Master's Thesis

Identification and Control Implementation of an AR.Drone 2.0

N.L.M. Jeurgens

DC 2017.013

Coach: dr. ir. A.A.J. Lefeber

Supervisor: prof. dr. H. Nijmeijer

January, 2017

Dynamics and Control
Department of Mechanical Engineering
Eindhoven University of Technology
PO Box 513
5600 MB Eindhoven
The Netherlands

Abstract

In this work, the goal is to implement controlling software that is designed in Matlab-Simulink on a Parrot AR.Drone 2.0. Hereto, we create an extensive model that not only represents the equations of motion regarding multi-rotors, but additionally includes sensor and actuator behaviour. The parameter values, specific to this platform are determined by conducting measurements and analysis of signals. Subsequently, we choose control laws for quad-rotor stabilisation and reconstruct the full state of the system, based on sensor measurements. Using the elements described, we test the performance of the observing and controlling software in simulations, where we make use of the extensive model. Herein, we can tune controller and observer parameters and test the behaviour of control laws under influence of discretisation, quantisation, saturation, and delay. Once satisfactory performance is achieved in simulation, the exact same control software can be embedded on the actual AR.Drone 2.0, where further testing can commence, in order to achieve flight.

Contents

List of symbols

Symbol	Description	Unit
a_{IMU}	Accelerometer sensor signal vector	-
B_{o}	Geo-magnetic field vector	T
B_{cam}	Reconstructed body-fixed magnetic field vector, based on camera measurements	T
B_{IMU}	Magnetometer sensor signal vector	-
c_1, c_2	Orientation control tuning parameters	-
C_{D}	Drag coefficient matrix	-
C_{ext}	External camera signal vector	-
$e_\rho,\ e_v$	Position and velocity errors	m, m/s
$f,\ f_{\text{r}}$	Actual and reference thrust forces	N
$\hat{f},\ \hat{f}_{\text{d}}$	Thrust and desired thrust vectors	N
F_{D}	Generalised translational damping force vector	N
g	Gravitational acceleration	m/s^2
H	Sensor matrix	-
J	Inertia matrix	kg m^2
J_{r}	Rotor inertia around rotation axis	kg m^2
$k_{\text{w}}, k_{\text{z}}, k_1, k_2$	Position control tuning parameters	-
l_1, l_2, l_3	Position and velocity observer tuning parameters	-
m	Mass	kg
P_{W}	Pulse-width motor signal	%
p, q, r	Angular velocities around the principle body axes	rad/s
Q	Process noise covariance matrix	-
$R,\ R_{\text{r}}$	Drone's actual and reference rotation matrices	-
R_{Acc}	Accelerometer noise covariance matrix	-
R_e	Orientation error matrix	-
R_g	Gyroscope noise covariance matrix	-
$S(.)$	Skew-symmetric matrix	-
t	Time	s
T_{D}	Desired trajectory identifier	-
$u,\ u_{\text{d}}$	Actual and desired position control input	N/kg
u, v, w	Body-fixed velocities	m/s
V_{batt}	Battery potential	V
$W_s(t, f_s, w)$	Square wave function	-
x, y, z	Drone position	m
$Y,\ C_{\text{b}},\ C_{\text{r}}$	Camera luminance and chrominance matrices	-
z_{us}	Altimeter sensor signal	-

Symbol	Description	Unit
δ	Time-delay	s
ω, ω_r	Drone's actual and reference angluar velocity vectors	rad/s
ω_e	Rotational velocity error vector	rad/s
ω_IMU	Gyroscope sensor signal vector	-
ω_r	Rotor angular velocity vector	rad/s
ϕ, θ, ψ	Roll, pitch, and yaw angles	rad
$\hat{\phi}, \hat{\theta}, \hat{\psi}$	Estimated roll, pitch, and yaw angles	rad
ψ_cam	Yaw angle, calculated from camera measurements	rad
ρ, ν	Drone's position and body-fixed velocity vectors	m, m/s
ρ_Air	Air density	kg/m^3
σ	Standard deviation	-
$\sigma_n(.)$	Smooth saturation function	-
τ, τ_r	Drone's torque and reference torque vectors	N m
$\mathcal{B}$	Body-fixed coordinate frame	-
$\mathcal{B}'$	Body-centred coordinate frame with the same orientation as $\mathcal{I}$	-
$\mathcal{D}$	Desired heading coordinate frame	-
$\mathcal{I}$	Eart-fixed coordinate frame	-
$\mathcal{N}(t, \sigma)$	White noise with standard deviation σ	-
$\mathcal{R}$	Reference coordinate system	-

