based on length and angle measurements in the load cell cage using a folding rule and a triangle ruler.

The wind speed was always measured all around the Crazyflie. The maximum and minimum wind speed was logged.
The results of proceeding like this were 112 force vectors, each one acting on the Crazyflie during one experiment under a specific combination of input command and wind speed.

4.4.3 Data Analysis

The data collected during the experiments was processed using the Matlab script from Appendix C.3.1.

Data Preparation After reading in the data from a file the first step was to remove the influences from sensor bias of the load cell, gravity and parasite drag (caused by wind blowing onto all surfaces of the Crazyflie). This was done by subtracting the average of the first 500 data points from all data points. As described in the experimental procedure section wind was already blowing on the Crazyflie when the load cell measurement was started. This is the reason why the effect of parasite drag is removed when subtracting the average value of the first few data points.
Subsequently in order to only end up with one value for every force component the corresponding data points were averaged over a manually chosen interval (as for example described in Section 3.2.1). This resulted in a single force vector for each combination of input command, wind velocity and wind angle. However this force does still not solely represent the drag force that is of interest, since the produced thrust is still a part of it. To eliminate the thrust's influence, the steady state thrust from the measurements without wind blowing onto the Crazyflie and with a certain input command was subtracted from all forces that were measured during experiments with the same input command. Like this, the pure drag force was obtained.
Equation (4.2) that is used to determine the drag coefficients contains, apart from the drag coefficients and the drag force, the sum of the rotors' angular velocities and the vehicle speed expressed in the body frame. The sum of the angular velocities is obtained using the RPM that was measured for each com-

bination of input commands and wind speeds. The vehicle speed is calculated from the measured wind speeds in combination with the orientation of the wind generator. In order to get only one wind speed per measurement the maximum and minimum values that were logged were averaged.

The next step was calculating estimates for the drag coefficients. This was done in three different ways: firstly fitting the simple drag force model given by equation (4.2) to the data, secondly fitting a more complex drag force model to the data and thirdly fitting the simple model with a full drag coefficient matrix instead of just a diagonal one to the data.

Simple Model In order to uncover any dependencies of the drag coefficients for the x and y direction they were fitted in different ways: firstly they were calculated separately using equation (4.2) resulting in values for this drag coefficient for x and y direction and for every combination of input command and wind velocity/angle separately.
Secondly one drag coefficient value was calculated for every combination of input command and wind velocity/angle. This was done by formulating and solving a least squares problem.
Thirdly one drag coefficient was calculated for all wind speeds resulting in values for every combination of wind angle and input command.
And finally one drag coefficient was fitted to all data at once.

The drag coefficient for the z direction was calculated separately for every experiment. To get only one value the average of the calculated ones can be taken.

Complex Model To cover more of the possible dependencies on rotor angular velocity and vehicle speed than those already covered by the simple model, a more complex model was introduced. Instead of equation (4.2), the following equation is employed:

$$\boldsymbol{f_a} = \boldsymbol{K_{aero,1}}\dot{\theta}_\Sigma \boldsymbol{v} + \boldsymbol{K_{aero,2}}(\boldsymbol{v}).\wedge 2 + \boldsymbol{K_{aero,3}}\dot{\theta}_\Sigma^2 + \boldsymbol{K_{aero,4}}\boldsymbol{v} + \boldsymbol{K_{aero,5}}\dot{\theta}_\Sigma + \boldsymbol{K_{aero,6}} \tag{4.5}$$

where $\boldsymbol{v} = \boldsymbol{R^{-1}}\dot{\boldsymbol{x}}$, $\boldsymbol{K_{aero,i}} \in \mathbb{R}^3$ for $i = 3, 5, 6$ and $\boldsymbol{K_{aero,i}} \in \mathbb{R}^{3\times 3}$ for $i = 1, 2, 4$. $().\wedge 2$ denotes the element-wise square of a vector.
It was fitted to the data using least squares. In contrast to the simple fit it was not calculated one coefficient per experiment (single combination of motor input command, air flow velocity and angle) but only one coefficient fitting the whole data as well as possible.

Cross-coupling Model This model was introduced to investigate whether there exist couplings between the different axes. An example for coupling would be a drag force in z direction although the Crazyflie is only translating in x direction. According to the simple model there should be no drag force then.

Cross-couplings were determined using a similar equation as in the simple model:

$$\boldsymbol{f_a} = \boldsymbol{K_{aero,full}}\dot{\theta}_\Sigma \boldsymbol{R^{-1}}\dot{\boldsymbol{x}}. \tag{4.6}$$

However to model the cross-couplings, a full matrix with nine coefficients was fitted. For the fitting, certain symmetries were expected: the drag coefficients as well as the cross coupling between x and y axis should be the same. Due to symmetries, a wind velocity in z direction causes the same force in x and y direction. Apart from that, a drag force in z direction caused by a velocity in x direction should be the same as a drag force in z direction caused be the same velocity in y direction. For these reasons, the matrix that was fitted has the following form:

$$\boldsymbol{K_{aero,full}} = \begin{pmatrix} \kappa_1 & \kappa_2 & \kappa_3 \\ \kappa_2 & \kappa_1 & \kappa_3 \\ \kappa_4 & \kappa_4 & \kappa_5 \end{pmatrix}. \tag{4.7}$$

Due to symmetry aspects, one would expect κ_2 as well as κ_3 to be rather small. The matrix was also fitted to the data using least squares.

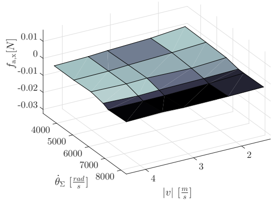
4.5 Results and Discussion

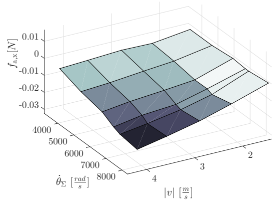
Figure 4.10 depicts the drag force data that was yielded from the experiments and prepared as described above. From the plots it is possible to see that the collected data is as expected from the model: the absolute value of every drag force component grows for increasing rotor angular velocity and wind velocity. Together with Figure 4.9 the signs of the components also become obvious. The Crazyflie's y axis is nearly orthogonal to the direction of the wind. For this reason the magnitude of the force in y direction is small compared to the other directions.

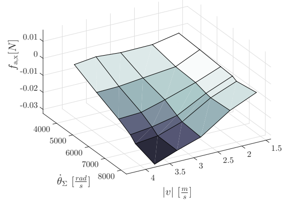
From fitting the simple model to the data, the expected results for all experiments were to get a constant drag coefficient $\kappa_\parallel$ and constant and equal drag coefficient $\kappa_\perp$. However a look at the results in Figure 4.11 shows that this is not the case. Instead some trends can be observed: $\kappa_\parallel$ is decreasing whereas $\kappa_\perp$ increasing for increasing rotor angular velocities. In addition both drag coefficients decrease for increasing relative wind speeds. Those trends were the reason why the more complex model was fitted to the data. When solving the least squares problem for all experiments for the drag coefficient in x and y direction and taking the average of all calculated drag coefficients in z direction the following matrix results:

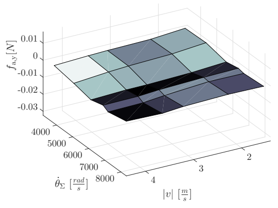
$$\boldsymbol{K_{aero}} = diag\left(-9.1785 \cdot 10^{-7}, -9.1785 \cdot 10^{-7}, -10.311 \cdot 10^{-7}\right) \text{ kg} \cdot \text{rad}^{-1} \tag{4.8}$$

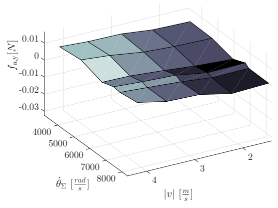
The drag coefficients resulting from fitting the more complex function to the

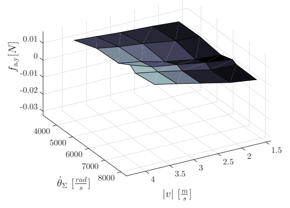

(a) Wind from 0°, x axis

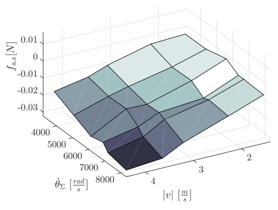

(b) Wind from 48.4°, x axis

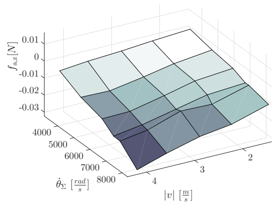

(c) Wind from 77.9°, x axis


(d) Wind from 0°, y axis


(e) Wind from 48.4°, y axis


(f) Wind from 77.9°, y axis


(g) Wind from 0°, z axis


(h) Wind from 48.4°, z axis

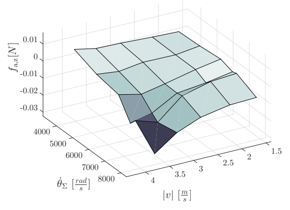
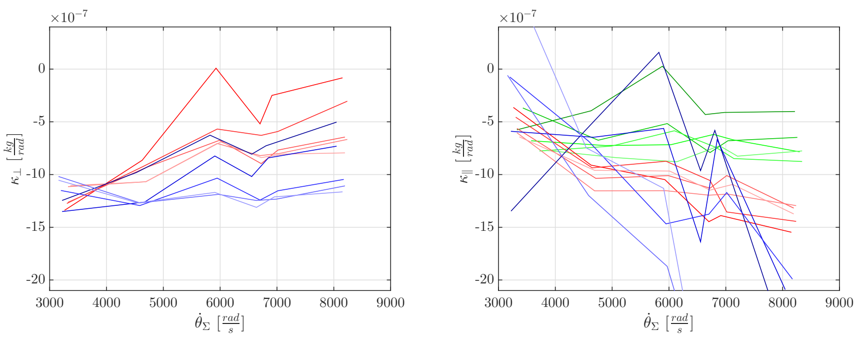

(i) Wind from 77.9°, z axis

Figure 4.10: Drag force data that resulted from the experiments. Each plots shows one component of the drag force plotted over all wind velocities and rotor angular velocities. $|v|$ is the absolute value of the wind velocity and $\dot{\theta}_\Sigma$ is the sum of all rotor angular velocities.



(a) Drag coeffiecient for x and y direction. (b) Drag coefficient for z direction.

Figure 4.11: Drag coefficients calculated from the experiment data. Least squares was applied to fit one drag coefficient for both x and y direction. Green curves show results from experiments where wind was blowing from above, red represents wind from 48.4° and blue from 77.9°. The lighter the curves, the higher the wind velocity.

data are

$$
\begin{aligned}
\boldsymbol{K_{aero,1}} &= diag\left(-8.9158 \cdot 10^{-7}, -8.9158 \cdot 10^{-7}, -3.1574 \cdot 10^{-7}\right) \text{ kg} \cdot \text{rad}^{-1} \\
\boldsymbol{K_{aero,2}} &= diag\left(-6.8572 \cdot 10^{-5}, -6.8572 \cdot 10^{-5}, 5.2068 \cdot 10^{-5}\right) \text{ kg} \cdot \text{rad}^{-1} \\
\boldsymbol{K_{aero,3}} &= \left(1.1254 \cdot 10^{-10}, 1.1254 \cdot 10^{-10}, -2.3662 \cdot 10^{-10}\right)^{T} \text{ kg} \cdot \text{rad}^{-1} \\
\boldsymbol{K_{aero,4}} &= diag\left(-5.4233 \cdot 10^{-4}, -5.4233 \cdot 10^{-4}, -2.2935 \cdot 10^{-3}\right) \text{ kg} \cdot \text{rad}^{-1} \\
\boldsymbol{K_{aero,5}} &= \left(-1.8443 \cdot 10^{-6}, -1.8443 \cdot 10^{-6}, 4.8857 \cdot 10^{-7}\right)^{T} \text{ kg} \cdot \text{rad}^{-1} \\
\boldsymbol{K_{aero,6}} &= \left(6.4061 \cdot 10^{-3}, 6.4061 \cdot 10^{-3}, 3.0985 \cdot 10^{-3}\right)^{T} \text{ kg} \cdot \text{rad}^{-1}
\end{aligned}
\tag{4.9}
$$

The matrix $\boldsymbol{K_{aero}}$ in the complex model corresponds to the matrix $\boldsymbol{K_{aero,1}}$ in the more simple model. Both matrices' coefficients are similar. However apart from this observation it is difficult to draw a conclusion from the results as they are right now because in equation (4.5) the coefficients are multiplied with values of very different magnitudes. In order to render the coefficients more comparable they were multiplied with the mean of all measured variables they belong to and divided by the sum of all means (see code in Appendix C.3.1 for details on this calculation). The following weighted parameters resulted from this:

$$\begin{aligned}
\boldsymbol{K_{aero,norm,1}} &= diag\left(-5.2370\cdot10^{-7}, -5.2370\cdot10^{-7}, -1.8546\cdot10^{-7}\right)\\
\boldsymbol{K_{aero,norm,2}} &= diag\left(-1.9155\cdot10^{-8}, -1.9155\cdot10^{-8}, 1.4545\cdot10^{-8}\right)\\
\boldsymbol{K_{aero,norm,3}} &= \left(2.3195\cdot10^{-11}, 2.3195\cdot10^{-11}, -4.8768\cdot10^{-11}\right)^T\\
\boldsymbol{K_{aero,norm,4}} &= diag\left(-5.3157\cdot10^{-8}, -5.3157\cdot10^{-8}, -22.4798\cdot10^{-8}\right)\\
\boldsymbol{K_{aero,norm,5}} &= \left(-3.8011\cdot10^{-7}, -3.8011\cdot10^{-7}, 1.0069\cdot10^{-7}\right)^T\\
\boldsymbol{K_{aero,norm,6}} &= \left(2.2032\cdot10^{-7}, 2.2032\cdot10^{-7}, 1.0656\cdot10^{-7}\right)^T
\end{aligned} \tag{4.10}$$

Now it is visible that in equation (4.5) the terms with $\boldsymbol{K_{aero,1}}$, $\boldsymbol{K_{aero,5}}$ and $\boldsymbol{K_{aero,6}}$ have the most significant influence. The influence of the term with $\boldsymbol{K_{aero,5}}$ could explain the trend in the drag coefficients, however at the moment there is no explanation for the strong influence of the term with $\boldsymbol{K_{aero,6}}$.

The fit of the cross-coupling model yielded the following matrix:

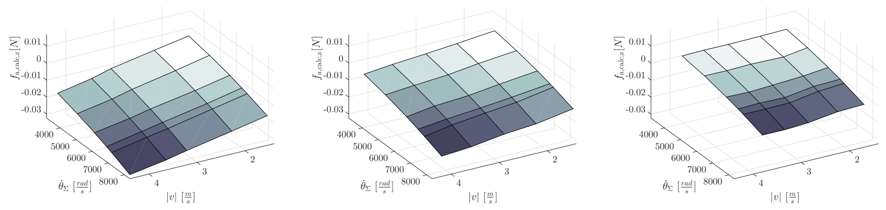
$$\boldsymbol{K_{aero,full}} = \begin{pmatrix} -10.2506 & -0.3177 & -0.4332\\ -0.3177 & -10.2506 & -0.4332\\ -7.7050 & -7.7050 & -7.5530 \end{pmatrix} \cdot 10^{-7}\,\mathrm{kg}\cdot\mathrm{rad}^{-1} \tag{4.11}$$

This result shows that the interdependency of velocity in x direction and drag force in y direction as well as vice versa is rather weak. Also, a velocity in z direction does not cause a strong drag force in the x-y plane. So far, this is as expected.
In contrast, the magnitude of the coefficients describing the effect of a velocity in the x-y plane on a drag force in z direction is large. This could be explained with the effects described in Section 4.1: the lifting force is tilted backwards which increases the horizontal component (drag) and decreases the vertical component (lift). This decrease of lift could also be regarded as drag.
However one has to pay attention with the signs as they are at the moment: the questionable coefficients are $-7.7050 < 0$. This means that a translation in positive x direction causes a negative drag force in z direction (which is correct) but a translation in negative x direction causes a drag force in positive z direction. This cannot be correct. In order to account for this, a new model would have to be introduced.

4.6 Verification

To verify that the calculated drag coefficients from the complex fit indeed fit the drag force data, the drag force was calculated using equation (4.5), the drag coefficients and the available data for the rotor angular velocity and the wind velocity. The result for the z component is depicted in Figure 4.12 and is similar to the experiment data in Figure 4.10. The sum of the squared errors between experimentally determined drag forces and calculated drag forces is $0.0022072\,\mathrm{N}^2$. When calculating back the forces using the drag coefficients from the simple fit the plot is nearly indistinguishable from the plots in Figure 4.12.

However the sum of the squared errors is $0.0035997\,\mathrm{N}^2$ which means that the data from the complex fit does fit the measured data better. Yet, the sum of the squared errors is the least for the cross-coupling model: $0.0019127\,\mathrm{N}^2$.



(a) Wind from 0° (b) Wind from 48.4° (c) Wind from 77.9°

Figure 4.12: Drag force data that is calculated back from the drag coefficients resulting from the complex fit. For brevity reasons only one component of the drag force is plotted over all wind velocities and rotor angular velocities. $|v|$ is the absolute value of the wind velocity and $\dot{\theta}_\Sigma$ is the sum of all rotor angular velocities.

Another experiment to verify the drag coefficients was not designed due to the complexity of this task. Instead the quality of the drag coefficients will be measured indirectly through their performance as a part of the new estimator that is going to be implemented on the Crazyflie.

Chapter 5

Conclusion & Outlook

During this work physical parameters that are needed to implement new model based estimation and control for the Crazyflie were determined experimentally. The inertia matrix was measured by observing and analyzing the Crazyflie's pendulum motion. Estimates for the moments of inertia were calculated by fitting the solution of the linearized equation of motion to the collected data. The method was verified by measuring the moments of inertia of a test body and comparing them to said analytically determined moments of inertia.
Several motor mappings between motor input command, produced thrust and torque as well as the rotor's angular velocities were defined to enable the controller to give appropriate input commands to the motors. Additionally a motor transfer function between motor input command and produced thrust was found. It can be used for simulation purposes. All data for these experiments were collected using a load cell and a tachometer.
Finally the Crazyflie's drag coefficients were determined by blowing air onto the Crazyflie, powering its motors and measuring the resulting forces with a load cell. The drag coefficients can be used to estimate the vehicle's velocity based on accelerometer data. For these experiments the load cell, a tachometer and an anemometer were used.

Of course all experimental methods have the potential to be improved with the goal to get more accurate results for the Crazyflie's physical parameters:

Inertia Matrix The problems with the friction could be analyzed in more depth. Also the friction could be reduced allowing operation at lower angles which would improve the validity of using a linearized model.

Motor Parameters For determining the transfer function a load cell with higher resolution and less noise could be used to improve the signal-to-noise ratio of the collected data. This would facilitate the further analysis.

Drag Coefficients The wind generator could be improved to provide a more homogenous stream of air. Then the results for the drag coefficients should vary less and be more accurate.

However, determining the physical parameters is not the final goal of this research. For this reason the most promising next step would be implementing the new estimator and controller on the Crazyflie and evaluating the overall system performance.

Acknowledgements

During the course of this work many people supported me and I would like to thank them. First of all, of course, my supervisor Mike. It was really a pleasure to work with him and he provided me with very helpful advice. Always being able to come to his office and ask questions was an invaluable help.
Furthermore I would like to thank everyone from IDSC as well as my friends who helped me a lot by discussing the problems that occurred when designing and evaluating the experiments with me.
And finally a big thank you to my parents who rendered my studies in Zurich possible.

Appendix A

Overview Results

This appendix is intended for readers who are mainly interested in the numeric results of this work. It summarizes the results from the experiments that are described above.

Inertia Matrix

Mass of the Crazyflie	$m = 28.0\,\mathrm{g}$
Inertia Matrix	$\boldsymbol{I_{CF}} = \begin{pmatrix} 16.571710 & 0.830806 & 0.718277 \\ 0.830806 & 16.655602 & 1.800197 \\ 0.718277 & 1.800197 & 29.261652 \end{pmatrix} \cdot 10^{-6}\mathrm{kg} \cdot \mathrm{m}^2$

Motor Parameters

Mappings

Input Command $\rightarrow$ Thrust	$f_\mathrm{i} = 2.130295 \cdot 10^{-11} \cdot cmd_\mathrm{i}^2 + 1.032633 \cdot 10^{-6} \cdot cmd_\mathrm{i} + 5.484560 \cdot 10^{-4}$
Input Command $\rightarrow$ Rotor Angular Velocity	$\theta_\mathrm{i} = 0.04076521 \cdot cmd_\mathrm{i} + 380.8359$
Thrust $\rightarrow$ Torque	$\tau_\mathrm{i} = 0.005964552 \cdot f_\mathrm{i} + 1.563383 \cdot 10^{-5}$

Transfer Function

$$H(z) = \frac{7.2345374 \cdot 10^{-8}}{1 - 0.9695404 \cdot z^{-1}}$$

Drag Coefficients

Simple Model

Model equation [13]	$\boldsymbol{f_a} = \boldsymbol{K_{aero}} \dot{\theta}_\Sigma \boldsymbol{R^{-1}} \dot{\boldsymbol{x}}$
Drag coefficients	$\boldsymbol{K_{aero}} = diag\left(-9.1785 \cdot 10^{-7}, -9.1785 \cdot 10^{-7}, -10.311 \cdot 10^{-7}\right)\ \mathrm{kg} \cdot \mathrm{rad}^{-1}$

Complex Model

Model equation $\boldsymbol{f_a} = \boldsymbol{K_{aero,1}}\dot{\theta}_\Sigma \boldsymbol{v} + \boldsymbol{K_{aero,2}}(\boldsymbol{v}).\wedge 2 + \boldsymbol{K_{aero,3}}\dot{\theta}_\Sigma^2 + \ldots$
$\boldsymbol{K_{aero,4}}\boldsymbol{v} + \boldsymbol{K_{aero,5}}\dot{\theta}_\Sigma + \boldsymbol{K_{aero,6}}$

Drag coefficients

$$
\begin{aligned}
\boldsymbol{K_{aero,1}} &= diag\left(-8.9158 \cdot 10^{-7}, -8.9158 \cdot 10^{-7}, -3.1574 \cdot 10^{-7}\right)\ \mathrm{kg} \cdot \mathrm{rad}^{-1} \\
\boldsymbol{K_{aero,2}} &= diag\left(-6.8572 \cdot 10^{-5}, -6.8572 \cdot 10^{-5}, 5.2068 \cdot 10^{-5}\right)\ \mathrm{kg} \cdot \mathrm{rad}^{-1} \\
\boldsymbol{K_{aero,3}} &= \left(1.1254 \cdot 10^{-10}, 1.1254 \cdot 10^{-10}, -2.3662 \cdot 10^{-10}\right)^T\ \mathrm{kg} \cdot \mathrm{rad}^{-1} \\
\boldsymbol{K_{aero,4}} &= diag\left(-5.4233 \cdot 10^{-4}, -5.4233 \cdot 10^{-4}, -2.2935 \cdot 10^{-3}\right)\ \mathrm{kg} \cdot \mathrm{rad}^{-1} \\
\boldsymbol{K_{aero,5}} &= \left(-1.8443 \cdot 10^{-6}, -1.8443 \cdot 10^{-6}, 4.8857 \cdot 10^{-7}\right)^T\ \mathrm{kg} \cdot \mathrm{rad}^{-1} \\
\boldsymbol{K_{aero,6}} &= \left(6.4061 \cdot 10^{-3}, 6.4061 \cdot 10^{-3}, 3.0985 \cdot 10^{-3}\right)^T\ \mathrm{kg} \cdot \mathrm{rad}^{-1}
\end{aligned}
$$

Cross-coupling Model

Model equation $\boldsymbol{f_a} = \boldsymbol{K_{aero,full}}\dot{\theta}_\Sigma \boldsymbol{R^{-1}}\dot{\boldsymbol{x}}$

Drag coefficients

$$
\boldsymbol{K_{aero,full}} = \begin{pmatrix} -10.2506 & -0.3177 & -0.4332 \\ -0.3177 & -10.2506 & -0.4332 \\ -7.7050 & -7.7050 & -7.5530 \end{pmatrix} \cdot 10^{-7}\,\mathrm{kg} \cdot \mathrm{rad}^{-1}
$$

Appendix B

Mathematics

B.1 Derivation of the Solution to the Equation of Motion of a Pendulum

The equation of motion for the general pendulum motion is

$$I_\mathrm{a}\ddot{\varphi}(t) = -mgr \cdot \sin(\varphi(t)) - M_{fr}\dot{\varphi}(t) \tag{B.1}$$

Linearizing it under the assumption that $\sin(\varphi) \approx \varphi$ for small φ leads to

$$I_\mathrm{a}\ddot{\varphi}(t) = -mgr \cdot \varphi(t) - M_{fr}\dot{\varphi}(t) \tag{B.2}$$

In order to keep the derivation simple, the following substitutions are made: $\delta = \frac{M_{fr}}{2I_a}$ and $\omega_0 = \sqrt{\frac{mgr}{I_a}}$. Plugging them into equation B.2 leads to

$$\ddot{\varphi}(t) + 2\delta \cdot \dot{\varphi}(t) + \omega_0^2 \cdot \varphi(t) = 0 \tag{B.3}$$

The characteristic polynomial for this constant coefficient differential equation is

$$\lambda^2 + 2\delta \cdot \lambda + \omega_0^2 = 0 \tag{B.4}$$

Its zeros are

$$\lambda_{1,2} = -\delta \pm \sqrt{\delta^2 - \omega_0^2} \tag{B.5}$$

From observing the Crazyflie's swinging motion we can already tell that the oscillation is underdamped which means that $\delta^2 < \omega_0^2$. With the additional substitution $\omega = \sqrt{\omega_0^2 - \delta^2}$ equation B.5 simplifies to

$$\lambda_{1,2} = -\delta \pm i\omega \tag{B.6}$$

Therefore the general solution to our ODE is

$$\varphi(t) = e^{-\delta t}\left(C_1 \cdot \cos \omega t + C_2 \cdot \sin \omega t\right) \tag{B.7}$$

By using the trigonometric identity $a \cdot \sin \omega t + b \cdot \cos \omega t = \sqrt{a^2 + b^2} \cdot \cos \omega t - \arctan \frac{a}{b}$ which can be proved using trigonometric sum identities equation B.7 can be restated as

$$\varphi(t) = \sqrt{C_1^2 + C_2^2} \cdot e^{-\delta t} \cdot \cos\left(\omega t - \arctan\left(\frac{C_2}{C_1}\right)\right) \tag{B.8}$$

Plugging the initial conditions $\varphi(0) = \varphi_0$ and $\dot{\varphi}(0) = \Omega_0$ into equation B.7 leads to

$$\begin{aligned} C_1 &= \varphi_0 \\ C_2 &= \frac{\Omega_0 + \delta\varphi_0}{\omega} \end{aligned} \tag{B.9}$$

and finally we get

$$\varphi(t) = \sqrt{\varphi_0^2 + \left(\frac{\Omega_0 + \delta\varphi_0}{\omega}\right)^2} \cdot e^{-\delta t} \cdot \cos\left(\omega t - \arctan\left(\frac{\Omega_0 + \delta\varphi_0}{\omega\varphi_0}\right)\right) \tag{B.10}$$

B.2 Derivation of the Solution to the Equation of Motion of a Dropping Weight

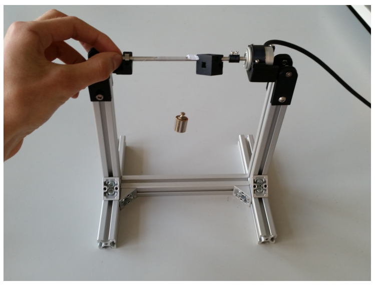


Figure B.1: Experiment Setup of the Dropping Weight Experiment

A weight with mass m_{weight} is attached to a fishing line which is wrapped around the axis (radius r) of the Crazyflie pendulum (see figure B.1). The equation of motion for this situation is

$$I_{\text{exp}} \cdot \ddot{\varphi}(t) + M_{\text{fr}} \cdot \dot{\varphi}(t) - m_{\text{weight}} g r = 0 \tag{B.11}$$

In this equation I_{exp} is the moment of inertia of the setup (axis, moving parts in encoder and mounting cube), $\varphi(t)$ is the angular position of the axis, g is the gravity of the earth and M_{fr} is the constant friction coefficient of the setup.

The first step is to determine the solution to the homogenous equation

$$I_{\mathrm{exp}} \cdot \ddot{\varphi}(t) + M_{\mathrm{fr}} \cdot \dot{\varphi}(t) = 0 \tag{B.12}$$

It has the characteristic polynomial

$$I_{\mathrm{exp}} \cdot \lambda^2 + M_{\mathrm{fr}} \cdot \lambda = 0 \tag{B.13}$$

with the solutions

$$\lambda_1 = 0, \lambda_2 = -\frac{M_{\mathrm{fr}}}{I_{\mathrm{exp}}} \tag{B.14}$$

Therefore the homogenous solution is

$$\varphi_{\mathrm{h}}(t) = C_1 + C_2 \cdot e^{-\frac{M_{\mathrm{fr}}}{I_{\mathrm{exp}}} t} \tag{B.15}$$

For the particular solution we choose as a started point the educated guess

$$\varphi_{\mathrm{p}}(t) = C_3 t \tag{B.16}$$

Plugging this into equation B.11 leads to

$$C_3 = \frac{m_{\mathrm{weight}} g r}{M_{\mathrm{fr}}} \tag{B.17}$$

The general solution is

$$\varphi(t) = \varphi_{\mathrm{p}}(t) + \varphi_{\mathrm{h}}(t) = C_1 + C_2 \cdot e^{-\frac{M_{\mathrm{fr}}}{I_{\mathrm{exp}}} t} + \frac{m_{\mathrm{weight}} g r}{M_{\mathrm{fr}}} t \tag{B.18}$$

With the initial conditions $\varphi(0) = 0$ and $\dot{\varphi}(0) = 0$ we get

$$\varphi(t) = \frac{I_{\mathrm{exp}} \cdot m_{\mathrm{weight}} g r}{M_{\mathrm{fr}}^2} \cdot \left(e^{-\frac{M_{\mathrm{fr}}}{I_{\mathrm{exp}}} t} - 1 \right) + \frac{m_{\mathrm{weight}} g r}{M_{\mathrm{fr}}} t \tag{B.19}$$

$$\dot{\varphi}(t) = \frac{m_{\mathrm{weight}} g r}{M_{\mathrm{fr}}} \cdot \left(1 - e^{-\frac{M_{\mathrm{fr}}}{I_{\mathrm{exp}}} t} \right) \tag{B.20}$$

B.3 Derivation of the formula for the moments of inertia of a cuboid

The situation is depicted in figure 2.16. The moments of inertia can be calculated using the general formula for calculating a moment of inertia [8]. The moment of inertia with respect to the x axis can be calculated as

$$\begin{aligned} I_{\mathrm{xx,G}} &= \int_m \left(y^2 + z^2\right) \mathrm{d}m \\ &= \rho \iiint\limits_V \left(y^2 + z^2\right) \mathrm{d}x \, \mathrm{d}y \, \mathrm{d}z \\ &= \rho \int_{-\frac{c}{2}}^{\frac{c}{2}} \int_{-\frac{b}{2}}^{\frac{b}{2}} \int_{-\frac{a}{2}}^{\frac{a}{2}} \left(y^2 + z^2\right) \mathrm{d}x \, \mathrm{d}y \, \mathrm{d}z \\ &= \dots \\ &= m \left(\frac{b^2}{12} + \frac{c^2}{12}\right) \end{aligned} \tag{B.21}$$

The moment of inertia with respect to the y and z axes can be calculated similarly.

Appendix C

Code

C.1 Sensor Data Processing for Inertia Matrix Experiments

C.1.1 Encoder Logger

```
import time
from datetime import date
from threading import Lock
from twisted.internet import reactor
from twisted.internet.serialport import SerialPort
from twisted.protocols import basic
from serial.tools import list_ports
import pandas as pd

# Running log of all Encoder communications
EncoderLog = []     # This is a list
EncoderLock = Lock()

axisLabel = ''

def QuitHandler():
    print('')
    EncoderLock.acquire()
    if len(EncoderLog)>0:
        print "Saving EncoderLog..."
        columnNames = ['t','phi']
        data = pd.DataFrame(data=EncoderLog, index=None, columns=
            columnNames)
        filename = 'Encoder_data_{0}_{1}_{2}.csv'.format(date.today(),
            time.strftime('%H-%M-%S'),axisLabel)
        data.to_csv(path_or_buf=filename, index=None, float_format="%.6
            f")
        print "Data saved to %s. Bye :-)" % filename
    else:
        print "Nothing to save. Bye :-)"


class EncoderLogger(basic.LineReceiver):
    delimiter = '\r'

    @staticmethod
    def getDevice():
        EncoderRanger = list(list_ports.grep("usbmodem1"))
        if len(EncoderRanger) > 0:
            return EncoderRanger[0][0]
        else:
            return None

    def connectionMade(self):
        print('Connected to Encoder at {0}'.format(self.getDevice()))
        print "Start logging..."
        self.delimiter = "".rstrip()
        self.sendLine("!")
        self.delimiter = '\r'

    def lineReceived(self,line):
        now = time.time()
        spl = line.split(":")      # '1:2' --> ['1','2']
        #print line
        #print spl
        #print '---'
        apd = (now,spl[0])      # This is a tuple
        EncoderLock.acquire()
        EncoderLog.append(apd)
        EncoderLock.release()
        self.delimiter = "".rstrip()
        self.sendLine("!")
        self.delimiter = '\r'


if __name__=="__main__":
    axisLabel = raw_input("What axis are we measuring? Input: ")
    while(EncoderLogger.getDevice() is None):
        print "Waiting for device..."
        time.sleep(1)
    SerialPort(EncoderLogger(),EncoderLogger.getDevice(), reactor,
        baudrate = 115200)
    reactor.addSystemEventTrigger('before','shutdown',QuitHandler)
    reactor.run()
```

C.1.2 Processing of Encoder Data from Dropping Weight Experiment

```
clc
clear all
close all

% Read the data in
filename='Data/2015-05-29_20-17-39_encoder_10g.csv';
NR_ignPks=0;
O_medfilt=7;
start_time=3500;
end_time=7000;

angle_raw=csvread(filename,1,1);
time_raw=csvread(filename,1,0,[1 0 length(angle_raw) 0]);
angle_raw=angle_raw(3:end);
time_raw=time_raw(3:end);

angle_raw=angle_raw-angle_raw(1);
```

```
maximum_angle = 4194047;
angle_raw = angle_raw / maximum_angle * 2*pi;



% Unwrap the data
[pks, loc]=findpeaks(angle_raw);

angle_unwr = angle_raw;

for i=1+NR_ignPks:length(loc)
    for j=loc(i)+1:length(angle_unwr)
        angle_unwr(j) = angle_unwr(j) + 2*pi;
    end
end

time_raw=time_raw-time_raw(1);
time_raw=time_raw/1000;     % Now the unit is milliseconds

figure
plot(time_raw/1000,angle_raw)
grid on
xlabel('Time [s]','Interpreter','LaTex','fontsize',20)
ylabel('Angle [rad]','Interpreter','LaTex','fontsize',20)
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';

% Make new time vector
Ts = 4; % ms

% Number of elements in new time vector
N = floor(max(time_raw)/Ts);
% New time vector
time = (0:Ts:N*Ts)';

angle = interp1(time_raw,angle_unwr,time);

figure
plot(time/1000,angle)
grid on
xlabel('Time [s]','Interpreter','LaTex','fontsize',20)
ylabel('Angle [rad]','Interpreter','LaTex','fontsize',20)
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
ylim([min(angle)-10 max(angle)+10])

% Determine slope of motion (--> speed)

angle_diff=diff(angle)/Ts*1000;     % Angular velocity in rad/s

% Apply median filter to reject outliers
angle_diff = medfilt1(angle_diff,O_medfilt);

% Average on a manually selected interval
for i=1:length(time)-1
    if time(i)<=start_time && time(i+1)>start_time
        start_idx=i;
    end
    if time(i)<=end_time && time(i+1)>end_time
        end_idx=i;
    end
end
if start_time>max(time)
    start_idx=length(time)-1;
end
if end_time>max(time)
    end_idx=length(time)-1;
end

speed = mean(angle_diff(start_idx:end_idx));
speed_vec=speed*ones(length(angle_diff),1);

time_plot=time/1000;

figure
hold on
plot(time_plot(1:end-1),angle_diff)
plot([time_plot(start_idx) time_plot(start_idx)],[min(angle_diff) max(
    angle_diff)],'k','linewidth',2)
plot([time_plot(end_idx) time_plot(end_idx)],[min(angle_diff) max(
    angle_diff)],'k','linewidth',2)
plot(time_plot(1:end-1),speed_vec,'linewidth',3)
grid on
box on
xlabel('Time [s]','Interpreter','LaTex','fontsize',20)
ylabel('Anglular Velocity [$\frac{rad}{s}$]','Interpreter','LaTex','
    fontsize',20)
ax=gca;
ax.FontSize=18;
%xlim([time(1) time(end-1)])
ax.TickLabelInterpreter='latex';
```