Chapter 1

Introduction

1.1 Background and motivation

Drones are becoming more readily available and less expensive. Also, due to technological advancement, microprocessors can be reduced in size and price, while still providing sufficient processing power to execute significant tasks. Due to this evolution, smaller and lighter autonomous or remote controlled aircraft can be developed, such as multi-rotors. The applications for these craft seem endless; they can be used for geo-archaeology (Oczipka et al., 2009), assisting rescue operations regarding natural disasters (Apvrille, Tanzi, & Dugelay, 2014), monitoring crops in agriculture (Tripicchio, Satler, Dabisias, Ruffaldi, & Avizzano, 2015), surveillance missions against poaching in Africa, and many other applications.

This thesis is part of a larger research project, which has as final goal to use drones in sound imaging technology, further expanding the versatility of drone applications. Sound imaging technology uses an array of microphones in a grid formation to record sound waves that, using methods developed by Scholte (2008), can be visualised. These sound cameras can be utilised for identification of acoustic noise sources, not only spatial locations, but also magnitude and frequency of these sources can be identified. This technology can be used in themes ranging from the automotive industry, where engine noise can be analysed, to the design of home appliances, where noise sources can be analysed and used for design improvements. Other applications where sound imaging can be useful are in buildings, healthcare, the offshore industry, and many other industrial applications, such as production machinery (Sorama, 2015). An example sound image using this technology is given in Figure 1.1.

Figure 1.1: Sound image of the engine compartment of a Volkswagen Caddy diesel. Source: Sorama (2015).

Using drones in sound imaging technology expands the possibilities even further. Applications can be expanded to image the sound production of entire buildings, factories, offshore drilling platforms, and

possibly even larger applications. In order to be able to achieve similar results using drones, these need to be able to fly in a grid array formation, maintaining accurate position. The first steps to achieve this final goal are choosing a platform and design a controller that is able to position a single drone. Later, the connection and communication between several drones can be made and, lastly, the microphones need to be added and (audio) noise, created by the flying drones, needs to be filtered out. The remaining audio data then can be used to visualise sound from sources below the grid of flying drones.

In this report we focus on the identification and control implementation of a Parrot AR.Drone 2.0. Literature has shown promising results regarding linear and non-linear control laws for stabilisation and reference tracking of multi-rotors. The steps involving the integration of theoretically designed control laws in an actual drone, however, are rarely discussed. In this work, we start by constructing an extensive model that not only represent the equations of motion regarding multi-rotors, but also includes sensor and actuator behaviour. The parameters, specific to the AR.Drone 2.0, are determined by conducting measurements and signal analysis such that the extensive model ultimately is used to test the control law that is to be implemented and for tuning control parameters. During simulations control and observer parameters can be tuned such that stable (and possibly robust) behaviour is achieved. Additionally, the effects of signal saturations, quantisations, delay, and discretisation can be analysed in simulations. Once satisfactory performance is reached in these simulations, the controller can be embedded on the actual AR.Drone 2.0, where further tuning can commence. Note that, while the prime objective of the larger research project is to use drones in conjunction with sound imaging technology, the results of this thesis can be used in many other AR.Drone 2.0 related applications.

1.2 OBJECTIVES

As stated, the final goal of this project is to use sound imaging technology in conjunction with drones. In previous parts of this project, a platform (the AR.Drone 2.0) was chosen for further work. Subsequently, Jeurgens (2016) established a connection between this specific platform and Simulink, based on the work of Daranlee and Slovak194 (n.d.), such that a Simulink model can be used to import sensor signals and control the motors. Thereafter, Jeurgens (2016) started system identification, but left room for improvement.

Resuming the project, the prime objective for this thesis is to autonomously fly a single Parrot AR.Drone 2.0 with the use of in Simulink designed control software. To achieve this objective, several sub-objectives are defined below:

1. Create an elaborate model that includes sensor and actuator behaviour, such that control software can be tested in simulations, prior to actual implementation.
2. Continue system estimation such that all model parameters are known.
3. Choose control laws that can be used to stabilise the quad-rotor
4. Reconstruct the full state of the quad-rotor based on sensor measurements during flight.
5. Test the observer and controller software on the created elaborate model in simulations and find viable values for control and observer parameters.
6. Embed the observing and controlling software in the drone and tune control and observer parameters for stable flight.