C.1.3 Linear and nonlinear fit to data from swing experiments

The main file that was used for the fit. Linear fit: lines 24 - 84, nonlinear fit: 85 - end of the file.

```
1  clc
2  clear all
3  close all
4
5  format long
6
7  global angle_FTP time_FTP phi_0 m g r time_non0 angle_non0 angle_exact
       angle_exact_dot angle_exact_dotdot pi_LS
8
9  filename='Data/Shrink3/2015-07-07_11-23-36_encoder_yz.csv';
10 [time_raw,angle_raw,time,angle,time_FTP,angle_FTP]=prepareCFdata(
       filename);
11
12 % Plot raw measurement data
13 figure
14 plot(time_FTP,angle_FTP)
15 xlabel('Time [s]','Interpreter','LaTex','fontsize',20)
16 ylabel('Angle [rad]','Interpreter','LaTex','fontsize',20)
17 ax=gca;
18 ax.FontSize=18;
19 ax.TickLabelInterpreter='latex';
20 grid on
21 %title('Crazyflie pendulum motion around x axis')
```

```matlab
xlim([0 time(length(time))])
ylimval = angle(1)+0.1;
ylim([-ylimval ylimval])

%%
% figure(1)
% plot(time_FTP,angle_FTP)

% Determine parameters of linear model to use them as initial conditions
% for calculating the exactly fitting function

disp('Linear fit')

% Known parameters
m = 0.028;
g = 9.81;
phi_0 = angle(1);
[r_x_y, r_z, r_xy, r_xz_yz]=Distances_from_CoM;
r = r_xz_yz;

% Initial guess
%M_fr_guess = 2e-5;
M_fr_guess = 0.1813e-5;
Theta_guess = 4.48e-5;   % Moment of inertia
Omega_0_guess = 0;

param_0 = [M_fr_guess Theta_guess Omega_0_guess];

[param_opt,fmin] = fminsearch(@objFunc_single,param_0);

%Evaluate result

fmin
M_fr = param_opt(1)
Theta = param_opt(2)
Omega_0 = param_opt(3)
delta=M_fr/2/Theta;
omega_0=sqrt(m*g*r/Theta);
omega=sqrt(omega_0^2-delta^2);

angle_model = zeros(length(angle),1);
for i=1:length(angle_model)
    angle_model(i)=sqrt(phi_0^2+((Omega_0+delta*phi_0)/omega)^2)*exp(-delta*time(i))*cos(omega*time(i)-atan((Omega_0+delta*phi_0)/omega/phi_0));
end

figure
plot(time,angle,time,angle_model)
h=legend('Data $\varphi_\mathrm{exp,i} (t)$','Model $\varphi (t)$');
h.Interpreter='latex';
h.FontSize=20;
xlim([0 time(length(time))])
ylimval = angle(1)+0.1;
ylim([-ylimval ylimval])
grid on
box on
%title('Fitting a linearized model to the data')
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
```

```matlab
xlabel('Time [s]','Interpreter','LaTex','fontsize',20)
ylabel('Angle [rad]','Interpreter','LaTex','fontsize',20)

fmin_man=0;
for i=1:length(angle_FTP)
    fmin_man=fmin_man+(angle_FTP(i)-angle_model(i))^2;
end
fmin_man


%%

% Cut part in the end where angle is constantly 0 in order to improve the fitting
tol=0.003;
for i=1:length(angle)-3
    if abs(angle(i))<tol && abs(angle(i+1))<tol && abs(angle(i+3))<tol
        end_idx = i;
        break
    end
end
angle_non0=angle(1:end_idx);
time_non0=time(1:end_idx);


% Calculate exactly fitting function
disp('Exact fit')

% Initial conditions
a3=sqrt(phi_0^2+((Omega_0+delta*phi_0)/omega)^2);
a7=-atan((Omega_0+delta*phi_0)/omega/phi_0);
% param_0 = [0 0 a3 delta omega 0 a7 0 0 0 0 0];     5 3
param_0 = [0 0 a3 delta omega 0 a7 0 0];     % Weird: adding one more zero than there are actually parameters improves fmin


[param_opt,fmin] = fminsearch(@objFunc_embracing_func,param_0);

angle_exact = zeros(length(angle_non0),1);
for i=1:length(angle_exact)
    % 5 3
    % angle_exact(i)=(param_opt(12)*time(i)^5+param_opt(9)*time(i)^4+param_opt(8)*time(i)^3+param_opt(1)*time(i)^2+param_opt(2)*time(i)+param_opt(3)+exp(-param_opt(4)*time(i))) * cos((param_opt(5)+param_opt(6)*time(i)+param_opt(10)*time(i)^2+param_opt(11)*time(i)^3)*time(i)+param_opt(7));
    angle_exact(i)=(param_opt(1)*time(i)^2+param_opt(2)*time(i)+param_opt(3)+exp(-param_opt(4)*time(i))) * cos((param_opt(5)+param_opt(6)*time(i)+param_opt(8)*time(i)^2)*time(i)+param_opt(7));
end

figure
plot(time_non0, angle_non0,'k', time_non0, angle_exact)
h=legend('Data $\varphi (t)$','Model $\varphi_\mathrm{flex} (t)$');
h.Interpreter='latex';
h.FontSize=20;
xlim([0 time(length(time))])
ylimval = angle(1)+0.1;
ylim([-ylimval ylimval])
grid on
xlabel('Time [s]','Interpreter','LaTex','fontsize',20)
```

```
ylabel('Angle [rad]','Interpreter','LaTex','fontsize',20)
%title('Fitting more flexible function to the data')
box on
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';



% Derivatives from Mathematica

a1=param_opt(1);
a2=param_opt(2);
a3=param_opt(3);
a4=param_opt(4);
a5=param_opt(5);
a6=param_opt(6);
a7=param_opt(7);
a8=param_opt(8);
angle_exact_dot = zeros(length(angle_exact),1);
angle_exact_dotdot = angle_exact_dot;
for i=1:length(angle_exact_dot)
    angle_exact_dot(i) = (a2 - a4 *exp(-a4 *time(i)) + 2* a1 *time(i)) *cos(a7 + time(i)* (a5 + a6 *time(i) + a8 *time(i)^2)) - (a3 + exp(-a4 *time(i)) + a2 *time(i) + a1 *time(i)^2)* (a5 + a6 *time(i) + a8 *time(i)^2 + time(i)* (a6 + 2 *a8 *time(i))) *sin(a7 + time(i)* (a5 + a6 *time(i) + a8 *time(i)^2));
    angle_exact_dotdot(i) = (2 *a1 + a4^2 *exp(-a4 *time(i))) *cos(a7 + time(i)* (a5 + a6 *time(i) + a8 *time(i)^2)) - (a3 + exp(-a4 *time(i)) + a2 *time(i) + a1 *time(i)^2)* (a5 + a6 *time(i) + a8 *time(i)^2 + time(i)* (a6 + 2 *a8 *time(i)))^2 *cos(a7 + time(i)* (a5 + a6 *time(i) + a8 *time(i)^2)) - (2 *a6 + 6* a8 *time(i)) *(a3 + exp(-a4 *time(i)) + a2 *time(i) + a1 *time(i)^2) *sin(a7 + time(i)* (a5 + a6 *time(i) + a8 *time(i)^2)) - 2 *(a2 - a4 *exp(-a4 *time(i)) + 2 *a1 *time(i)) *(a5 + a6 *time(i) + a8 *time(i)^2 + time(i)* (a6 + 2* a8 *time(i))) *sin(a7 + time(i)* (a5 + a6 *time(i) + a8 *time(i)^2));
end

figure
[hAx,~,~]=plotyy(time_non0,angle_exact,[time_non0,time_non0],[angle_exact_dot,angle_exact_dotdot]);
h=legend('$\varphi_\mathrm{flex} (t)$','$\dot{\varphi}_\mathrm{flex} (t) $','$\ddot{\varphi}_\mathrm{flex} (t) $');
h.Interpreter='latex';
h.FontSize=20;
grid on
box on
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
xlabel('Time [s]','Interpreter','LaTex','fontsize',20)
ylabel(hAx(1),'Angle [rad]','Interpreter','LaTex','fontsize',20)
ylabel(hAx(2),'Angular velocity/acceleration [$\frac{rad}{s}$]/[$\frac{rad}{s^2}$]','Interpreter','LaTex','fontsize',20)

% Check whether derivative makes sense
angle_exact_diffdot=diff(angle_exact)/0.004;
%figure
%plot(time_non0(1:end-1),angle_exact_diffdot,time_non0(1:end-1),angle_exact_dot(1:end-1))

```

```

% Fit parameters of nonlinear differential equation

disp('Nonlinear fit')

% Initial condition
param_0 = [Theta M_fr];

[param_opt,fmin] = fminsearch(@objFunc_nonLin,param_0);

fmin
M_fr_exact = param_opt(2)
Theta_exact = param_opt(1)

%% Plug parameters from nonlinear fit into linearized differential
    equation
disp('Linear model with exact parameters')
delta=M_fr_exact/2/Theta_exact;
omega_0=sqrt(m*g*r/Theta_exact);
omega=sqrt(omega_0^2-delta^2);

angle_model = zeros(length(angle),1);
for i=1:length(angle_model)
    angle_model(i)=sqrt(phi_0^2+((Omega_0+delta*phi_0)/omega)^2)*exp(-
        delta*time(i))*cos(omega*time(i)-atan((Omega_0+delta*phi_0)/
        omega/phi_0));
end

figure
plot(time,angle,time,angle_model)
legend('Data','Model')
xlim([0 time(length(time))])
ylimval = angle(1)+0.1;
ylim([-ylimval ylimval])
grid on
xlabel('Time [s]')
ylabel('Angle [rad]')
title('Linearized model with exact parameters')

fmin_man=0;
for i=1:length(angle_FTP)
    fmin_man=fmin_man+(angle_FTP(i)-angle_model(i))^2;
end
fmin_man

%% Simulate differential equation

pi_LS=[Theta_exact, M_fr_exact];
[T_verif,Y_verif] = ode45(@crazymotion,[0 time(end)],[angle(1) 0]);

figure
plot(time, angle,T_verif,Y_verif(:,1))
legend('Sensor data','ODE solve')
xlim([0 time(length(time))])
grid on
ylimval = angle(1)+0.1;
ylim([-ylimval ylimval])
xlabel('Time [s]')
ylabel('Angle [rad]')
title('Simulation using ODE solver with exact parameters')
```

File that was used to prepare the data.

```matlab
function [time_raw,angle_raw,time,angle,time_FTP,angle_FTP] =
    prepareCFdata(filename)

angle_raw=csvread(filename,1,1);
time_raw=csvread(filename,1,0,[1 0 length(angle_raw) 0]);
angle_raw=angle_raw(3:end);
time_raw=time_raw(3:end);

figure
plot(time_raw,angle_raw)
xlabel('Time [$\mu s$]','Interpreter','LaTex','fontsize',20)
ylabel('Encoder Value [-]','Interpreter','LaTex','fontsize',20)
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
grid on
min(angle_raw);
ylim([min(angle_raw)-100000,max(angle_raw)+100000])

% Die Einheit des Zeitvektors ist Mikrosekunden
time_raw = time_raw / 1000/1000;
% Jetzt ist die Einheit des Zeitvektors Sekunden

% Adjust angle axis to radians.
maximum_angle = 4194047;
angle_raw = angle_raw / maximum_angle * 2*pi;

a0 = angle_raw(20);
angle_raw = angle_raw - a0;

% Cut part where CF is deflected. Make the first peak be located at t
    =0.
[~,idx] = max(angle_raw);

NR_cutfirstelements=idx-1;
time_cut = zeros(length(time_raw)-NR_cutfirstelements,1);
angle_cut = zeros(length(angle_raw)-NR_cutfirstelements,1);
for i=(NR_cutfirstelements+1):length(time_raw)
    time_cut(i-NR_cutfirstelements)=time_raw(i);
    angle_cut(i-NR_cutfirstelements)=angle_raw(i);
end

time_cut = time_cut - time_cut(1);

% Regularize data

% Create new time vector up to a value that is nearly as high as the
% highest value in time_raw.
Ts = 0.004;

% Number of elements in new time vector
N = floor(time_cut(length(time_cut))/Ts);
% New time vector
time = (0:Ts:N*Ts)';
% New angle vector
angle = interp1(time_cut,angle_cut,time);

%plot(time,angle)

% Modify data so that only the first n_pks periods are used for the
    estimate
n_pks = 14;
```

```

% Find the peaks
[~, loc] = findpeaks(angle);

% Time and angle vectors containing only the data up to the 3rd peak
angle_FTP = zeros(loc(n_pks),1);
time_FTP = zeros(loc(n_pks),1);
for i=1:length(angle_FTP)
    angle_FTP(i) = angle(i);
    time_FTP(i) = time(i);
end

end
```

Objective function for linear fit.

```
function f = objFunc_single(a)

% a = [M_fr Theta Omega_0]

global angle_FTP time_FTP phi_0 m g r
f=0;
delta=a(1)/2/a(2);
omega_0=sqrt(m*g*r/a(2));
omega=sqrt(omega_0^2-delta^2);
a0=sqrt(phi_0^2+((a(3)+delta*phi_0)/omega)^2);
a1=-delta;
a2=omega;
a3=-atan((a(3)+delta*phi_0)/omega/phi_0);

for i=1:length(angle_FTP)
    f = f + (angle_FTP(i)-a0*exp(a1*time_FTP(i))*cos(a2*time_FTP(i)+a3)
        )^2;
end
end
```

Objective function for the fit to a more flexible function. The result of this fit can then be used to directly fit the data to the nonlinear equation of motion. This fit is needed to determine the first and second derivative of the data.

```
function f = objFunc_embracing_func(a)

% phi(t) = (a1*t^2+a2*t+a3+exp(-a4*t)) * cos((a5+a6*t+a8*t^2)*t+a7)

global time_non0 angle_non0
f=0;
for i=1:length(angle_non0)
%     f = f + (angle_non0(i)- (a(12)*time_non0(i)^5+a(9)*time_non0(i)
    ^4+a(8)*time_non0(i)^3+a(1)*time_non0(i)^2+a(2)*time_non0(i)+a(3)+
    exp(-a(4)*time_non0(i))) * cos((a(5)+a(6)*time_non0(i)+a(10)*
    time_non0(i)^2+a(11)*time_non0(i)^3)*time_non0(i)+a(7)))^2;
    f = f + (angle_non0(i)- (a(1)*time_non0(i)^2+a(2)*time_non0(i)+a(3)
        +exp(-a(4)*time_non0(i))) * cos((a(5)+a(6)*time_non0(i)+a(8)*
        time_non0(i)^2)*time_non0(i)+a(7)))^2;
end
end
```

Objective function to fit the nonlinear equation of motion to the data.

```
function f = objFunc_nonLin(a)
```

```

% Theta*phi_dotdot = -M_fr*phi_dot - m*g*r*sin(phi)

global angle_exact angle_exact_dot angle_exact_dotdot m g r
f=0;

for i=1:length(angle_exact)

    f = f + (a(1)*angle_exact_dotdot(i)+a(2)*angle_exact_dot(i)+m*g*r*
        sin(angle_exact(i)))^2;
end

end
```

ODE that describes the nonlinear equation of motion. This file is used to verify the result by simulating the equation of motion with Matlab's `ode45` solver.

```
function dy = crazymotion( t,y )
%crazymotion ODE that describes CF's motion.

global pi_LS m g r
dy = zeros(2,1);
dy(1) = y(2);
dy(2) = -1/pi_LS(1)*(pi_LS(2)*y(2)+m*g*r*sin(y(1)));

end
```

C.1.4 Calculation of the Crazyflie's inertia matrix

```
% Script to calculate the Crazyflie's inertia matrix
clc
clear all
close all

[r_x_y, r_z, r_xy, r_xz_yz] = Distances_from_CoM;
m=0.028;

% % Values for presentation
% I_a_x = 3.123854506572300E-05;
% I_a_y = 3.131407384567920E-05;
% I_a_z = 7.081772686347800E-05;
% I_a_xy = 5.019475570019130E-05;
% I_a_xz = 4.686939838435990E-05;
% I_a_yz = 4.849766349082040E-05;

% % Values for report, n_pks = 7
% I_a_x = 3.144988206680423E-05;
% I_a_y = 3.151127080468852E-05;
% I_a_z = 7.058874150001670E-05;
% I_a_xy = 5.003777167580184E-05;
% I_a_xz = 4.640539662252733E-05;
% I_a_yz = 4.780234696836194E-05;

% Values for report, n_pks varying, see report
I_a_x = 3.119770273213515e-05;
I_a_y = 3.128159434626518e-05;
I_a_z = 7.004148241971259e-05;
I_a_xy = 4.940332984711082e-05;
```

```
I_a_xz = 4.600706365028182e-05;
I_a_yz = 4.713092929680177e-05;

% Transfer I's to CoM
I_a_x_G = I_a_x - m*r_x_y^2;
I_a_y_G = I_a_y - m*r_x_y^2;
I_a_z_G = I_a_z - m*r_z^2;
I_a_xy_G = I_a_xy - m*r_xy^2;
I_a_xz_G = I_a_xz - m*r_xz_yz^2;
I_a_yz_G = I_a_yz - m*r_xz_yz^2;

y=[I_a_x_G I_a_y_G I_a_z_G I_a_xy_G I_a_xz_G I_a_yz_G]';

u_x = [1 0 0];
u_y = [0 1 0];
u_z = [0 0 1];
u_xy = [1/sqrt(2) 1/sqrt(2) 0];
u_xz = [1/sqrt(2) 0 1/sqrt(2)];
u_yz = [0 1/sqrt(2) 1/sqrt(2)];

A=[u_x(1)^2, u_x(2)^2, u_x(3)^2, -2*u_x(1)*u_x(2), -2*u_x(2)*u_x(3),
    -2*u_x(1)*u_x(3);
    u_y(1)^2, u_y(2)^2, u_y(3)^2, -2*u_y(1)*u_y(2), -2*u_y(2)*u_y(3),
        -2*u_y(1)*u_y(3);
    u_z(1)^2, u_z(2)^2, u_z(3)^2, -2*u_z(1)*u_z(2), -2*u_z(2)*u_z(3),
        -2*u_z(1)*u_z(3);
    u_xy(1)^2, u_xy(2)^2, u_xy(3)^2, -2*u_xy(1)*u_xy(2), -2*u_xy(2)*
        u_xy(3), -2*u_xy(1)*u_xy(3);
    u_xz(1)^2, u_xz(2)^2, u_xz(3)^2, -2*u_xz(1)*u_xz(2), -2*u_xz(2)*
        u_xz(3), -2*u_xz(1)*u_xz(3);
    u_yz(1)^2, u_yz(2)^2, u_yz(3)^2, -2*u_yz(1)*u_yz(2), -2*u_yz(2)*
        u_yz(3), -2*u_yz(1)*u_yz(3)];

b=A\y;
I_x = b(1)
I_y=b(2)
I_z=b(3)
I_xy=b(4)
I_yz=b(5)
I_xz=b(6)
```

C.1.5 Analytic calculation of moments of inertia for the test body

```


clc
close all
clear all


% Everything in SI units.

m=0.0176;


% Shaft (1)
l1=0.026;
b1=0.0051;
h1=0.0048;
```

```

V1=l1*h1*b1;

% Body (2)
l2=0.05975;
b2=0.0498;
h2=0.0048;

V2=l2*b2*h2;


V=V1+V2;
rho=m/V;

m1=rho*V1;
m2=rho*V2;

% Distance of common CoM to bottom edge:
yG_dash = (l2/2*m2+(l2+l1/2)*m1)/(m1+m2);


%%

% Distances of axes to CoM
[r_x,r_z]=Distances_from_CoM(yG_dash)
r_y=0.04;       % Not the correct value!!!

% Moments of inertia x axis

% 1, CoM_1
Theta_x1G = m1*(l1^2+h1^2)/12;

% 1, CoM_common
Theta_x1 = Theta_x1G + m1*(l1/2+l2-yG_dash)^2;

% 2, CoM_2
Theta_x2G = m2*(l2^2+h2^2)/12;

% 2, CoM_common
Theta_x2 = Theta_x2G+m2*(yG_dash-l2/2)^2;

% Total, CoM_common
Theta_xG = Theta_x1 + Theta_x2

% Total, Position of axis
Theta_x = Theta_xG + m*r_x^2




% Moments of inertia z axis

% 1, CoM_1
Theta_z1G = m1*(b1^2+l1^2)/12;

% 1, CoM_common
Theta_z1 = Theta_z1G + m1*(l1/2+l2-yG_dash)^2;

% 2, CoM_2
Theta_z2G = m2*(b2^2+l2^2)/12;

% 2, CoM_common
```

```
Theta_z2 = Theta_z2G+m2*(yG_dash-l2/2)^2;

% Total, CoM_common
Theta_zG = Theta_z1 + Theta_z2

% Total, Position of axis
Theta_z = Theta_zG + m*r_z^2


% Moments of inertia xy axis

% 1, CoM_1
Theta_y1G = m1*(b1^2+h1^2)/12;

% 1, CoM_common
Theta_y1 = Theta_y1G + m1*(l1/2+l2-yG_dash)^2;

% 2, CoM_2
Theta_y2G = m2*(b2^2+h2^2)/12;

% 2, CoM_common
Theta_y2 = Theta_y2G + m2*(yG_dash-l2/2)^2;

% Total, CoM_common
Theta_yG = Theta_y1 + Theta_y2;

% Total, Position of axis
Theta_y = Theta_yG + m*r_y^2;
```

The distances of the rotational axes from mass center are calculated using the geometry of the test body and the mounting cube.

```
function [d_x, d_z] = Distances_from_CoM(y_com)


% All values in m

% Origin of the coordinate system is on the lowest edge of the testbody
    in
% the middle:
%    _
%  _| |_
% |     |
% |     |
% |__x__|


% Position of the center of mass:
com = [0,y_com,0]';

% z axis

% Point that z-axis goes through
a_z = [-5.6,59.75+1+15,0]';
a_z = a_z * 1e-3;

% Direction of z-axis
b_z = [0,0,1]';

%disp('Distance of CoM to z-axis: ')
d_z = norm(cross((com - a_z),b_z))/norm(b_z);

```

```matlab


% x axis

% Point that x_axis goes through
a_x = [0,59.75+1+11,4.8/2+6.7-4.2]';
a_x = a_x * 1e-3;

% Direction of x-axis
b_x = [1,0,0]';

%disp('Distance of CoM to x axis: ')
d_x = norm(cross((com - a_x),b_x))/norm(b_x);

% y axis

% Point that y axis goes through
%a_y=[0,


end
```

C.1.6 Calculation of the test body's moments of inertia from experiment results

```matlab

m=0.0176;

%% x axis

% Value for n_pks = 9
I_xx_1 = 3.507552821629088e-5;
% Value for n_pks = 7
I_xx_2=3.517772052094052e-05;

r_x = 0.040344532815297;

I_xx_1_G = I_xx_1 - m*r_x^2

Error_x_1 = abs(I_xx_1_G-6.410179e-6)/6.410179e-6

I_xx_2_G = I_xx_2 - m*r_x^2

Error_x_2 = abs(I_xx_2_G-6.410179e-6)/6.410179e-6


%% z axis

% Value for n_pks = 9
I_zz_1 = 4.487304318368619e-5;
% Value for n_pks = 12
I_zz_2 = 4.469524719338436e-5;
% Value for n_pks = 7
I_zz_3 = 4.512302287526182e-05;


r_z = 0.044400430747127;

I_zz_1_G = I_zz_1 - m*r_z^2

```

```
Error_z_1 = abs(I_zz_1_G-9.860228e-6)/9.860228e-6

I_zz_2_G = I_zz_2 - m*r_z^2

Error_z_2 = abs(I_zz_2_G-9.860228e-6)/9.860228e-6

I_zz_3_G = I_zz_3 - m*r_z^2

Error_z_3 = abs(I_zz_3_G-9.860228e-6)/9.860228e-6
```

C.1.7 Other Calculations

Calculating distances from rotation axes to mass center of the Crazyflie

Calculates the distances between all possible rotation axes the Crazyflie can be mounted to and the Crazyflie's mass center. The calculations are based on the geometry of the Crazyflie and the mounting cube.

```
function [d_x_y, d_z, d_xy, d_xz_yz] = Distances_from_CoM

% Origin has the same x and y position as the center of mass but is on
    the
% surface the Crazyflie stands on.

% A line is represented using a point the line goes through (a) and a
% vector that specifies the line's direction (b).


% Position of the center of mass:
z_com = 17.425e-3;     % Determined through experiments
com = [0,0,z_com]';

%% z axis

% a in coordinate frame with y_dash axis along the arm of the CF
a_z_dash = 10^(-3)*[-5.2/2-3,92/2-6.9/2-0.8-4,0]';

% Transformation matrix from x/y_dash to x/y
tm = [cos(pi/4),sin(pi/4),0;-sin(pi/4),cos(pi/4),0;0,0,1];

% Point that z-axis goes through
a_z = tm * a_z_dash;

% Direction of z-axis
b_z = [0,0,1]';

%disp('Distance of CoM to z-axis: ')
d_z = norm(cross((com - a_z),b_z))/norm(b_z);

%% xy axis

% a in coordinate frame with y_dash axis along the arm of the CF
a_xy_dash = 10^(-3)*[0,92/2-6.9/2-0.8-8,17.3-0.85+6.7-4.2]';

% Trafo matrix is the same as for z axis

% Point that xy_axis goes through
a_xy = tm * a_xy_dash;

% Direction of xy-axis
b_xy = [-sqrt(2)/2,sqrt(2)/2,0]';
```

```

%disp('Distance of CoM to xy-axis: ')
d_xy = norm(cross((com - a_xy),b_xy))/norm(b_xy);

%% x or y axis

% a in coordinate frame with y_dash axis along the arm of the CF
a_x_y_dash = 10^(-3)*[0,92/2-6.9/2-0.8-19/2,17.3-0.85+6.7-4.2]';

% Trafo matrix is the same as for z axis

% Point that xy_axis goes through
a_x_y = tm * a_x_y_dash;

% Direction of xy-axis
b_x_y = [1,0,0]';

%disp('Distance of CoM to x or y axis: ')
d_x_y = norm(cross((com - a_x_y),b_x_y))/norm(b_x_y);

%% xz or yz axis

% a in coordinate frame with y_dash axis along the arm of the CF
a_xz_yz_dash = 10^(-3)*[-0.8,92/2-6.9/2-0.8-7,17.3-0.85+6.7]';

% Trafo matrix is the same as for z axis

% Point that xz or yz axis goes through
a_xz_yz = tm * a_xz_yz_dash;

% Direction of xz or yz axis
b_xz_yz = [-1,0,-1]';

%disp('Distance of CoM to xz or yz axis: ')
d_xz_yz = norm(cross((com - a_xz_yz),b_xz_yz))/norm(b_xz_yz);

end
```

C.2 Code in connection with thrust parameters

C.2.1 Static Thrust Tests

Python script that allows to transmit thrust setpoints for an arbitrary number of motors to the Crazyflie.

```
# -*- coding: utf-8 -*-
#
# Written by Julian Foerster based on the example ramp.py that was
    developed by Bitcraze.
#

"""
    Program that allows entering motor commands for each motor
        separately or for several motors at the same time and sending
        them to the Crazyflie. This script communicates with the thrust
         module on the Crazyflie.
"""

import time, sys
```

```python
from threading import Thread

sys.path.append("../lib")
import cflib
from cflib.crazyflie import Crazyflie
from cflib.crtp.crtpstack import CRTPPort
from cflib.crtp.crtpstack import CRTPPacket
from twisted.internet import reactor


import logging
logging.basicConfig(level=logging.ERROR)

import struct

class SendThrustSetpoints:
    """Script that allows to run motor commands"""
    def __init__(self, link_uri):
        """ Initialize and run the script with the specified link_uri """

        self._cf = Crazyflie()

        self._cf.connected.add_callback(self._connected)
        self._cf.disconnected.add_callback(self._disconnected)
        self._cf.connection_failed.add_callback(self._connection_failed)
        self._cf.connection_lost.add_callback(self._connection_lost)

        self._cf.open_link(link_uri)

        print "Connecting to %s" % link_uri

    def _connected(self, link_uri):
        """ This callback is called form the Crazyflie API when a Crazyflie
            has been connected and the TOCs have been downloaded."""

        # Start a separate thread to do the motor test.
        # Do not hijack the calling thread!
        Thread(target=self._motor_control).start()

    def _connection_failed(self, link_uri, msg):
        """Callback when connection initial connection fails (i.e no Crazyflie
            at the speficied address)"""
        print "Connection to %s failed: %s" % (link_uri, msg)

    def _connection_lost(self, link_uri, msg):
        """Callback when disconnected after a connection has been made (i.e
            Crazyflie moves out of range)"""
        print "Connection to %s lost: %s" % (link_uri, msg)

    def _disconnected(self, link_uri):
        """Callback when the Crazyflie is disconnected (called in all cases)"""
        print "Disconnected from %s" % link_uri

    def _motor_control(self):
        self._cf.commander.send_setpoint(0,0,0,0)
        print "WARNING: This script allows to let the CF motors run with full thrust. Make sure that the Crazyflie is secured properly to prevent it from flying away and suffering severe damage."
        print "The motor IDs are indicated on the Crazyflie's circuit board: If the two blue LEDs are pointing towards you, motor 1 is the one on the top right and the other motors are numbered clockwisely."

        while (1):
            choice = raw_input("Enter the number of motors you would like to adress. To turn off all motors hit P. To adress all motors type A. To leave this program type Q. Input: ")

            pk = CRTPPacket()
            pk.port = CRTPPort.THRUST

            if (choice == "P"):
                pk.data = struct.pack('<Hi', 0, 4)
                self._cf.send_packet(pk)
                time.sleep(0.1)
                print "All motors stopped."
            elif (choice == "A"):
                thrustVal = int(input("Enter the input command all motors should be provided with (Max. 65535): "))
                if (thrustVal>65535):
                    print "The entered value is too high, try again."
                    continue
                pk.data = struct.pack('<Hi', thrustVal, 4)
                self._cf.send_packet(pk)
                time.sleep(0.1)
                print "Command sent."
            elif (choice == "Q"):
                pk.data = struct.pack('<Hi', 0, 4)
                self._cf.send_packet(pk)
                time.sleep(0.1)
                print "All motors stopped."
                print "Disconnecting from Crazyflie and terminating execution..."
                break
            else:
                choice = int(choice)
                if (choice >= 1 and choice <= 3):
                    thrust = [0,0,0]
                    motor_NR = [0,0,0]
                    x=1
                    while x<=choice:
                        motor_NR[x-1]=int(raw_input("%d) What motor number? " % x))
                        if motor_NR[x-1]<1 or motor_NR[x-1]>4:
                            print "Input not valid, try again."
                            continue
                        thrust[x-1]=int(raw_input("%d) What input command? " % x))
                        if (thrust[x-1]>65535):
                            print "The entered value is too high, try again."
                            continue
                        x += 1
                    for x in range(0,choice):
                        pk.data = struct.pack('<Hi', thrust[x], motor_NR[x]-1)
                        self._cf.send_packet(pk)
                        time.sleep(0.1)
                    print "Commands sent."
                else:
                    print "Input not valid."

        self._cf.close_link()

if __name__ == '__main__':
    # Initialize the low-level drivers (don't list the debug drivers)
    cflib.crtp.init_drivers(enable_debug_driver=False)
    # Scan for Crazyflies and use the first one found
    print "Scanning interfaces for Crazyflies..."
    available = cflib.crtp.scan_interfaces()
    print "Crazyflies found:"
    for i in available:
        print i[0]

    if len(available) > 0:
        #le = SendThrustSetpoints(available[0][0])
        #le = SendThrustSetpoints("radio://0/80/250K")
        reactor.callInThread(SendThrustSetpoints,available[0][0])
        reactor.run()
    else:
        print "No Crazyflies found, cannot run example"
```

Crazyflie module that receives the thrust setpoints and passes them to the motors. ATTENTION: when using this the four consecutive lines `motorsSetRatio(MOTOR_Mi, motorPowerMi)` with $i = 1, \dots, 4$ have to be commented out. Otherwise the commander watchdog of the firmware will turn off the motors immediately after they were started because there is no continuous stream of input commands being transmitted.

```
//Created by Julian Foerster

// Module that is used to determine the CF's thrust map. It allows to pass inputs to every single motor.

// This module was written with the Crazyflie firmware modules from Bitcraze as reference.

#ifndef THRUST_H_
#define THRUST_H_

//Member functions
void thrustInit(void);


#endif /* THRUST_H_ */
```

```
// Created by Julian Foerster

// Module that is used to determine the CF's thrust map. It allows to pass inputs to every single motor.

// This module was written with the Crazyflie firmware modules from Bitcraze as reference.
```

```c


#include "FreeRTOS.h"
#include "task.h"

#include "crtp.h"   //Used to send and receive information
#include "thrust.h"

#include "motors.h" // Contains function motorsSetRatio

struct thrustCrtpValues {       // Inspired by struct defined in
    commander.c
    uint16_t thrust;
    int motorID;
} __attribute__((packed));


// Member variables
static bool isInit;
struct thrustCrtpValues targetValue;


// Function prototypes
static void thrustCrtpCB(CRTPPacket* pk);
static uint16_t limitThrust(int32_t value);     // Taken from
    stabilizer.c

//Member functions
void thrustInit(void)
{
    // Only has to be initialized once...
    if(isInit)
        return;

    // We need crtp, so make sure it is initialized (if it is already
        initialized, nothing will happen
    crtpInit();
    // Register the function that will be called when a comes in on the
         HELLO port
    crtpRegisterPortCB(CRTP_PORT_THRUST, thrustCrtpCB);

    isInit = true;
}

static void thrustCrtpCB(CRTPPacket* pk)
{
    targetValue = *((struct thrustCrtpValues*)pk->data);

    uint32_t motorRatio = limitThrust(targetValue.thrust);

    if (targetValue.motorID == 4) {
        motorsSetRatio(MOTOR_M1, motorRatio);
        motorsSetRatio(MOTOR_M2, motorRatio);
        motorsSetRatio(MOTOR_M3, motorRatio);
        motorsSetRatio(MOTOR_M4, motorRatio);
    } else {
        motorsSetRatio(targetValue.motorID, motorRatio);
    }

}

// Taken from stabilizer.c
```

```
static uint16_t limitThrust(int32_t value)
{
    if(value > UINT16_MAX)
    {
        value = UINT16_MAX;
    }
    else if(value < 0)
    {
        value = 0;
    }

    return (uint16_t)value;
}
```

C.2.2 Load Cell Logger

Simple logger that connects to the load cell, asks it to stream force and torque data continuously and saves this data in a .csv file.

```
import time
from datetime import date
from threading import Lock
from struct import pack, unpack
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol
import pandas as pd
import socket



# Running log of all FTS (=Force Torque Sensor) communications
FTSLog = []      # This is a list
FTSLock = Lock()

def QuitHandler():
    print('')
    FTSLock.acquire()
    if len(FTSLog)>0:
        print "Saving FTSLog..."
        columnsNames = ['t','Fx','Fy','Fz','Tx','Ty','Tz']
        data = pd.DataFrame(data=FTSLog,columns=columnsNames, index=
            None)
        filename = 'FTS_data_{0}_{1}.csv'.format(date.today(),time.
            strftime('%H-%M-%S'))
        data.to_csv(path_or_buf=filename, index=None)
        print "Data saved to %s. Bye :-)" % filename
    else:
        print "Nothing to save. Bye :-)"


class FTSLogger(DatagramProtocol):
    # Based on http://twistedmatrix.com/documents/12.3.0/core/howto/udp
        .html
    HOST = '192.168.1.200'
    PORT = 49152
    ADR = (HOST,PORT)

    def startProtocol(self):
        print('Connecting to {0}:{1}...'.format(self.HOST,self.PORT))
        self.transport.connect(self.HOST,self.PORT)
        print "Requesting data..."
        self.transport.write(pack('>HHI',0x1234,0x0002,0))

    def datagramReceived(self,datagram,address):
        now = time.time()
        #print "Received something..."
        seqNum1,seqNum2,status,Fx,Fy,Fz,Tx,Ty,Tz = unpack('>IIIiiiiii',
            datagram)
        apd = (now,Fx,Fy,Fz,Tx,Ty,Tz)
        FTSLock.acquire()
        FTSLog.append(apd)
        FTSLock.release()

    def connectionRefused(self):
        print "No one listening..."

if __name__=="__main__":

    reactor.listenUDP(FTSLogger.PORT, FTSLogger())
    reactor.addSystemEventTrigger('before','shutdown',QuitHandler)
    reactor.run()
```

C.2.3 Data processing for the mappings

Input command → angular velocity

```
clc
close all

% Input command
cmd_ = 0:2000:64000;
cmd = zeros(length(cmd_)-1,1);
cmd(1)=cmd_(1);
cmd(2:end)=cmd_(3:end);

% First data points
M1_ = xlsread('RPM_measurements.xlsx','E4:E36');
M1 = zeros(length(M1_)-1,1);
M1(1)=M1_(1);
M1(2:end)=M1_(3:end);
M1=M1/2;         % Reflective stickers on both blades --> double the RPM
     is measured...
M1=M1/60*2*pi;       % convert to angular velocity

% Second data points
M2_ = xlsread('RPM_measurements.xlsx','F4:F36');
M2 = zeros(length(M2_)-1,1);
M2(1)=M2_(1);
M2(2:end)=M2_(3:end);
M2=M2/2;
M2=M2/60*2*pi;       % convert to angular velocity


[p1,e1]=polyfit(cmd,M1,1);
[p2,e2]=polyfit(cmd,M2,1);

figure
scatter(cmd,M1)
hold on
plot(cmd,p1(1)*cmd+p1(2))
```

```
grid on
box on
hleg=legend('Experiment data','Linear fit','Location','best');
set(hleg,'FontSize',20)
hleg.Interpreter='latex';
%title('1st measurement row')
xlabel('Input command [0, 65535]','Interpreter','latex','FontSize',20)
ylabel('Angular velocity [rad/s]','Interpreter','latex','FontSize',20)
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';

figure
scatter(cmd,M2)
hold on
plot(cmd,p2(1)*cmd+p2(2))
grid on
box on
hleg=legend('Experiment data','Linear fit','Location','best');
set(hleg,'FontSize',20)
hleg.Interpreter='latex';
%title('2nd measurement row')
xlabel('Input command [0, 65535]','Interpreter','latex','FontSize',20)
ylabel('Angular velocity [rad/s]','Interpreter','latex','FontSize',20)
%title('Input Command -> Angular Velocity')
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';

disp('First row')
disp(['omega = ' num2str(p1(1)) ' * command + ' num2str(p1(2))])
disp(['Error: ' num2str(e1.normr)])

disp('Second row')
disp(['omega = ' num2str(p2(1)) ' * command + ' num2str(p2(2))])
disp(['Error: ' num2str(e2.normr)])
```

Input command → thrust and thrust → torque

```

clc
close all

global data M2

%% Thrust -> torque

date = '150608_';

% First column: Fz, second column: Tz
data=zeros(32,2);

times = zeros(32,2);
times(2,:)=[5 7];
times(3,:)=[4 8];
times(4,:)=[4 8];
times(5,:)=[4 8];
times(6,:)=[4 7];
times(7,:)=[5.5 10];
times(8,:)=[4.5 8];
times(9,:)=[5 9];
times(10,:)=[6 10];
times(11,:)=[4 7.6];
times(12,:)=[4.5 8.5];
times(13,:)=[5 8];
times(14,:)=[7 10.5];
times(15,:)=[7 11.5];
times(16,:)=[5 9];
times(17,:)=[4 7];
times(18,:)=[3 6.5];
times(19,:)=[2.5 6];
times(20,:)=[2.5 5.5];
times(21,:)=[3.3 7];
times(22,:)=[3 7.5];
times(23,:)=[2.5 7];
times(24,:)=[3.5 8];
times(25,:)=[2 6.5];
times(26,:)=[2 6];
times(27,:)=[2 6];
times(28,:)=[3 7];
times(29,:)=[4 8];
times(30,:)=[3.5 7.2];
times(31,:)=[2 7];
times(32,:)=[3 10];

plot_true=0;
for i=2:length(data)
    if i==50
        plot_true=1;
    end
    [data(i,1), data(i,2)] = prepare_data([date num2str(i+1) '.csv'],
        plot_true,times(i,1),times(i,2));
    plot_true=0;
end

[m,~]=polyfit(data(:,1),data(:,2),1);

figure
scatter(data(:,1),data(:,2))
hold on
plot(data(:,1),m(1)*data(:,1)+m(2))
xlabel('Thrust [N]','Interpreter','latex','FontSize',20)
ylabel('Torque [Nm]','Interpreter','latex','FontSize',20)
hleg=legend('Experiment data','Linear fit','Location','southeast');
set(hleg,'FontSize',20)
hleg.Interpreter='latex';
grid on
box on
%title('Thrust -> Torque')
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';

format long
disp(['Torque = ' num2str(m(1)) ' * Thrust + ' num2str(m(2))])

%% input command -> thrust

cmd_ = 0:2000:64000;
cmd = zeros(length(cmd_)-1,1);
cmd(1)=cmd_(1);
cmd(2:end)=cmd_(3:end);

```

```matlab
[p,~]=polyfit(cmd,data(:,1),2);

% [p_matthew,~]=polyfit(cmd,data(:,1),1);

figure
scatter(cmd,data(:,1))
hold on
plot(cmd,p(1).*cmd.^2+p(2).*cmd+p(3))
grid on
box on
xlabel('Input command [0, 65535]','Interpreter','latex','FontSize',20)
ylabel('Thrust [N]','Interpreter','latex','FontSize',20)
hleg=legend('Experiment data','Quadratic fit','Location','best');
set(hleg,'FontSize',20)
hleg.Interpreter='latex';
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
%title('Input Command -> Thrust')

disp(['Thrust = ' num2str(p(1)) ' * cmd^2 + ' num2str(p(2)) ' * cmd + '
     num2str(p(3))])

% % Linear
% figure
% scatter(cmd,data(:,1))
% hold on
% plot(cmd,p_matthew(1).*cmd+p_matthew(2))
% grid on
% xlabel('Input command [-]')
% ylabel('Thrust [N]')
% legend('Experiment data','Quadratic fit','Location','southeast')
%
% disp(['Thrust = ' num2str(p_matthew(1)) ' * cmd + ' num2str(p_matthew
    (2))])

%% Verification input -> thrust

%Compare to verify data measured with the scales
figure
scatter(cmd,data(:,1))
hold on
plot(cmd,p(1).*cmd.^2+p(2).*cmd+p(3))
grid on
xlabel('Input command [0, 65535]','Interpreter','latex','FontSize',20)
ylabel('Thrust [N]','Interpreter','latex','FontSize',20)
%title('Input Command -> Thrust')
cmd_SD = 0:4000:64000;
SD = xlsread('Verification','F5:F21');
box on
scatter(cmd_SD,SD,'MarkerEdgeColor',[1 204/255 51/255],'linewidth',2)
hleg=legend('Experiment data','Quadratic fit','Verification data','
    Location','best');
set(hleg,'FontSize',20)
hleg.Interpreter='latex';
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';


%% Rotor speed -> thrust