Whereas the objective of the entire research project is to use sound imaging technology in conjunction with drones, the objectives and results of this thesis are applicable for any other drone-related application.

1.3 Report outline

This report is comprised of several chapters, followed by several appendices. Additionally, a DVD is provided, containing a Matlab tool set in addition to several film clips. The composition of this thesis is described below:

Chapter 2: Modelling and identification of a quad-rotor
We discuss the mathematical model that describe the motions of a quad rotor. This model is extended with sensor and actuator models to more accurately resemble the behaviour of an actual quad-rotor in flight. The main purpose of this extended model is not primarily to design a controller, however, but for simulation purposes, such that designed controllers can be tested on said model prior to implementation.

Chapter 3: Controllers
We discuss several controllers, aiming to stabilise the quad-rotor and enabling it to track a predefined trajectory. We start by discussing the global control structure and the roles of each component. Then, we discuss a position controller in Section 3.1 and orientation controller in Section 3.2, both designed by Van den Eijnden (2017). Then, we shortly discuss how a simple supervisory controller is used for proper and safe drone operation. Lastly, in Section 3.4, we discuss how the desired force and torques are converted to PWM signals that the quad-rotor can use to control the motors.

Chapter 4: State reconstruction from sensor data
We discuss methods to reconstruct the quad-rotor's states from sensor measurements. We start by determination of the quad-rotor's orientation, based on two vectors that are available from the sensors. Then, in Section 4.2, we discuss a method for position measurements using the on-board camera. Subsequently, as sensor measurements are subject to noise, in Section 4.3 we discuss a two-linear Kalman filter observer design that fuses the signals of several sensors into the two vectors required to calculate the orientation, improving accuracy and reducing noise. Lastly, as only the position can be measured and the controller additionally requires a velocity measurement, we discuss a position and velocity observer (Van den Eijnden (2017)) that estimates and filters the position and velocity of the quad-rotor.

Chapter 5: Implementation
The proposed controlling software, discussed in Chapter 3, and the state reconstruction, discussed in Chapter 4, are subjected to the quad-rotor model of Chapter 2 during simulations. Subsequently, the control-law is embedded in the actual AR.Drone 2.0 for real-world testing. We start by giving the Simulink structures, both for simulations and implementation. Then, in Section 5.2, we discuss the results of the simulations where we aim to track a three dimensional ellipsoid trajectory. Subsequently, in Section 5.3, we discuss the alterations in the Matlab tool-set that were made in this work, extending the capabilities compared to the previous iteration by Jeurgens (2016). Then, in Section 5.4, we present the results from experiments, where the controller is embedded in the drone. First, we aim to track a simple hovering trajectory to verify the stability of the embedded controller, whereas subsequently the tracking performance in case of the three dimensional ellipsoid trajectory is discussed. Lastly, we discuss non-modelled dynamics that may influence the quad-rotor's behaviour in Section 5.5.

Chapter 6: Conclusions and recommendations
In this chapter we present the conclusions of this thesis and provide several recommendations for further research.

These chapters are followed by several appendices, of which Appendix A contains specific hardware information and the derivation of the sensor models discussed in Chapter 2. Appendix C contains the

measurement plans that were used to determine the system parameters and Appendix D contains several manuals for using and installing the AR.Drone 2.0 toolbox. In addition to this report, this work also includes a DVD that contains the Matlab tools that were used during experiments and measurements. This set of tools enables the user to connect to the AR.Drone 2.0 and compile and run control software designed in Simulink. Additionally, the DVD also includes several film clips of the AR.Drone 2.0 during experiments and a 20 second clip that shows the essence of this thesis.

Chapter 2

Modelling and identification of a quad-rotor

In this chapter we discuss the model of a quad-rotor, starting with a general model of a translating and rotating body subject to torques and forces, representing the AR.Drone 2.0's body. As the states of the quad-rotor cannot be measured directly, only through sensors, we subsequently give models that simulate the sensory signals as a result of the quad-rotor's state. Finally, as the controlling forces and torques are generated by motors, we give a model that simulates the motor-rotor combinations. Using all these sub-models, a model can be built that has PWM inputs and sensor outputs, similar to the real AR.Drone 2.0. The prime goal of this elaborate model is to use in simulations, such that a designed control law can be tested virtually, prior to implementation. A schematic representation of the model structure is given in Figure 2.1.