```

```
% Second data points rotor speed (chose them because error is smaller)
M2_ = xlsread('../Input_Speed/RPM_measurements.xlsx','F4:F36');
M2 = zeros(length(M2_)-1,1);
M2(1)=M2_(1);
M2(2:end)=M2_(3:end);
M2=M2/2;     % Divide by two because each propeller blade has a
    reflective marker on it so the real RPM is half of the measured one
M2=M2/60*2*pi;      % Convert to rad/s

[n,~]=polyfit(M2,data(:,1),2);

init=n(1);
[param_opt,fmin]=fminsearch(@objFunc_Mat,init);

figure
scatter(M2,data(:,1))
hold on
plot(M2,n(1).*M2.^2+n(2).*M2+n(3))
grid on
xlabel('Angular velocity [rad/s]')
ylabel('Thrust [N]')
legend('Experiment data','Quadratic fit','Location','southeast')
title('Angluar velocity -> Thrust')

disp(['Thrust = ' num2str(n(1)) ' * omega^2 + ' num2str(n(2)) ' * omega
     + ' num2str(n(3))])

% Only x^2
figure
scatter(M2,data(:,1))
hold on
plot(M2,param_opt.*M2.^2)
grid on
xlabel('Angular velocity [rad/s]')
ylabel('Thrust [N]')
legend('Experiment data','Quadratic fit','Location','southeast')

disp(['Thrust = ' num2str(param_opt) ' * omega^2'])
```

```
function [Fz_mean, Tz_mean] = prepare_data(filename,plot_true,
    start_time,end_time)

Tz_raw = csvread(filename,1,6);
Fz_raw = csvread(filename,1,3,[1 3 length(Tz_raw) 3]);
t_raw = csvread(filename,1,0,[1 0 length(Tz_raw) 0]);
t_raw=t_raw-t_raw(1);

counts_per_force_torque=1000000;
Tz_raw=(Tz_raw./counts_per_force_torque)/2;   % Divide by 2 bcs
    experiment was carried out with 2 props/motors at the same time.
Fz_raw=(Fz_raw./counts_per_force_torque)/2;

bias=mean(Fz_raw(10:60));
Fz_raw =Fz_raw-bias;
bias=mean(Tz_raw(10:60));
Tz_raw =Tz_raw-bias;

%max_beginning=max(Fz_raw(10:60));

for i=1:length(t_raw)-1
    if t_raw(i)<=start_time && t_raw(i+1)>start_time
```

```
        start_idx=i;
    end
    if t_raw(i)<=end_time && t_raw(i+1)>end_time
        end_idx=i;
    end
end

Fz_mean=mean(Fz_raw(start_idx:end_idx));
Tz_mean=mean(Tz_raw(start_idx:end_idx));

if plot_true==1
    figure
    subplot(121)
    plot(t_raw,Fz_raw)
    hold on
    % Plot the mean value
    plot(t_raw,Fz_mean*ones(length(t_raw),1))
    % Plot vertical lines that mark where the mean calculation starts and ends
    plot([t_raw(start_idx) t_raw(start_idx)],[min(Fz_raw) max(Fz_raw)],'color','k')
    plot([t_raw(end_idx) t_raw(end_idx)],[min(Fz_raw) max(Fz_raw)],'color','k')
    title('Fz')

    subplot(122)
    plot(t_raw,Tz_raw)
    hold on
    % Plot the mean value
    plot(t_raw,Tz_mean*ones(length(t_raw),1))
    % Plot vertical lines that mark where the mean calculation starts and ends
    plot([t_raw(start_idx) t_raw(start_idx)],[min(Tz_raw) max(Tz_raw)],'color','k')
    plot([t_raw(end_idx) t_raw(end_idx)],[min(Tz_raw) max(Tz_raw)],'color','k')
    title('Tz')
end

end
```

```
function [ res ] = objFunc_Mat(a)
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here
global data M2

res=0;
for i=1:length(data(:,1))
    res=res+(data(i,1)-a*M2(i))^2;
end
end
```

C.2.4 Signal generation and data logging for the transfer function

Python script that triggers the generation of sinusoidal input commands on the Crazyflie and logs the resulting load cell data.

```
import time, sys, struct
from datetime import date
```

```python
from threading import Lock, Thread, Event
from struct import pack, unpack
from twisted.internet import reactor
from twisted.internet.protocol import DatagramProtocol
import pandas as pd
import socket
from math import pi, cos

sys.path.append("../../crazyflie-clients-python-master/lib")
import cflib
from cflib.crazyflie import Crazyflie
from cflib.crtp.crtpstack import CRTPPort
from cflib.crtp.crtpstack import CRTPPacket

import logging
logging.basicConfig(level=logging.ERROR)

# Config
UseLoadcell = 1
UseCF = 1        # Not yet implemented


# Running log of all FTS (=Force Torque Sensor) communications
FTSLog = []      # This is a list
FTSLock = Lock()

ThrustLog = []
ThrustLock = Lock()

TriggerEvent = Event()


def QuitHandler():
    print('')
    FTSLock.acquire()
    if len(FTSLog)>0:
        print "Saving FTSLog..."
        columnsNames = ['t','Fx','Fy','Fz','Tx','Ty','Tz']
        data = pd.DataFrame(data=FTSLog,columns=columnsNames, index=None)
        filename = 'FTS_data_{0}_{1}.csv'.format(date.today(),time.strftime('%H-%M-%S'))
        data.to_csv(path_or_buf=filename, index=None,float_format="%.6f")
        print "Data saved to %s. Bye :-)" % filename
    else:
        print "Nothing to save. Bye :-)"

class FTSLogger(DatagramProtocol):
    # Based on http://twistedmatrix.com/documents/12.3.0/core/howto/udp.html
    HOST = '192.168.1.200'
    PORT = 49152
    ADR = (HOST,PORT)

    def startProtocol(self):
        print('Connecting to loadcell under {0}:{1}...'.format(self.HOST,self.PORT))
        self.transport.connect(self.HOST,self.PORT)
        print "Requesting data from loadcell..."
        self.transport.write(pack('>HHI',0x1234,0x0002,0))

    def datagramReceived(self,datagram,address):
        now = time.time()
        #print "Received something..."
        seqNum1,seqNum2,status,Fx,Fy,Fz,Tx,Ty,Tz = unpack('>IIIiiiiii',
            datagram)
        apd = (now,Fx,Fy,Fz,Tx,Ty,Tz)
        FTSLock.acquire()
        FTSLog.append(apd)
        FTSLock.release()

    def connectionRefused(self):
        print "No one listening..."


class TriggerSine:
    """Script that allows to run motor commands"""
    def __init__(self, link_uri):
        """ Initialize and run the script with the specified link_uri
            """

        self._cf = Crazyflie()

        self._cf.connected.add_callback(self._connected)
        self._cf.disconnected.add_callback(self._disconnected)
        self._cf.connection_failed.add_callback(self._connection_failed
            )
        self._cf.connection_lost.add_callback(self._connection_lost)

        # Add callback (gets called when data comes in from the Flie)
        self._cf.add_port_callback(CRTPPort.SINE, self._receiving)

        self._cf.open_link(link_uri)

        self.thrustsetpoints = []
        self.thrustsetpointsLock = Lock()

        self.ampl = 0
        self.omega_l = 0

        self.TriggerTime = 0


        print "Connecting to %s" % link_uri

    def _connected(self, link_uri):
        """ This callback is called form the Crazyflie API when a
            Crazyflie
            has been connected and the TOCs have been downloaded."""

        # Start a separate thread.
        # Do not hijack the calling thread!
        Thread(target=self._trigger).start()

    def _connection_failed(self, link_uri, msg):
        """Callback when connection initial connection fails (i.e no
            Crazyflie
            at the speficied address)"""
        print "Connection to %s failed: %s" % (link_uri, msg)

    def _connection_lost(self, link_uri, msg):
        """Callback when disconnected after a connection has been made
            (i.e
            Crazyflie moves out of range)"""
        print "Connection to %s lost: %s" % (link_uri, msg)

    def _disconnected(self, link_uri):
        """Callback when the Crazyflie is disconnected (called in all cases)"""
        print "Disconnected from %s" % link_uri

    def _trigger(self):
        print "WARNING: Don't run this script before fastening the Crazyflie securely."

        while (1):
            TriggerEvent.clear()

            choice = raw_input("Enter S to start the experiment, D to start it with a default set of values apart from l (motor=3, ampl=20000, N=8000) or Q to end this program. Input: ")

            pk = CRTPPacket()
            pk.port = CRTPPort.SINE
            # Structure of the data that will be sent to the Crazyflie: (motor_nr, amplitude, omega_l, N)

            if (choice=="Q"):
                break
            elif (choice=="D"):
                motor_NR = 3
                self.ampl = 20000
                N = 8000

                if (N % 2 == 0):
                    rightBorder = N/2
                else:
                    rightBorder = (N-1)/2

                try:
                    l = int(raw_input("Enter l (pick one in the range [{0},{1}]). Input: ".format(0,rightBorder)))
                except ValueError:
                    print "Entered value not an int. Try again..."
                    continue
                if l<0 or l>rightBorder:
                    print "Not valid. Try again..."
                    continue

                self.omega_l = 2*pi*l/N
                self.omega_l = float(self.omega_l)
                pk.data = pack('<HHfI',motor_NR,self.ampl,self.omega_l,N)
                self._cf.send_packet(pk)
                self.TriggerTime = time.time()
                print "Trigger Time is %f" % self.TriggerTime
                print "Waiting for the experiment to finish..."

                # Wait until execution on the Crazyflie is finished.
                TriggerEvent.wait()

            elif (choice=="S"):
                print "The motors will be given the commands u[n]=A*cos(omega_l*n) for n=0,..,N with omega_l=2*pi*l/N"

                try:
                    motor_NR = int(raw_input("Enter the number of the motor you want to let run or enter 0 to address all motors. Input: "))
                except ValueError:
                    print "Entered value not an int. Try again..."
                    continue
                if motor_NR<0 or motor_NR>4:
                    print "Not valid. Try again..."
                    continue

                try:
                    self.ampl = int(raw_input("Enter the integer amplitude A (Max. 30000). Input: "))
                except ValueError:
                    print "Entered value not an int. Try again..."
                    continue
                if (self.ampl>30000 or self.ampl<0):
                    print "Amplitude not valid. Try again."
                    continue

                try:
                    N = int(raw_input("Enter N (Max. 4'294'967'295). Input: "))
                except ValueError:
                    print "Entered value not an int. Try again..."
                    continue
                if N>4294967295 or N<0:
                    print "Not valid. Try again..."
                    continue

                if (N % 2 == 0):
                    rightBorder = N/2
                else:
                    rightBorder = (N-1)/2

                try:
                    l = int(raw_input("Enter l (pick one in the range [{0},{1}]). Input: ".format(0,rightBorder)))
                except ValueError:
                    print "Entered value not an int. Try again..."
                    continue
                if l<0 or l>rightBorder:
                    print "Not valid. Try again..."
                    continue

                self.omega_l = 2*pi*l/N
                self.omega_l = float(self.omega_l)

                pk.data = pack('<HHfI',motor_NR,self.ampl,self.omega_l,N)
                self._cf.send_packet(pk)
                self.TriggerTime = time.time()
                print "Trigger Time is %f" % self.TriggerTime

                print "Waiting for the experiment to finish..."

                # Execution gets certainly until here...

                # Wait until execution on the Crazyflie is finished.
                TriggerEvent.wait()
```

```

            else:
                print "Unvalid choice. Try again..."

        print "Quit connection to the CF..."


        self._cf.close_link()

    def _receiving(self, packet):
        """
        Is called when a new packet comes in
        """
        now = time.time()
        if (packet.channel==1):
            # Save the ThrustLog
            print('')
            ThrustLock.acquire()
            if len(ThrustLog)>0:
                print "Saving ThrustLog..."
                thr_columnsNames = ['t','thrust_rec']
                thr_data = pd.DataFrame(data=ThrustLog,columns=thr_columnsNames, index=None)
                # print thr_data
                thr_filename = 'Thrust_data_{0}_{1}.csv'.format(date.today(),time.strftime('%H-%M-%S'))
                thr_data.to_csv(path_or_buf=thr_filename, index=None, float_format="%.6f")
                print "Data saved to %s. Bye :-)" % thr_filename
                del ThrustLog[:]
            else:
                print "Nothing to save."

            ThrustLock.release()

            # Set the event for the waiting trigger.
            print "Reset Trigger."
            TriggerEvent.set()


        else:
            thrust_rec = unpack('<H',packet.data)[0]
            # print thrust_rec
            apd = (now, thrust_rec)
            print apd
            ThrustLock.acquire()
            ThrustLog.append(apd)
            ThrustLock.release()



if __name__=="__main__":

    if UseLoadcell ==1:
        reactor.listenUDP(FTSLogger.PORT, FTSLogger())
        reactor.addSystemEventTrigger('before','shutdown',QuitHandler)

    # Initialize the low-level drivers (don't list the debug drivers)
    cflib.crtp.init_drivers(enable_debug_driver=False)
    # Scan for Crazyflies and use the first one found
    print "Scanning interfaces for Crazyflies..."
    available = cflib.crtp.scan_interfaces()
    print "Crazyflies found:"
    for i in available:
        print i[0]

    if len(available) > 0:
        reactor.callInThread(TriggerSine,available[0][0])
        #le = TriggerSine("radio://0/80/250K")
        reactor.run()
    else:
        print "No Crazyflies found, cannot run example"
```

Crazyflie module that starts, upon reception of the trigger packet from the Python script, applying sinusoidal inputs to the motors.

```
//Created by Julian FÃ¶rster

// App that is used to determine the CF's thrust map. It allows to pass
     sinusoidal inputs to every single motor.

#ifndef SINE_H_
#define SINE_H_

#include <stdbool.h>


//Member functions
void sineInit(void);


#endif /* SINE_H_ */
```

```
// Created by Julian FÃ¶rster

// App that is used to determine the CF's thrust map. It allows to pass
     sinusoidal inputs to every single motor.

#include <math.h>
#include <string.h>

#include "FreeRTOS.h"
#include "task.h"
#include "semphr.h"

#include "config.h"
#include "system.h"
#include "crtp.h"   //Used to send and receive information
#include "sine.h"

#include "lps25h.h"     // Don't know what that is.

#include "motors.h" // Contains function motorsSetRatio

struct sineCrtpValues {       // Inspired by struct defined in
    commander.c
    uint16_t motorID;
    uint16_t ampl;
    float omega;
    uint32_t N;
} __attribute__((packed));

```

```

// Member variables
static bool isInit;
static struct sineCrtpValues targetValue;
CRTPPacket logSetpoint;
CRTPPacket finished;


// Function prototypes
static void sineCrtpCB(CRTPPacket* pk);
static uint16_t limitThrust(int32_t value);     // Taken from stabilizer.c

// Task prototype
static void sineTask(void* param);

// Semaphore that signals when the sinusoidal input can be applied
static xSemaphoreHandle triggerSem;

//Member functions
void sineInit(void)
{
    // Only has to be initialized once...
    if(isInit) return;

    // We need crtp and the motors, so make sure they are initialized (if it is already initialized, nothing will happen)
    crtpInit();
    motorsInit();

    // Register the function that will be called when a comes in on the HELLO port
    crtpRegisterPortCB(CRTP_PORT_SINE, sineCrtpCB);

    // Set up CRTP
    logSetpoint.size = sizeof(uint16_t);
    logSetpoint.header = CRTP_HEADER(CRTP_PORT_SINE,0);
    finished.size = sizeof(uint8_t);
    finished.header = CRTP_HEADER(CRTP_PORT_SINE,1);

    // Create the task
    xTaskCreate(sineTask, (const signed char * const)SINE_TASK_NAME, SINE_TASK_STACKSIZE, NULL, SINE_TASK_PRI, NULL);

    vSemaphoreCreateBinary(triggerSem);
    xSemaphoreTake(triggerSem, portMAX_DELAY);

    isInit = true;
}

static void sineCrtpCB(CRTPPacket* pk)
{
    targetValue = *((struct sineCrtpValues*)pk->data);

    xSemaphoreGive(triggerSem);
}

static void sineTask(void* param) {

    uint16_t motor;
    uint16_t ampl;
    float omega;
    uint32_t N;

    int i;
    uint16_t freq = 500; //Hz       // How high should this frequency be?
    uint32_t thrust_raw;
    uint16_t thrust;
    uint16_t thrust_old = 0;

    uint32_t lastWakeTime;

    while (1) {

        xSemaphoreTake(triggerSem, portMAX_DELAY);

        lastWakeTime = xTaskGetTickCount();

        motor = targetValue.motorID;
        ampl = targetValue.ampl;
        omega = targetValue.omega;
        N = targetValue.N;

        i=0;

        switch (motor) {
        case 0:
            // All motors
            while (i<N) {
                vTaskDelayUntil(&lastWakeTime, F2T(freq));

                thrust_raw = ampl * cos(omega * i) + 30000;
                thrust = limitThrust(thrust_raw);

                if ((thrust_old>30000 && thrust<=30000)||(thrust_old<30000 && thrust>=30000)||i==0) {
                    memcpy(logSetpoint.data, &thrust, sizeof(uint16_t));
                    crtpSendPacket(&logSetpoint);
                }

                motorsSetRatio(MOTOR_M1, thrust);
                motorsSetRatio(MOTOR_M2, thrust);
                motorsSetRatio(MOTOR_M3, thrust);
                motorsSetRatio(MOTOR_M4, thrust);
                i++;

                thrust_old = thrust;
            }

            break;

        default:
            // One motor

            while (i<N) {
                vTaskDelayUntil(&lastWakeTime, F2T(freq));

                thrust_raw = ampl * cos(omega * i) + 30000;
                thrust = limitThrust(thrust_raw);

                if ((thrust_old>30000 && thrust<=30000)||(thrust_old<30000 && thrust>=30000)||i==0) {
                    memcpy(logSetpoint.data, &thrust, sizeof(uint16_t))
                        ;
                    crtpSendPacket(&logSetpoint);
                }

                motorsSetRatio(motor-1, thrust);
                i++;

                thrust_old = thrust;
            }

            break;
        }
        // Turn all motors off
        motorsSetRatio(MOTOR_M1, 0);
        motorsSetRatio(MOTOR_M2, 0);
        motorsSetRatio(MOTOR_M3, 0);
        motorsSetRatio(MOTOR_M4, 0);

        thrust = 0;
        memcpy(logSetpoint.data, &thrust, sizeof(uint16_t));
        crtpSendPacketBlock(&logSetpoint);

        vTaskDelay(1000);   // Wait for a second

        finished.data[0] = 'a';
        crtpSendPacketBlock(&finished);
    }
}


// Taken from stabilizer.c
static uint16_t limitThrust(int32_t value)
{
    if(value > UINT16_MAX)
    {
        value = UINT16_MAX;
    }
    else if(value < 0)
    {
        value = 0;
    }

    return (uint16_t)value;
}
```

C.2.5 Data processing for the transfer function

These files were used to process the data of the first series of measurements. The ones for the second series are identical apart from series specific constants such as the file names that include the data or the number of sample N.

```
%clc
%clear all
%close all

Use_Force=1;
fancy_fft=1;

% Parameters
```

```
Ts = 1/500;

% Vector containing all values of l

l = xlsread('Experiment_Index1.xlsx','H4:H46');

% Number of Frequencies to consider for system identification
% L = length(l);
L=33;

l=l(1:L);
H_hat = zeros(L,1);
Omega_l = zeros(L,1);



%%
% Get all estimates for frequency response

% l = 0,...,4
if L>=5
    for i=1:5
        if i==6
            [H_hat(i),Omega_l(i),time_indi,Ty]=prepare_data1_return_F('FTS_150615_1.csv',['CF_150615_' num2str(i) '.csv'],1,1,1,l(i),Ts,Use_Force,fancy_fft);
        else
            [H_hat(i),Omega_l(i)]=prepare_data1('FTS_150615_1.csv',['CF_150615_' num2str(i) '.csv'],0,0,0,l(i),Ts,Use_Force,fancy_fft);
        end
    end
end
%%
% l = 5,...,10
if L>=11
    for i=6:11
        if i==5
            [H_hat(i),Omega_l(i)]=prepare_data1('FTS_150615_2.csv',['CF_150615_' num2str(i) '.csv'],1,1,1,l(i),Ts,Use_Force,fancy_fft);
        else
            [H_hat(i),Omega_l(i)]=prepare_data1('FTS_150615_2.csv',['CF_150615_' num2str(i) '.csv'],0,0,0,l(i),Ts,Use_Force,fancy_fft);
        end
    end
end
%%
% l = 11,...,40
if L>=21
    for i=12:21
        if i==13
            [H_hat(i),Omega_l(i),time_indi,Ty1]=prepare_data1_return_F('FTS_150615_3.csv',['CF_150615_' num2str(i) '.csv'],0,0,0,l(i),Ts,Use_Force,fancy_fft);
        elseif i==18
            [H_hat(i),Omega_l(i),time_indi,Ty2]=prepare_data1_return_F('FTS_150615_3.csv',['CF_150615_' num2str(i) '.csv'],0,0,0,l(i),Ts,Use_Force,fancy_fft);
        else
```

```
            [H_hat(i),Omega_l(i)]=prepare_data1('FTS_150615_3.csv',['CF_150615_' num2str(i) '.csv'],0,0,0,l(i),Ts,Use_Force,fancy_fft);
        end
    end
end
%%
% l = 45,...,90
if L>=33
    for i=22:31
        [H_hat(i),Omega_l(i)]=prepare_data1('FTS_150615_4.csv',['CF_150615_' num2str(i) '.csv'],0,0,0,l(i),Ts,Use_Force,fancy_fft);
    end

    % l = 95
    [H_hat(32),Omega_l(32)]=prepare_data1('FTS_150615_5.csv',['CF_150615_' num2str(32) '.csv'],0,0,0,l(32),Ts,Use_Force,fancy_fft);

    % l = 100
    [H_hat(33),Omega_l(33)]=prepare_data1('FTS_150615_4.csv',['CF_150615_' num2str(33) '.csv'],0,0,0,l(33),Ts,Use_Force,fancy_fft);
end

% l = 110,...,2000
if L>=43
    for i=34:43
        [H_hat(i),Omega_l(i)]=prepare_data1('FTS_150615_5.csv',['CF_150615_' num2str(i) '.csv'],0,0,0,l(i),Ts,Use_Force,fancy_fft);
    end
end


%%
%close all

% Design parameters
% A=2;
% B=5;
A=2;
B=1;


F=zeros(2*L,A+B-1);
G=zeros(2*L,1);
W=zeros(2*L,2*L);
pos=1;
for i=1:L
    for idx_a=1:A-1
        % Real
        F(pos,idx_a)=abs(H_hat(i))*cos(angle(H_hat(i))-idx_a*Omega_l(i));
        % Imaginary
        F(pos+1,idx_a)=abs(H_hat(i))*sin(angle(H_hat(i))-idx_a*Omega_l(i));
    end
    for idx_b=0:B-1
        F(pos,idx_b+A)=(-1)*cos(idx_b*Omega_l(i));
        F(pos+1,idx_b+A)=sin(idx_b*Omega_l(i));
    end
    G(pos)=(-1)*abs(H_hat(i))*cos(angle(H_hat(i)));
    G(pos+1)=(-1)*abs(H_hat(i))*sin(angle(H_hat(i)));

    if i<=16
        W(pos,pos)=2;
        W(pos+1,pos+1)=2;
    else
        W(pos,pos)=0.5;
        W(pos+1,pos+1)=0.5;
    end

    pos=pos+2;
end


weighted=1;
if weighted==1
    F=W*F;
    G=W*G;
end
% Least squares solution
Theta = (F'*F)\(F'*G);

num=Theta(A:end)';
den=Theta(1:A-1)';
SYS1 = tf(num,[1 den],Ts,'variable','z^-1')

% Estimated frequency response
H=Omega_l;
for i=1:length(Omega_l)
    H(i)=freqResp(Omega_l(i),Theta,A);
end

% Plot magnitude response
figure
subplot(211)
%title('Magnitude response Exp. 1','FontSize',15)
hold on
scatter(Omega_l/Ts,20*log10(abs(H_hat)))
ylim([-135 -110])
plot(Omega_l/Ts,log10(abs(H))*20)
grid on
box on
xlabel('CT Frequency [$\frac{rad}{s}$]','Interpreter','latex','FontSize
    ',20)
ylabel('Magnitude [dB]','Interpreter','latex','FontSize',20)
h_legend=legend('Frequency response estimates','Weighted least squares
    fit');
set(h_legend,'FontSize',14)
h_legend.Interpreter='latex';
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';

% Plot phase response
subplot(212)
%title('Phase response Exp. 1','FontSize',15)
hold on
scatter(Omega_l/Ts,angle(H_hat)/pi*180)
plot(Omega_l/Ts,angle(H)/pi*180)
grid on
```

```
box on
xlabel('CT Frequency [$\frac{rad}{s}$]','Interpreter','latex','FontSize
    ',20)
ylabel('Phase [$^{\circ}$]','Interpreter','latex','FontSize',20)
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';


%% Make bode plot
% opts=bodeoptions;
% opts.TickLabel.FontSize=18;
% %opts.TickLabel.Interpreter='latex';
% opts.xlabel.FontSize=20;
% opts.ylabel.FontSize=20;
% opts.xlabel.Interpreter='latex';
% opts.ylabel.Interpreter='latex';
% opts.title.String=' ';

figure
bode(SYS1)

xlim([1 Omega_l(end)/Ts])
%title('Bode Plot Exp. 1','FontSize',15)
grid on
box on


%% Simulate response to different inputs

Tfinal=8;
t=0:Ts:Tfinal;

% Calculate impulse response
h=impulse(SYS1,t);
% figure
% plot(t,h)

% Create input signals
x1=20000*cos(2*pi*12/4000/0.002*t)+30000;
x2=20000*cos(2*pi*25/4000/0.002*t)+30000;
x3=20000*cos(2*t)+30000;

% Calculate outputs
y1=conv(x1,h);
y2=conv(x2,h);
y3=conv(x3,h);

% figure
% plotyy(t,x1,t,y1(1:length(t)))
% figure
% plotyy(t,x2,t,y2(1:length(t)))
% figure
% plotyy(t,x3,t,y3(1:length(t)))

% l=12
figure
y=lsim(SYS1,x1,t);
len_=min(length(time_indi),length(t));
[hAx,hLine1,hLine2]=plotyy(t(1:len_),x1(1:len_),[time_indi(1:len_),t(1:
    len_)'],[Ty1(1:len_),y(1:len_)]);
%title('Simulation of system response l=12','FontSize',14)
```

```matlab
xlabel('Time [s]','Interpreter','latex','FontSize',20)
ylabel(hAx(1),'Input command [-]','Interpreter','latex','FontSize',20)
ylabel(hAx(2),'Thrust [N]','Interpreter','latex','FontSize',20)
set(hAx(1),'ylim',[5000 55000])
set(hAx(2),'ylim',[mean(Ty1)-0.2 mean(Ty1)+0.2])
hLine1.Color = [255 64 0]/255;
set(hAx(1),'ycolor',[255 64 0]/255)
hLine2(1).Color = [0 140 204]/255;
hLine2(2).Color = 'k';
hLine2(2).LineWidth = 3;
h_legend=legend('Input','Measured output','Simulated output');
set(h_legend,'FontSize',14)
h_legend.Interpreter='latex';
hAx(1).FontSize=18;
hAx(2).FontSize=18;
hAx(1).TickLabelInterpreter='latex';
hAx(2).TickLabelInterpreter='latex';

% l=25
figure
y=lsim(SYS1,x2,t);
len_=min(length(time_indi),length(t));
[hAx,hLine1,hLine2]=plotyy(t(1:len_),x2(1:len_),[time_indi(1:len_),t(1:
    len_)'],[Ty2(1:len_),y(1:len_)]);
%title('Simulation of system response l=25','FontSize',14)
xlabel('Time [s]','Interpreter','latex','FontSize',20)
ylabel(hAx(1),'Input command [-]','Interpreter','latex','FontSize',20)
ylabel(hAx(2),'Thrust [N]','Interpreter','latex','FontSize',20)
set(hAx(1),'ylim',[5000 55000])
set(hAx(2),'ylim',[mean(Ty2)-0.2 mean(Ty2)+0.2])
hLine1.Color = [255 64 0]/255;
set(hAx(1),'ycolor',[255 64 0]/255)
hLine2(1).Color = [0 140 204]/255;
hLine2(2).Color = 'k';
hLine2(2).LineWidth = 3;
h_legend=legend('Input','Measured output','Simulated output');
set(h_legend,'FontSize',14)
h_legend.Interpreter='latex';
hAx(1).FontSize=18;
hAx(2).FontSize=18;
hAx(1).TickLabelInterpreter='latex';
hAx(2).TickLabelInterpreter='latex';
```

```matlab
function [H_hat,Omega_l] = prepare_data1(filename_FT,filename_CF,
    plot_FT,plot_CF,plot_fft,l,Ts,Use_Force,fancy_fft)
%disp(['l=' num2str(l)])
% Import FT data
FT_data = csvread(filename_FT,1,0);
% Looking at torque around y axis.
Ty_raw = FT_data(:,6)*(-1);
t_FT = FT_data(:,1);

% Get rid of bias
bias=mean(Ty_raw(1:1000));
Ty_raw=Ty_raw-bias;

% Adapt unit
Ty_raw=Ty_raw/1000000;
% Now the unit is Nm
if Use_Force==1
    d=92e-3/2;   % Distance from propeller axis to loadcell origin.
    Ty_raw=Ty_raw/d;
    % Now the unit is N
end

%plot(Ty_raw)

% Import motor input data
CF_data = csvread(filename_CF,1,0);
t_CF = CF_data(:,1);
refPoints = CF_data(:,2);

for i=1:length(t_FT)-1
    if t_FT(i)<=t_CF(1) && t_FT(i+1)>t_CF(1)
        start_idx=i;
    end
    if t_FT(i)<t_CF(end) && t_FT(i+1)>=t_CF(end)
        end_idx=i+1;
    end
end

% Relevant FT data
t_FT_rel = t_FT(start_idx:end_idx);
Ty_rel = Ty_raw(start_idx:end_idx);

% Adjust timescales
t_FT_rel = t_FT_rel - t_CF(1);
t_CF = t_CF - t_CF(1);

% New timescale
%Ts=mean(diff(t_FT_rel))
% Individual N used to calculate the motor_input
N_indi=ceil(t_CF(end)/Ts);
n_indi=0:N_indi-1;
time_indi=(0:Ts:(N_indi-1)*Ts)';

% Interpolate FT data
Ty=interp1(t_FT_rel,Ty_rel,time_indi);

% Reconstruct motor input command
motor_input = (refPoints(1)-30000) * cos(2*pi*l/N_indi*n_indi) + 30000;

if plot_FT==1
    figure
    plot(time_indi,Ty)
    grid on
    box on
    xlabel('Time [s]','Interpreter','LaTex','fontsize',20)
    ylimval=max(abs(min(Ty)),abs(max(Ty)))+0.015;
    ylim([-ylimval+mean(Ty) ylimval+mean(Ty)])
    if Use_Force~=1
        ylabel('Torque [Nm]','Interpreter','LaTex','fontsize',20)
    else
        ylabel('Thrust [N]','Interpreter','LaTex','fontsize',20)
    end
    title(['l = ' num2str(l)],'FontSize',14)
    ax=gca;
    ax.FontSize=18;
    ax.TickLabelInterpreter='latex';
end
if plot_CF==1
    figure
    scatter(t_CF,refPoints)
    hold on
    plot(time_indi,motor_input)
    grid on
    box on
    xlabel('Time[s]','Interpreter','LaTex','fontsize',20)
    ylabel('Input command [-]','Interpreter','LaTex','fontsize',20)
    ylim([5000 55000])
    title(['l = ' num2str(l)],'FontSize',14)
    ax=gca;
    ax.FontSize=18;
    ax.TickLabelInterpreter='latex';
end

% Calculate frequency response estimate
N_T = 500;
Omega_l=2*pi*l/N_indi;

if fancy_fft==1
    % Get correct region for fft (Mike's idea)
    if l==0 || l==1
        N_fft=N_indi-N_T;
    else
        N_period=round(4000/l);
        number_periods=floor((N_indi-N_T)/N_period);
        N_fft=N_period*number_periods;
    end

    Y_m=fft(Ty(N_T:N_T+N_fft-1));
    U_e=fft(motor_input(N_T:N_T+N_fft-1));

    % Get rid of middle value
    % if l~=0
    %     Y_m(1)=0;
    %     U_e(1)=0;
    % end

    % Plot ffts
    if plot_fft==1
        figure
        n=0:length(motor_input(N_T:N_T+N_fft-1))-1;
        Omega=2*pi*n/length(n);
        stem(Omega,abs(U_e))
        xlim([0 0.1])
        xlabel('l','FontSize',14)
        ylabel('Magnitude of FFT','FontSize',14)
        title(['Input l = ' num2str(l)],'FontSize',14)
        figure
        stem(Omega,abs(Y_m))
        xlim([0 0.1])
        xlabel('l','FontSize',14)
        ylabel('Magnitude of FFT','FontSize',14)
        title(['Output l = ' num2str(l)],'FontSize',14)
    end

    %H_hat = Y_m(l_fft+1)/U_e(l_fft+1);

    [~, idx_max_y]=max(abs(Y_m(2:end)));
    [~, idx_max_u]=max(abs(U_e(2:end)));


    if l==0
        idx_max_y=0;
        idx_max_u=0;
    end


    H_hat = Y_m(idx_max_y+1)/U_e(idx_max_u+1);  % +1 because searching the max started at index 2
else
    Y_m = fft(Ty(N_T:end));
    U_e = fft(motor_input(N_T:end));

    N_fft = length(Ty(N_T:end));
    l_fft=round(Omega_l/2/pi*N_fft);

    % Get rid of middle value
    %     if l~=0
    %         Y_m(1)=0;
    %         U_e(1)=0;
    %     end

    % Plot ffts
    if plot_fft==1
        figure
        n=0:length(motor_input(N_T:end))-1;
        Omega=2*pi*n/length(n);
        stem(Omega,abs(U_e))
        xlim([0 0.1])
        xlabel('\Omega [rad]','FontSize',14)
        ylabel('Magnitude of FFT','FontSize',14)
        title(['Input l = ' num2str(l)],'FontSize',14)
        figure
        stem(Omega,abs(Y_m))
        xlim([0 0.1])
        xlabel('\Omega [rad]','FontSize',14)
        ylabel('Magnitude of FFT','FontSize',14)
        title(['Output l = ' num2str(l)],'FontSize',14)
    end

    %H_hat = Y_m(l_fft+1)/U_e(l_fft+1);

    [~, idx_max_y]=max(abs(Y_m(2:end)));
    [~, idx_max_u]=max(abs(U_e(2:end)));

    if l==0
        idx_max_y=0;
        idx_max_u=0;
    end

    H_hat = Y_m(idx_max_y+1)/U_e(idx_max_u+1);

end

end
```

The function `prepare_data1_return_F` is identical to the function `prepare_data1` apart from that it returns the time and force vectors in addition.

```
function res = freqResp(Omega,Theta,A)
num=0;
for i=A:length(Theta)
    num=num+Theta(i)*exp(-1i*Omega*(i-A));
end
den=1;
```

```
for i=1:A-1
    den=den+Theta(i)*exp(-1i*Omega*i);
end
res=num/den;
end
```

C.2.6 Ping Test

Python script that can be used to initiate the ping test.

```
# -*- coding: utf-8 -*-
#
# Written by Julian Foerster based on the example ramp.py written by
    Bitcraze
#

"""
    Programm that allows measure the latency of the connection between
        Crazyradio PA and Crazyflie
"""

import time, sys
from threading import Thread, Lock, Event
from numpy import mean, std

sys.path.append("../lib")
import cflib
from cflib.crazyflie import Crazyflie
from cflib.crtp.crtpstack import CRTPPort
from cflib.crtp.crtpstack import CRTPPacket
from twisted.internet import reactor


import logging
logging.basicConfig(level=logging.ERROR)

import struct

WaitLock = Lock()

class Ping:
    """Script that allows to run motor commands"""
    def __init__(self, link_uri):
        """ Initialize and run the script with the specified link_uri
            """

        self._cf = Crazyflie()

        self._cf.connected.add_callback(self._connected)
        self._cf.disconnected.add_callback(self._disconnected)
        self._cf.connection_failed.add_callback(self._connection_failed
            )
        self._cf.connection_lost.add_callback(self._connection_lost)

        # Add callback (gets called when data comes in from the Flie)
        self._cf.add_port_callback(CRTPPort.PING, self._receiving)

        self._cf.open_link(link_uri)

        self.rxTime = 0
        self.txTime = 0
        self.times = []
        self.timesLock = Lock()

        print "Connecting to %s" % link_uri

    def _connected(self, link_uri):
        """ This callback is called form the Crazyflie API when a Crazyflie
            has been connected and the TOCs have been downloaded."""

        # Start a separate thread to do the motor test.
        # Do not hijack the calling thread!
        Thread(target=self._send_ping).start()

    def _connection_failed(self, link_uri, msg):
        """Callback when connection initial connection fails (i.e no Crazyflie
            at the speficied address)"""
        print "Connection to %s failed: %s" % (link_uri, msg)

    def _connection_lost(self, link_uri, msg):
        """Callback when disconnected after a connection has been made (i.e
            Crazyflie moves out of range)"""
        print "Connection to %s lost: %s" % (link_uri, msg)

    def _disconnected(self, link_uri):
        """Callback when the Crazyflie is disconnected (called in all cases)"""
        print "Disconnected from %s" % link_uri

    def _send_ping(self):
        self._cf.commander.send_setpoint(0,0,0,0)
        a=0     # Index variable

        choice = raw_input("Enter the number of executions. Input: ")
        choice = int(choice)
        print "0 %"
        while (a<=choice):
            a=a+1
            if ((100*float(a)/float(choice)) % 10 == 0):
                print "%i %%" % (100*a/choice)
            pk = CRTPPacket()
            pk.port = CRTPPort.PING
            WaitLock.acquire()
            self.txTime = time.time()
            self._cf.send_packet(pk)
            WaitLock.release()
            time.sleep(0.1)

        self.timesLock.acquire()
        mean_data = mean(self.times)
        std_data = std(self.times)

        print "Mean: {}\nStandard Deviation: {}".format(mean_data, std_data)

        self._cf.close_link()

    def _receiving(self, packet):
        """
        Is called when a new packet comes in
        """
        self.rxTime = time.time()
        WaitLock.acquire()

        Difference = self.rxTime - self.txTime

        self.timesLock.acquire()
        self.times.append(Difference)
        self.timesLock.release()
        WaitLock.release()

if __name__ == '__main__':
    # Initialize the low-level drivers (don't list the debug drivers)
    cflib.crtp.init_drivers(enable_debug_driver=False)
    # Scan for Crazyflies and use the first one found
    print "Scanning interfaces for Crazyflies..."
    available = cflib.crtp.scan_interfaces()
    print "Crazyflies found:"
    for i in available:
        print i[0]

    if len(available) > 0:
        reactor.callInThread(Ping,available[0][0])
        reactor.run()
    else:
        print "No Crazyflies found, cannot run example"
```

Crazyflie firmware module that responds to an incoming packet by immediately sending back an empty packet.

```





#ifndef PING_H_
#define PING_H_



//Member functions
void pingInit(void);




#endif /* PING_H_ */
```

```
//Created by Julian FÃ¶rster

/* Module that answers on an incoming packet right away in order to
    measure the Crazyflie's response time. */

#include "FreeRTOS.h"
#include "task.h"


#include "crtp.h"   //Used to send and receive information
#include "ping.h"
```

```c
#include "debug.h"
#include "console.h"

//Member variables
static bool isInit = false;
CRTPPacket answerping;

// Function prototype
static void pingCrtpCB(CRTPPacket* pk);

//Member functions
void pingInit(void)
{
    // Only has to be initialized once...
    if(isInit)
        return;

    // We need crtp, so make sure it is initialized (if it is already
        initialized, nothing will happen
    crtpInit();
    // Register the function that will be called when a comes in on the
         HELLO port
    crtpRegisterPortCB(CRTP_PORT_PING, pingCrtpCB);

    answerping.size = 0;
    // Make sure that the packet we send back reaches the right
        function within the client
    answerping.header = CRTP_HEADER(CRTP_PORT_PING, 0);

    isInit = true;
}

static void pingCrtpCB(CRTPPacket* pk)
{
    // Send answer CRTPPacket
    if (crtpSendPacket(&answerping) == pdTRUE) {
        // Packet was sent successfully
    }
}
```