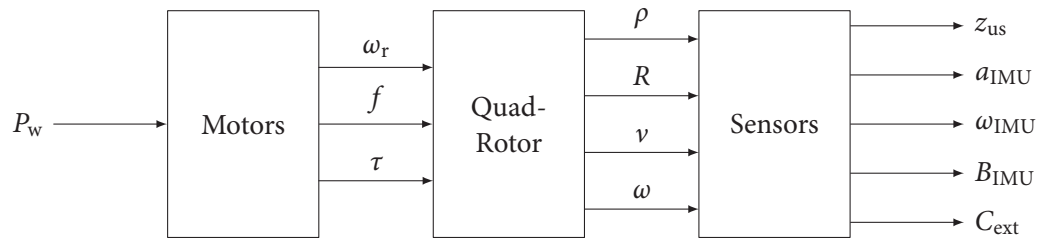

Figure 2.1: Schematic representation of the quad-rotor model structure.

2.1 Multi-rotor model

The Parrot AR.Drone 2.0 is a quad-rotor in ×-configuration. This means that the rotors are not aligned on the principle axes of the body-fixed coordinate system, as is the case in a +-configuration. A quad-rotor is a rotating and translating body with four actuators, each providing a force in the body-fixed z-direction and a torque to the body. A schematic representation is given in Figure 2.2. We consider two coordinate frames $\mathcal{I}$ and $\mathcal{B}$; the inertial, earth-fixed, frame and the body-fixed frame. Van den Eijnden (2017) has derived a model for the quad-rotor's behaviour in general terms, which we have extended with terms that include rotor inertia (as by Choi and Ahn (2015)) and air resistance. The resulting model is:

$$\dot{\rho} = Rv \tag{2.1}$$

$$\dot{R} = RS(\omega) \tag{2.2}$$

$$\dot{v} = \frac{1}{m}\left(B_f f - G - mS(\omega)v + F_D(v)\right) \tag{2.3}$$

$$\dot{\omega} = J^{-1}\left(\tau + S(J\omega)\omega - J_r S(\omega)B_f\Omega\right) \tag{2.4}$$

where $\rho = [x, y, z]^\top \in \mathbb{R}^3$ is the position of the drone with respect to $\mathcal{I}$, $v = [u, v, w]^\top \in \mathbb{R}^3$ are the body-fixed velocities, expressed in $\mathcal{B}$, and $\omega = [p, q, r]^\top \in \mathbb{R}^3$ are the body-fixed angular velocities, expressed

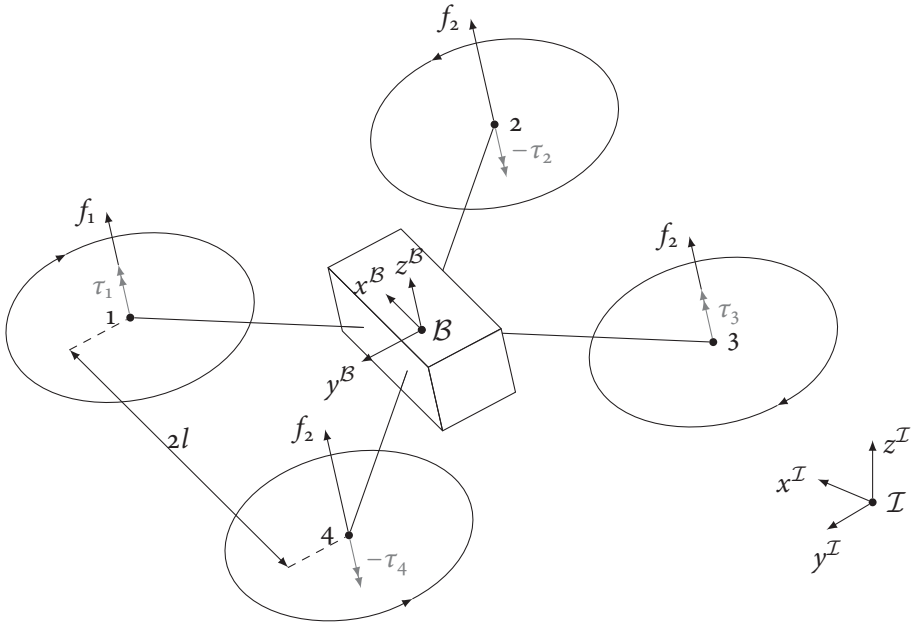

Figure 2.2: Schematic representation of a quad-rotor with inertial and body-fixed coordinate frames $\mathcal{I}$ and $\mathcal{B}$.