C.3 Code in connection with drag coefficients

C.3.1 Data processing of the wind tunnel experiment data

```matlab
clc
clear all
close all

format long
% All forces except from aerodynamic ones occur in the measurement. So
    we
% subtract this force from the other two

%% Import all forces
F = zeros(7*3,16);

% Get vector with all filenames
[~,filenames1,~] = xlsread('ProofofConcept5.xlsx','C25:C43');
```

```
[~,filenames2,~] = xlsread('ProofofConcept5.xlsx','H25:H43');
[~,filenames3,~] = xlsread('ProofofConcept5.xlsx','M25:M43');
[~,filenames4,~] = xlsread('ProofofConcept5.xlsx','R25:R43');
[~,filenames5,~] = xlsread('ProofofConcept5.xlsx','W25:W43');
[~,filenames6,~] = xlsread('ProofofConcept5.xlsx','W6:W22');
filenames=[filenames1; filenames2; filenames3; filenames4; filenames5;
    filenames6];

% Vector with input commands
input_vec = [10000 20000 30000 37300 40000 50000];


% 0 V
spalte = 1;
start=1;

filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,2000,4000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,7500,9000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,9000,11500,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,7000,10000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,6000,9000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,5000,8000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,5000,10000,0);

% 5 V, 0 deg
spalte=2;
start=8;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,2000,4000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,6000,9000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,14000,17000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,8000,11000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,6000,8000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,8000,11000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,6000,10000,0);
```

```

% 6 V, 0 deg
spalte=3;
start=15;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,1500,4000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data_end(filename
    ,6000,9000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,6000,9000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,6000,8000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,7000,10000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,10000,13000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,6000,9000,0);

% 8 V, 0 deg
spalte=4;
start=22;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,2000,4000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,6000,9000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,6000,9000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,9000,12000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,6400,10200,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,6000,10000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,8000,13000,0);

% 10 V, 0 deg
spalte=5;
start=29;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,2000,6000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,5000,9000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
```

```
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,7000,10000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,7000,10000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,7000,11000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,6000,9000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,8000,13000,0);

% 12 V, 0 deg
spalte=6;
start=36;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,2000,5000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,5000,9000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,7000,12000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,7600,11400,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,8000,12000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,9000,14000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,6000,10600,0);

% 5 V, 48,4 deg
spalte=7;
start=43;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,4000,8000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,7000,11000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,6000,10000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,11000,15000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,9000,15000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,6000,11000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
```

```
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,6000,10000,0);

% 6 V, 48,4 deg
spalte=8;
start=50;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,3000,7000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,7000,11000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,7000,12000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,8000,13000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,10000,13000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,9000,13000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,8000,11900,0);

% 8 V, 48,4 deg
spalte=9;
start=57;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,3000,7000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,7000,11000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,6000,11000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,7300,11300,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,10000,15000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,9000,14000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,8000,12000,0);

% 10 V, 48,4 deg
spalte=10;
start=64;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,2500,6500,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,7300,12000,0);
```

```
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,7000,11000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,8000,13000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,11000,15000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,8000,14000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,9000,14000,0);

% 12 V, 48,4 deg
spalte=11;
start=71;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,2000,7000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,8000,13000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,7000,12000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,7000,12000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,9000,14000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,8000,14000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,13000,19600,0);

% 5 V, 77.9 deg
spalte=12;
start=78;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,4000,8000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,12000,16000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,9000,15000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,7000,13000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,8000,16000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,10000,14000,0);
```

```
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,8000,14000,0);

% 6 V, 77.9 deg
spalte=13;
start=85;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,3000,9000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,8000,13000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,8000,14000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,8000,15000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,9000,15000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,8000,15000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,7000,14000,0);

% 8 V, 77.9 deg
spalte=14;
start=92;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,3000,8000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,8000,13000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,8000,14000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,10000,16000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,8000,16000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,8000,15000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,8000,14000,0);

% 10 V, 77.9 deg
spalte=15;
start=99;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,3000,10000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
```

```
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,9000,17000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,9000,15000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,10000,17000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,10000,17000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,9000,17000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,11000,17000,0);

% 12 V, 77.9 deg
spalte=16;
start=106;
filename=['Data/' char(filenames(start)) '.csv'];
[F(1,spalte),F(2,spalte),F(3,spalte)]=prepare_data(filename
    ,6000,11000,0);
filename=['Data/' char(filenames(start+1)) '.csv'];
[F(4,spalte),F(5,spalte),F(6,spalte)]=prepare_data(filename
    ,10000,16000,0);
filename=['Data/' char(filenames(start+2)) '.csv'];
[F(7,spalte),F(8,spalte),F(9,spalte)]=prepare_data(filename
    ,10000,17000,0);
filename=['Data/' char(filenames(start+3)) '.csv'];
[F(10,spalte),F(11,spalte),F(12,spalte)]=prepare_data(filename
    ,9000,16000,0);
filename=['Data/' char(filenames(start+4)) '.csv'];
[F(13,spalte),F(14,spalte),F(15,spalte)]=prepare_data(filename
    ,8000,16000,0);
filename=['Data/' char(filenames(start+5)) '.csv'];
[F(16,spalte),F(17,spalte),F(18,spalte)]=prepare_data(filename
    ,11000,20000,0);
filename=['Data/' char(filenames(start+6)) '.csv'];
[F(19,spalte),F(20,spalte),F(21,spalte)]=prepare_data(filename
    ,9000,19000,0);

%% Matrix that contains the corrected drag forces for each experiment

F_drag = zeros(3*6,15);
% F_drag = F_tot - F_thr

for i=1:size(F_drag,2)
    for j=1:3:size(F_drag,1)
        F_drag(j:j+2,i) = F(j+3:j+3+2,i+1) - F(j+3:j+3+2,1);
    end
end

% Read in Theta_sigmas
Thetas_raw = xlsread('ProofofConcept5.xlsx','C9:Q19');
Thetas = zeros(6,15);
j=1;
for i=1:size(Thetas,1)
    Thetas(i,:) = Thetas_raw(j,:);
    j=j+2;
end
```

```
Thetas=Thetas/2/60*2*pi;    % Change unit to rad/s
Theta_sigma=Thetas*4;

%%
% Wind velocities
v_abs = [1.5 2.25 (2.8+3.6)/2 (4.2+3.1)/2 (3.7+5.1)/2 (1+1.8)/2
     (1.6+2.8)/2 (2+3.7)/2 (2.8+4.3)/2 (3.3+5)/2 (1.4+1.8)/2 (1.8+2.5)/2
      (2.2+3.4)/2 (2.6+4)/2 (3+4.5)/2]; % [m/s]
v = zeros(3,length(v_abs));
angles = [0 0 0 0 0 48.4 48.4 48.4 48.4 48.4 77.9 77.9 77.9 77.9
     77.9]/180*pi;
for i=1:length(v_abs)
        v(1:3,i) = [v_abs(i)*sin(angles(i))*cos(0.4967);-v_abs(i)*sin(
            angles(i))*sin(0.4967);v_abs(i)*cos(angles(i))];
end

% Calculate drag coefficients kappa_par (z direction)
kappa_par=zeros(6,15);
% Rows: input commands, Columns: Angles and wind speeds
for i=1:size(F_drag,2)
    for j=1:size(F_drag,1)/3
        kappa_par(j,i) = F_drag(j*3,i)/Theta_sigma(j,i)/v(3,i);
    end
end

% Calculate drag coefficients kappa_orth (x,y direction) each alone
kappa_orth=zeros(2*6,10);
% Rows: input commands (two rows per command, one for x and one for y),
% Columns: angles and wind speeds (excluding angle 0 deg)
idx_kappa=1;
for i=6:size(F_drag,2)
    for j=1:size(F_drag,1)/3
        kappa_orth(idx_kappa,i-5) = F_drag(j*3-2,i)/Theta_sigma(j,i)/v
            (1,i);    % x
        kappa_orth(idx_kappa+1,i-5) = F_drag(j*3-1,i)/Theta_sigma(j,i)/
            v(2,i);    % y
        idx_kappa = idx_kappa + 2;
    end
    idx_kappa = 1;
end

% Calculate drag coefficients in x and y direction with LS for each
    wind
% velocity and each angular velocity alone
kappa_orth_LS=zeros(6,10);
for i=6:size(F_drag,2)
    for j=1:size(F_drag,1)/3
        y=[F_drag(j*3-2,i)/Theta_sigma(j,i)/v(1,i); F_drag(j*3-1,i)/
            Theta_sigma(j,i)/v(2,i)];
        H=[1;1];
        kappa_orth_LS(j,i-5)=(H'*H)\(H'*y);
    end
end

% Calculate drag coefficients in x and y direction with LS for all wind
% velocities but each angular velocity alone (still separately for each
% wind angle)
kappa_orth_LS_allWinds=zeros(6,2);
y48_4 = zeros(2*5,1);
y77_9 = y48_4;
H=ones(10,1);
idx_force=6;
```

```
for j=1:size(F_drag,1)/3    % Iterate RPMs
    for m=1:2:10    % Iterate wind speeds
        y48_4(m) = F_drag(j*3-2,idx_force)/Theta_sigma(j,idx_force)/v
            (1,idx_force);
        y48_4(m+1) = F_drag(j*3-1,idx_force)/Theta_sigma(j,idx_force)/v
            (2,idx_force);
        y77_9(m) = F_drag(j*3-2,idx_force+5)/Theta_sigma(j,idx_force+5)
            /v(1,idx_force+5);
        y77_9(m+1) = F_drag(j*3-1,idx_force+5)/Theta_sigma(j,idx_force
            +5)/v(2,idx_force+5);
        idx_force=idx_force+1;
    end
    kappa_orth_LS_allWinds(j,1)=(H'*H)\(H'*y48_4);
    kappa_orth_LS_allWinds(j,2)=(H'*H)\(H'*y77_9);
    idx_force=6;
end


% Calculate drag coefficients in x and y direction with LS for all
    angular
% velocities but each wind velocity alone
kappa_orth_LS_allInputs=zeros(2,5);
y48_4 = zeros(2*6,1);
y77_9 = y48_4;
H=ones(12,1);
idx_force=1;
for i=1:5       % Iterate wind speeds
    for m=1:2:12
        y48_4(m) = F_drag(idx_force*3-2,i+5)/Theta_sigma(idx_force,i+5)
            /v(1,i+5);
        y48_4(m+1) = F_drag(idx_force*3-1,i+5)/Theta_sigma(idx_force,i
            +5)/v(2,i+5);
        y77_9(m) = F_drag(idx_force*3-2,i+10)/Theta_sigma(idx_force,i
            +10)/v(1,i+10);
        y77_9(m+1) = F_drag(idx_force*3-1,i+10)/Theta_sigma(idx_force,i
            +10)/v(2,i+10);
        idx_force=idx_force+1;
    end
    kappa_orth_LS_allInputs(1,i) = (H'*H)\(H'*y48_4);
    kappa_orth_LS_allInputs(2,i) = (H'*H)\(H'*y77_9);
    idx_force=1;
end

% Calculate drag coefficients in x and y direction with LS for all
    angular
% velocities and all wind velocities
kappa_orth_LS_all=zeros(2,1);
y48_4 = zeros(2*6*5,1);
y77_9 = y48_4;
H=ones(length(y48_4),1);
idx_y=1;
for i=6:10  % Iterate Wind velocities
    for j=1:6   % Iterate input commands
        y48_4(idx_y) = F_drag(j*3-2,i)/Theta_sigma(j,i)/v(1,i);
        y48_4(idx_y+1) = F_drag(j*3-1,i)/Theta_sigma(j,i)/v(2,i);
        y77_9(idx_y) = F_drag(j*3-2,i+5)/Theta_sigma(j,i+5)/v(1,i+5);
        y77_9(idx_y+1) = F_drag(j*3-1,i+5)/Theta_sigma(j,i+5)/v(2,i+5);
        idx_y=idx_y+2;
    end
end
kappa_orth_LS_all(1) = (H'*H)\(H'*y48_4);
kappa_orth_LS_all(2) = (H'*H)\(H'*y77_9);
```

```

%% LS new force model

% y vector containing all the forces
y=zeros(numel(F_drag),1);
H=zeros(length(y),12);
idx_y=1;
idx_3=1;
for j=1:size(F_drag,2)
    for i=1:size(F_drag,1)    % 1:6*3
        y(idx_y) = F_drag(i,j);

        if mod(idx_y,3)==0
            idx_H = 2;
        else
            idx_H = 1;
        end

        % K_aero1
        H(idx_y,idx_H) = Theta_sigma(ceil(i/3),j) * v(idx_3,j);
        % K_aero2
        H(idx_y,idx_H+2) = v(idx_3,j)^2;
        % K_aero3
        H(idx_y,idx_H+4) = Theta_sigma(ceil(i/3),j)^2;
        % K_aero4
        H(idx_y,idx_H+6) = v(idx_3,j);
        % K_aero5
        H(idx_y,idx_H+8) = Theta_sigma(ceil(i/3),j);
        % K_aero6
        H(idx_y,idx_H+10) = 1;

        idx_y=idx_y+1;
        idx_3=idx_3+1;
        if idx_3==4
            idx_3=1;
        end
    end
end

%H_sub=H-mean(H,1);
%H_sub=H_sub

%K=(H'*W*H)\(H'*W*y)
K=H\y;
K_aero1=diag([K(1) K(1) K(2)]);
K_aero2=diag([K(3) K(3) K(4)]);
K_aero3=[K(5);K(5);K(6)];
K_aero4=diag([K(7) K(7) K(8)]);
K_aero5=[K(9);K(9);K(10)];
K_aero6=[K(11);K(11);K(12)];

% Weight all coefficients using the variables they are multiplied with
    in
% order to make them comparable.
norm1=mean(mean(Theta_sigma))*mean(v_abs)
norm2=mean(v_abs)^2
norm3=mean(mean(Theta_sigma))
norm4=mean(v_abs)
norm5=mean(mean(Theta_sigma))
norm6=1
norm_sum=norm1+norm2+norm3+norm4+norm5+norm6

```

```
K_norm1=K_aero1*norm1/norm_sum
K_norm2=K_aero2*norm2/norm_sum
K_norm3=K_aero3*norm3/norm_sum
K_norm4=K_aero4*norm4/norm_sum
K_norm5=K_aero5*norm5/norm_sum
K_norm6=K_aero6*norm6/norm_sum


F_drag_calc = F_drag;
for j=1:size(F_drag,2)
    for i=1:size(F_drag,1)/3    % 1:6
        F_drag_calc(i*3-2:i*3,j) = K_aero1*Theta_sigma(i,j)*v(:,j) + K_aero2*(v(:,j).^2) + K_aero3*Theta_sigma(i,j)^2 + K_aero4*v(:,j) + K_aero5*Theta_sigma(i,j) + K_aero6;
    end
end

%% Forces based on old force model

F_drag_calc_simpleModel = F_drag;
K_aero_simple = diag([mean(kappa_orth_LS_all) mean(kappa_orth_LS_all) mean(mean(kappa_par))]);
%K_aero_simple=diag([20 20 10]);
for j=1:size(F_drag,2)
    for i=1:size(F_drag,1)/3    % 1:6
        F_drag_calc_simpleModel(i*3-2:i*3,j) = K_aero_simple*Theta_sigma(i,j)*v(:,j);
    end
end

%% Calculate sum of squared errors
Err_simpleModel=sum(sum((F_drag-F_drag_calc_simpleModel).^2))

Err_newModel=sum(sum((F_drag-F_drag_calc).^2))

%% Fitting - whole matrix forcing symetries

% The matrix used looks like
%            ( a b c )
% K_aero = ( b  a  c )
%            ( d d e )

y=zeros(numel(F_drag),1);
H=zeros(length(y),5);   % Five coefficients
idx_y=1;
for j=1:size(F_drag,2)  % iterate wind speeds
    for i=1:size(F_drag,1)/3  % iterate input commands
        y(idx_y:idx_y+2)=F_drag(3*i-2:3*i,j);

        H(idx_y,:)=Theta_sigma(i,j)*[v(1,j),v(2,j),0,0,v(3,j)];
        H(idx_y+1,:)=Theta_sigma(i,j)*[v(2,j),v(1,j),0,0,v(3,j)];
        H(idx_y+2,:)=Theta_sigma(i,j)*[0,0,v(1,j)+v(2,j),v(3,j),0];

        idx_y=idx_y+3;
    end
end

K=(H'*H)\(H'*y);
K_aero4_1 = [K(1),K(2),K(5);K(2),K(1),K(5);K(3),K(3),K(4)];


K2=H\y;
```

```
K_aero4_2 = [K2(1),K2(2),K2(5);K2(2),K2(1),K2(5);K2(3),K2(3),K2(4)]


F_drag_calc4 = F_drag;
for j=1:size(F_drag,2)
    for i=1:size(F_drag,1)/3    % 1:6
        F_drag_calc4(i*3-2:i*3,j) = K_aero4_1*Theta_sigma(i,j)*v(:,j);
    end
end

Err_newModel4=sum(sum((F_drag-F_drag_calc4).^2))

%% Surface plot of force in x - data

min_v=min(v_abs);
max_v=max(v_abs);
min_theta=min(min(Theta_sigma));
max_theta=max(max(Theta_sigma));
min_F=min(min(F_drag));
max_F=max(max(F_drag));

% Absolute wind velocity vs. Theta_sigma
figure
labelx='$|v|$ $[\frac{m}{s}]$';
labely='$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$';
labelz='$f_{\mathrm{a,x}} [N]$';
size_font_labels=20;
% 0 deg
%surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,F_drag(1:3:end,1:5))
surf_wind=v_abs(1:5);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],Theta_sigma(:,1:5),F_drag(1:3:end,1:5))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Data, wind from $0^\circ$','interpreter','latex','FontSize',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
% 48.4 deg
figure
%surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,F_drag(1:3:end,6:10))
surf_wind=v_abs(6:10);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],Theta_sigma(:,6:10),F_drag(1:3:end,6:10))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Data, wind from $48.4^\circ$','interpreter','latex','FontSize',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
```

```
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
% 77.9 deg
figure
%surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag(1:3:end,11:15))
surf_wind=v_abs(11:15);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,11:15),F_drag(1:3:end,11:15))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Data, wind from $77.9^\circ$','interpreter','latex','FontSize
    ',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=16;
ax.TickLabelInterpreter='latex';


%% Surface plot of force in y - data

min_v=min(v_abs);
max_v=max(v_abs);
min_theta=min(min(Theta_sigma));
max_theta=max(max(Theta_sigma));
min_F=min(min(F_drag));
max_F=max(max(F_drag));

% Absolute wind velocity vs. Theta_sigma
figure
labelx='$|v|$ $[\frac{m}{s}]$';
labely='$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$';
labelz='$f_{\mathrm{a,y}} [N]$';
size_font_labels=20;
% 0 deg
%surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag(1:3:end,1:5))
surf_wind=v_abs(1:5);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,1:5),F_drag(2:3:end,1:5))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Data, wind from $0^\circ$','interpreter','latex','FontSize',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
% 48.4 deg
figure
%surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag(1:3:end,6:10))
```

```
surf_wind=v_abs(6:10);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,6:10),F_drag(2:3:end,6:10))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Data, wind from $48.4^\circ$','interpreter','latex','FontSize
    ',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
% 77.9 deg
figure
%surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag(1:3:end,11:15))
surf_wind=v_abs(11:15);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,11:15),F_drag(2:3:end,11:15))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Data, wind from $77.9^\circ$','interpreter','latex','FontSize
    ',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=16;
ax.TickLabelInterpreter='latex';

%% Surface plot of force in z - data

min_v=min(v_abs);
max_v=max(v_abs);
min_theta=min(min(Theta_sigma));
max_theta=max(max(Theta_sigma));
min_F=min(min(F_drag));
max_F=max(max(F_drag));

% Absolute wind velocity vs. Theta_sigma
figure
labelx='$|v|$ $[\frac{m}{s}]$';
labely='$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$';
labelz='$f_{\mathrm{a,z}} [N]$';
size_font_labels=20;
% 0 deg
%surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag(1:3:end,1:5))
surf_wind=v_abs(1:5);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,1:5),F_drag(3:3:end,1:5))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
```

```
%title('Data, wind from $0^\circ$','interpreter','latex','FontSize',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
% 48.4 deg
figure
%surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag(1:3:end,6:10))
surf_wind=v_abs(6:10);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,6:10),F_drag(3:3:end,6:10))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Data, wind from $48.4^\circ$','interpreter','latex','FontSize
    ',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
% 77.9 deg
figure
%surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag(1:3:end,11:15))
surf_wind=v_abs(11:15);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,11:15),F_drag(3:3:end,11:15))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Data, wind from $77.9^\circ$','interpreter','latex','FontSize
    ',16)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=16;
ax.TickLabelInterpreter='latex';

%% 2D plot of kappa_par

figure
plot(Theta_sigma(:,1),kappa_par(1:end,1),'Color',[0 150 0]/255)
hold on
grid on
box on
plot(Theta_sigma(:,2),kappa_par(1:end,2),'Color',[0 200 0]/255)
plot(Theta_sigma(:,3),kappa_par(1:end,3),'Color',[0 255 0]/255)
plot(Theta_sigma(:,4),kappa_par(1:end,4),'Color',[50 255 50]/255)
plot(Theta_sigma(:,5),kappa_par(1:end,5),'Color',[100 255 100]/255)
plot(Theta_sigma(:,6),kappa_par(1:end,6),'Color',[255 0 0]/255)
plot(Theta_sigma(:,7),kappa_par(1:end,7),'Color',[255 40 40]/255)
plot(Theta_sigma(:,8),kappa_par(1:end,8),'Color',[255 80 80]/255)
plot(Theta_sigma(:,9),kappa_par(1:end,9),'Color',[255 120 120]/255)
plot(Theta_sigma(:,10),kappa_par(1:end,10),'Color',[255 160 160]/255)
plot(Theta_sigma(:,11),kappa_par(1:end,11),'Color',[0 0 153]/255)
plot(Theta_sigma(:,12),kappa_par(1:end,12),'Color',[0 0 220]/255)
plot(Theta_sigma(:,13),kappa_par(1:end,13),'Color',[40 40 255]/255)
plot(Theta_sigma(:,14),kappa_par(1:end,14),'Color',[100 100 255]/255)
plot(Theta_sigma(:,15),kappa_par(1:end,15),'Color',[150 150 255]/255)
xlabel('$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$','interpreter','latex',
    'FontSize',20)
ylabel('$\kappa_{\parallel }$ $[\frac{kg}{rad}]$','interpreter','latex'
    ,'FontSize',20)
%title('Green: from $0^\circ$ data, Red: from $48.4^\circ$ data, Blue:
    from $77.9^\circ$ data','interpreter','latex','FontSize',14)
ylim([-21e-7 4e-7])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';


%% 2D plot of kappa_orth_x

figure
hold on
grid on
box on
plot(Theta_sigma(:,6),kappa_orth(1:2:end,1),'Color',[255 0 0]/255)
plot(Theta_sigma(:,7),kappa_orth(1:2:end,2),'Color',[255 40 40]/255)
plot(Theta_sigma(:,8),kappa_orth(1:2:end,3),'Color',[255 80 80]/255)
plot(Theta_sigma(:,9),kappa_orth(1:2:end,4),'Color',[255 120 120]/255)
plot(Theta_sigma(:,10),kappa_orth(1:2:end,5),'Color',[255 160 160]/255)
plot(Theta_sigma(:,11),kappa_orth(1:2:end,6),'Color',[0 0 153]/255)
plot(Theta_sigma(:,12),kappa_orth(1:2:end,7),'Color',[0 0 220]/255)
plot(Theta_sigma(:,13),kappa_orth(1:2:end,8),'Color',[40 40 255]/255)
plot(Theta_sigma(:,14),kappa_orth(1:2:end,9),'Color',[100 100 255]/255)
plot(Theta_sigma(:,15),kappa_orth(1:2:end,10),'Color',[150 150
    255]/255)
xlabel('$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$','interpreter','latex',
    'FontSize',20)
ylabel('Drag coefficient in x direction')
title('Red: 48.4 deg, Blue: 77.9 deg')
ylim([-21e-7 4e-7])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';

%% 2D plot of kappa_orth_y

figure
hold on
grid on
plot(Theta_sigma(:,6),kappa_orth(2:2:end,1),'Color',[255 0 0]/255)
plot(Theta_sigma(:,7),kappa_orth(2:2:end,2),'Color',[255 40 40]/255)
plot(Theta_sigma(:,8),kappa_orth(2:2:end,3),'Color',[255 80 80]/255)
plot(Theta_sigma(:,9),kappa_orth(2:2:end,4),'Color',[255 120 120]/255)
plot(Theta_sigma(:,10),kappa_orth(2:2:end,5),'Color',[255 160 160]/255)
plot(Theta_sigma(:,11),kappa_orth(2:2:end,6),'Color',[0 0 153]/255)
plot(Theta_sigma(:,12),kappa_orth(2:2:end,7),'Color',[0 0 220]/255)
plot(Theta_sigma(:,13),kappa_orth(2:2:end,8),'Color',[40 40 255]/255)
plot(Theta_sigma(:,14),kappa_orth(2:2:end,9),'Color',[100 100 255]/255)
plot(Theta_sigma(:,15),kappa_orth(2:2:end,10),'Color',[150 150
    255]/255)
```

```
xlabel('Theta sigma [rad/s]')
ylabel('Drag coefficient in y direction')
title('Red: 48.4 deg, Blue: 77.9 deg')
ylim([-19e-7 4e-7])

%% 2D plot of kappa_orth_LS
figure
hold on
grid on
box on
plot(Theta_sigma(:,6),kappa_orth_LS(:,1),'Color',[255 0 0]/255)
plot(Theta_sigma(:,7),kappa_orth_LS(:,2),'Color',[255 40 40]/255)
plot(Theta_sigma(:,8),kappa_orth_LS(:,3),'Color',[255 80 80]/255)
plot(Theta_sigma(:,9),kappa_orth_LS(:,4),'Color',[255 120 120]/255)
plot(Theta_sigma(:,10),kappa_orth_LS(:,5),'Color',[255 160 160]/255)
plot(Theta_sigma(:,11),kappa_orth_LS(:,6),'Color',[0 0 153]/255)
plot(Theta_sigma(:,12),kappa_orth_LS(:,7),'Color',[0 0 220]/255)
plot(Theta_sigma(:,13),kappa_orth_LS(:,8),'Color',[40 40 255]/255)
plot(Theta_sigma(:,14),kappa_orth_LS(:,9),'Color',[100 100 255]/255)
plot(Theta_sigma(:,15),kappa_orth_LS(:,10),'Color',[150 150 255]/255)
xlabel('$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$','interpreter','latex',
    'FontSize',20)
ylabel('$\kappa_{\perp }$ $[\frac{kg}{rad}]$','interpreter','latex','
    FontSize',20)
%title('Red: 48.4 deg, Blue: 77.9 deg')
ylim([-21e-7 4e-7])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';


%% 2D plot of kappa_orth_LS_allWinds
figure
hold on
grid on
plot(input_vec,kappa_orth_LS_allWinds(:,1),'Color',[255 0 0]/255)
plot(input_vec,kappa_orth_LS_allWinds(:,2),'Color',[0 0 255]/255)
xlabel('Motor input command [-]')
ylabel('Drag coefficient in x and y direction')
title('Red: 48.4 deg, Blue: 77.9 deg')
ylim([-19e-7 4e-7])


%% 2D plot of kappa_orth_LS_all
figure
hold on
grid on
plot(input_vec,kappa_orth_LS_all(1)*ones(length(input_vec),1),'Color'
    ,[255 0 0]/255)
plot(input_vec,kappa_orth_LS_all(2)*ones(length(input_vec),1),'Color'
    ,[0 0 255]/255)
xlabel('Motor input command [-]')
ylabel('Drag coefficient in x and y direction')
title('Red: 48.4 deg, Blue: 77.9 deg')
ylim([-19e-7 4e-7])


%% Surface plot of calculated F_drag in x direction - new model

% Absolute wind velocity vs. Theta_sigma
figure
labelx='$|v|$ $[\frac{m}{s}]$';
```

```
labely='$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$';
labelz='$F_x [N]$';
% 0 deg
surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(1:3:end,1:5))
xlabel(labelx,'interpreter','latex','FontSize',14)
ylabel(labely,'interpreter','latex','FontSize',14)
zlabel(labelz,'interpreter','latex','FontSize',14)
title('Extended model, wind from $0^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)
% 48.4 deg
figure
surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(1:3:end,6:10))
xlabel(labelx,'interpreter','latex','FontSize',14)
ylabel(labely,'interpreter','latex','FontSize',14)
zlabel(labelz,'interpreter','latex','FontSize',14)
title('Extended model, wind from $48.4^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)
% 77.9 deg
figure
surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(1:3:end,11:15))
xlabel(labelx,'interpreter','latex','FontSize',14)
ylabel(labely,'interpreter','latex','FontSize',14)
zlabel(labelz,'interpreter','latex','FontSize',14)
title('Extended model, wind from $77.9^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)

%% Surface plot of calculated F_drag in y direction - new model

% Absolute wind velocity vs. Theta_sigma
figure
labelx='Absolute value of wind velocity [m/s]';
labely='Theta sigma [rad/s]';
labelz='Calculated drag force in y direction [N]';
% 0 deg
surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(2:3:end,1:5))
xlabel(labelx)
ylabel(labely)
zlabel(labelz)
title('Calculated from 0 deg measurements')
view(150,40)
% 48.4 deg
figure
surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(2:3:end,6:10))
xlabel(labelx)
ylabel(labely)
zlabel(labelz)
title('Calculated from 48.4 deg measurements')
view(150,40)
% 77.9 deg
figure
surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(2:3:end,11:15))
xlabel(labelx)
ylabel(labely)
```

```
zlabel(labelz)
title('Calculated from 77.9 deg measurements')
view(150,40)

%% Surface plot of calculated F_drag in z direction - new model

min_v=min(v_abs);
max_v=max(v_abs);
min_theta=min(min(Theta_sigma));
max_theta=max(max(Theta_sigma));
min_F=min(min(F_drag));
max_F=max(max(F_drag));

% Absolute wind velocity vs. Theta_sigma
figure
labelx='$|v|$ $[\frac{m}{s}]$';
labely='$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$';
labelz='$f_\mathrm{a,calc,z} [N]$';
size_font_labels=20;
% 0 deg
%surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(3:3:end,1:5))
colormap bone
surf_wind=v_abs(1:5);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,1:5),F_drag_calc(3:3:end,1:5))
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Extended model, wind from $0^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
% 48.4 deg
figure
%surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(3:3:end,6:10))
surf_wind=v_abs(6:10);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,6:10),F_drag_calc(3:3:end,6:10))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Extended model, wind from $48.4^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';
% 77.9 deg
figure
%surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc(3:3:end,11:15))
```

```
surf_wind=v_abs(11:15);
surf([surf_wind;surf_wind;surf_wind;surf_wind;surf_wind;surf_wind],
    Theta_sigma(:,11:15),F_drag_calc(3:3:end,11:15))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',size_font_labels)
ylabel(labely,'interpreter','latex','FontSize',size_font_labels)
zlabel(labelz,'interpreter','latex','FontSize',size_font_labels)
%title('Extended model, wind from $77.9^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)
xlim([min_v max_v])
ylim([min_theta max_theta])
zlim([min_F max_F])
ax=gca;
ax.FontSize=18;
ax.TickLabelInterpreter='latex';

%% Plot new drag coefficients that are equivalent to the ones of old
    model in same plots as old model

% z
figure
plot(Theta_sigma(:,1),kappa_par(1:end,1),'y')
hold on
grid on
plot(Theta_sigma(:,2),kappa_par(1:end,2),'y')
plot(Theta_sigma(:,3),kappa_par(1:end,3),'y')
plot(Theta_sigma(:,4),kappa_par(1:end,4),'y')
plot(Theta_sigma(:,5),kappa_par(1:end,5),'y')
plot(Theta_sigma(:,6),kappa_par(1:end,6),'m')
plot(Theta_sigma(:,7),kappa_par(1:end,7),'m')
plot(Theta_sigma(:,8),kappa_par(1:end,8),'m')
plot(Theta_sigma(:,9),kappa_par(1:end,9),'m')
plot(Theta_sigma(:,10),kappa_par(1:end,10),'m')
plot(Theta_sigma(:,11),kappa_par(1:end,11),'c')
plot(Theta_sigma(:,12),kappa_par(1:end,12),'c')
plot(Theta_sigma(:,13),kappa_par(1:end,13),'c')
plot(Theta_sigma(:,14),kappa_par(1:end,14),'c')
plot(Theta_sigma(:,15),kappa_par(1:end,15),'c')
plot(Theta_sigma(:,1),K(2)*ones(length(input_vec),1),'Color',[0 155
    0]/255,'LineWidth',3)
xlabel('Theta sigma [rad/s]')
ylabel('Drag coefficient in z direction')
title('yellow: 0 deg, magenta: 48.4 deg, cyan: 77.9 deg, green: new
    model fit')
ylim([-21e-7 4e-7])

% x,y
figure
hold on
grid on
plot(Theta_sigma(:,6),kappa_orth_LS(:,1),'Color',[255 0 0]/255)
plot(Theta_sigma(:,7),kappa_orth_LS(:,2),'Color',[255 40 40]/255)
plot(Theta_sigma(:,8),kappa_orth_LS(:,3),'Color',[255 80 80]/255)
plot(Theta_sigma(:,9),kappa_orth_LS(:,4),'Color',[255 120 120]/255)
plot(Theta_sigma(:,10),kappa_orth_LS(:,5),'Color',[255 160 160]/255)
plot(Theta_sigma(:,11),kappa_orth_LS(:,6),'Color',[0 0 153]/255)
plot(Theta_sigma(:,12),kappa_orth_LS(:,7),'Color',[0 0 220]/255)
plot(Theta_sigma(:,13),kappa_orth_LS(:,8),'Color',[40 40 255]/255)
plot(Theta_sigma(:,14),kappa_orth_LS(:,9),'Color',[100 100 255]/255)
plot(Theta_sigma(:,15),kappa_orth_LS(:,10),'Color',[150 150 255]/255)
plot(Theta_sigma(:,6),K(1)*ones(length(input_vec),1),'Color',[0 155
    0]/255,'LineWidth',3)
xlabel('Theta sigma [rad/s]')
ylabel('Drag coefficient in x and y direction')
title('Red: 48.4 deg, Blue: 77.9 deg, Green: new model fit')
ylim([-19e-7 4e-7])

%% Surface plot of calculated F_drag in x direction - simple model

% Absolute wind velocity vs. Theta_sigma
figure
labelx='Absolute value of wind velocity [m/s]';
labely='Theta sigma [rad/s]';
labelz='Calculated drag force in x direction (simple model) [N]';
% 0 deg
surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(1:3:end,1:5))
xlabel(labelx)
ylabel(labely)
zlabel(labelz)
title('Calculated from 0 deg measurements')
view(150,40)
% 48.4 deg
figure
surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(1:3:end,6:10))
xlabel(labelx)
ylabel(labely)
zlabel(labelz)
title('Calculated from 48.4 deg measurements')
view(150,40)
% 77.9 deg
figure
surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(1:3:end,11:15))
xlabel(labelx)
ylabel(labely)
zlabel(labelz)
title('Calculated from 77.9 deg measurements')
view(150,40)

%% Surface plot of calculated F_drag in y direction - simple model

% Absolute wind velocity vs. Theta_sigma
figure
labelx='Absolute value of wind velocity [m/s]';
labely='Theta sigma [rad/s]';
labelz='Calculated drag force in y direction (simple model) [N]';
% 0 deg
surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(2:3:end,1:5))
xlabel(labelx)
ylabel(labely)
zlabel(labelz)
title('Calculated from 0 deg measurements')
view(150,40)
% 48.4 deg
figure
surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(2:3:end,6:10))
xlabel(labelx)
ylabel(labely)
```

```
zlabel(labelz)
title('Calculated from 48.4 deg measurements')
view(150,40)
% 77.9 deg
figure
surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(2:3:end,11:15))
xlabel(labelx)
ylabel(labely)
zlabel(labelz)
title('Calculated from 77.9 deg measurements')
view(150,40)

%% Surface plot of calculated F_drag in z direction - simple model

% Absolute wind velocity vs. Theta_sigma
figure
labelx='$|v|$ $[\frac{m}{s}]$';
labely='$\dot{\theta}_\Sigma$ $[\frac{rad}{s}]$';
labelz='$F_\mathrm{z} [N]$';
% 0 deg
surf(v_abs(1:5),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(3:3:end,1:5))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',14)
ylabel(labely,'interpreter','latex','FontSize',14)
zlabel(labelz,'interpreter','latex','FontSize',14)
title('Simple model, wind from $0^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)
zlim([-0.04 0.005])
% 48.4 deg
figure
surf(v_abs(6:10),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(3:3:end,6:10))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',14)
ylabel(labely,'interpreter','latex','FontSize',14)
zlabel(labelz,'interpreter','latex','FontSize',14)
title('Simple model, wind from $48.4^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)
% 77.9 deg
figure
surf(v_abs(11:15),[15400 21900 27800 31600 33400 39300]/2/60*2*pi*4,
    F_drag_calc_simpleModel(3:3:end,11:15))
colormap bone
xlabel(labelx,'interpreter','latex','FontSize',14)
ylabel(labely,'interpreter','latex','FontSize',14)
zlabel(labelz,'interpreter','latex','FontSize',14)
title('Simple model, wind from $77.9^\circ$','interpreter','latex','
    FontSize',14)
view(150,40)
```

```
function [Fx,Fy,Fz] = prepare_data(filename,start_idx,end_idx,
    plot_forces)

counts = 1000000;

data = csvread(filename,1,0);
Fx_raw = data(:,2)/counts;
```

```
Fy_raw = data(:,3)/counts;
Fz_raw = data(:,4)/counts;
% Unit is N now
clear data

% Get rid of bias in the beginning --> all forces apart from thrust and
% drag will be ignored
Fx_raw = Fx_raw - mean(Fx_raw(1:500));
Fy_raw = Fy_raw - mean(Fy_raw(1:500));
Fz_raw = Fz_raw - mean(Fz_raw(1:500));

% Calculate forces
Fx=mean(Fx_raw(start_idx:end_idx));
Fy=mean(Fy_raw(start_idx:end_idx));
Fz=mean(Fz_raw(start_idx:end_idx));
F=[Fx;Fy;Fz];

% Transfer forces to CF body frame
rot_mat = [cos(pi/4) -sin(pi/4) 0;sin(pi/4) cos(pi/4) 0;0 0 1];
F=rot_mat*F;
Fx=F(1);
Fy=F(2);
Fz=F(3);

% Plot forces and start and end idx and means
if plot_forces==1
    figure
    plot(Fx_raw)
    hold on
    plot([start_idx start_idx],[min(Fx_raw) max(Fx_raw)])
    plot([end_idx end_idx],[min(Fx_raw) max(Fx_raw)])
    plot(Fx*ones(length(Fx_raw),1))
    xlabel('idx [-]')
    ylabel('Fx [N]')
    grid on
    figure
    plot(Fy_raw)
    hold on
    plot([start_idx start_idx],[min(Fy_raw) max(Fy_raw)])
    plot([end_idx end_idx],[min(Fy_raw) max(Fy_raw)])
    plot(Fy*ones(length(Fx_raw),1))
    xlabel('idx [-]')
    ylabel('Fy [N]')
    grid on
    figure
    plot(Fz_raw)
    hold on
    plot([start_idx start_idx],[min(Fz_raw) max(Fz_raw)])
    plot([end_idx end_idx],[min(Fz_raw) max(Fz_raw)])
    plot(Fz*ones(length(Fx_raw),1))
    xlabel('idx [-]')
    ylabel('Fz [N]')
    grid on
end


end
```

Bibliography

[1] ATI Industrial Automation. *Network Force/Torque Sensor System Installation and Operation Manual*, 9620-05-net ft edition, February 2013.

[2] Bitcraze. Crazyflie 2.0 product website. `https://www.bitcraze.io/crazyflie-2/`. Accessed: 2015-08-23.

[3] Louis Cattafesta, Chris Bahr, and Jose Mathew. Fundamentals of wind-tunnel design. *Encyclopedia of Aerospace Engineering*, 2010.

[4] Rick Chartrand. Numerical differentiation of noisy, nonsmooth data. *ISRN Applied Mathematics*, 2011, 2011.

[5] Raffaello D'Andrea, Federico Augugliaro, and Michael Hamer. Lecture 11 - system identification. Lecture notes for the lecture "Signals and Systems" by R. D'Andrea, December 2014.

[6] Raffaello D'Andrea, Federico Augugliaro, and Michael Hamer. Lecture 6 - fourier analysis: Applied concepts. Lecture notes for the lecture "Signals and Systems" by R. D'Andrea, October 2014.

[7] Julian Förster. *Crazyflie 2.0 - Firmware Documentation.* IDSC @ ETH Zurich, April 2015.

[8] R. C. Hibbeler and Kai Beng Yap. *Mechanics for Engineer: Dynamics.* Pearson Education South Asia Pte Ltd, thirteenth si edition edition, 2012.

[9] Pijush K Kundu, Ira M Cohen, David R Dowling, P S Ayyaswamy, and H H Hu. *Fluid Mechanics.* Academic Press, 5th edition, 2008.

[10] Sergei Lupashin, Markus Hehn, Mark W Mueller, Angela P Schoellig, Michael Sherback, and Raffaello D'Andrea. A platform for aerial robotics research and demonstration: The flying machine arena. *Mechatronics*, 24(1):41–54, 2014.

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

[12] L Marconi, Claudio Melchiorri, Michael Beetz, Dejan Pangercic, R Siegwart, Stefan Leutenegger, Raffaella Carloni, Stefano Stramigioli, Herman Bruyninckx, Patrick Doherty, et al. The sherpa project: Smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments. In *Safety, Security, and Rescue Robotics (SSRR), 2012 IEEE International Symposium on*, pages 1–4. IEEE, 2012.

[13] Mark W Mueller, Michael Hamer, and Raffaello D'Andrea. Fusing ultra-wideband range measurements with accelerometers and rate gyroscopes for quadrocopter state estimation. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 1730–1736. IEEE, 2015.

[14] Geert Verhoeven, Sergei Lupashin, Christian Briese, and Michael Doneus. Airborne imaging for heritage documentation using the fotokite tethered flying camera. In *EGU General Assembly Conference Abstracts*, volume 16, page 15202, 2014.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
Prof. Dr. R. D'Andrea, Prof. Dr. L. Guzzella

Title of work:

System Identification of the Crazyflie 2.0 Nano Quadrocopter

Thesis type and date:

Bachelor Thesis, August 2015

Supervision:

Michael Hamer
Prof. Dr. Raffaello D'Andrea

Student:

Name:	Julian Förster
E-mail:	fjulian@student.ethz.ch
Legi-Nr.:	12-937-835
Semester:	6

Statement regarding plagiarism:

Zurich, 1. 9. 2015: ____________________