in $\mathcal{B}$. The matrix $R \in \mathbb{R}^{3\times3}$ is the rotation matrix that describes the rotation from $\mathcal{B}$ to $\mathcal{I}$. Furthermore,

$$S(\omega) = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}, \qquad G = mR^\top \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}, \qquad B_f = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \text{and} \quad \Omega = \sum_{i=1}^{4} \omega_{\mathrm{r},i}.$$

The drone's mass is given by m, $J = \operatorname{diag}\left(\left[J_x, J_y, J_z\right]\right)$ is the body inertial matrix, J_r is the rotor inertia, $\omega_{\mathrm{r,i}}$ is the rotor angular velocity, and g is the gravitational acceleration. The control force and torque are defined as $f \in \mathbb{R}$ and $\tau = \left[\tau_x, \tau_y, \tau_z\right]^\top \in \mathbb{R}^3$, respectively.

Air friction (drag) is defined as:

$$F_\mathrm{D}(v) = -\frac{1}{2}\rho_\mathrm{Air} C_\mathrm{D} \operatorname{sgn}(v) \circ v \circ v, \qquad C_\mathrm{D} = \begin{bmatrix} c_{\mathrm{D},x} & & \\ & c_{\mathrm{D},y} & \\ & & c_{\mathrm{D},z} \end{bmatrix}, \tag{2.5}$$

where $c_{\mathrm{D},x}$, $c_{\mathrm{D},y}$, and $c_{\mathrm{D},z}$ are the drones drag coefficients in x-, y-, and z- directions, respectively, and ρ_Air is the air density. In (2.5), $\circ$ represents the Hadamard matrix product; an element-wise multiplication of two vectors or matrices. The sgn function also operates element-wise.

Whereas this model describes the rigid-body dynamics of a quad-rotor, the states are measured not directly, but by means of the internal sensory equipment. Also, the control outputs f and τ are not controlled directly. Hence, we extend the quad-rotor model by simulating the sensors and actuators, to give a more complex, yet more accurate model that is more suitable for simulations of the control laws to be implemented.

2.2 Hardware I/O models

As stated before, the states of the AR.Drone 2.0 cannot be measured directly, but we are dependent on the internal sensory equipment. In this section we model the behaviour of the sensors and actuators, listed

below, in order to extend the mathematical model of the quad-rotor discussed in Section 2.1 to a more complete and realistic model, as shown in Figure 2.1.

- ultrasonic altimeter
- 3-axis accelerometer
- 3-axis gyroscope
- 3-axis magnetometer
- external camera
- motors

Measurements have been conducted to model the sensor behaviour under influence of temperature and orientation, for example. The results and additional information on the sensors and actuators are given in Appendix A. For these measurements, the AR.Drone 2.0 toolbox for Matlab was used (Jeurgens, 2016), providing the necessary Simulink blocks to interface with the sensors and actuators. The resulting models for sensor behaviour are presented below.

Altimeter

The ultrasonic altimeter is a down-facing distance sensor that uses ultrasonic pulses to determine the distance to an object. It has a range of 0.275 m to 6 m and a linear relation between distance and sensor signal. The measured altitude, however, depends on the angle α between the body fixed z-axis $\vec{z}^{\mathcal{B}}$ and the z-axis of the inertial frame $\vec{z}^{\mathcal{I}}$. The relation between the actual altitude z and the measured altitude $\hat{z}$ is given by

$$\hat{z} = \frac{z + \zeta^\top R^\top o_{\text{us}}}{\cos\alpha} - a_1(z)(\cos\alpha - 1) - a_2(z)(\cos\alpha - 1)^2, \tag{2.6}$$

where $\zeta = [0, 0, 1]^\top$, o_{us} is the offset of the ultrasonic altimeter with respect to $\mathcal{B}$, and the angle $\alpha = \arccos(\zeta^\top R \zeta)$ is the angle between $z^{\mathcal{B}}$ and $z^{\mathcal{I}}$. The measured altitude can be converted to counts using the following relation:

$$\hat{z}_{\text{us}} = \frac{\min(6, \max[0.275, \hat{z}]) - b_{\text{us}}}{a_{\text{us}}}, \tag{2.7}$$

where a_{us} and b_{us} are the coefficients of a linear relation $\hat{z} = a_{\text{us}}\hat{z}_{us} + b_{\text{us}}$, given in Appendix A.2. In addition to sensor noise, a square pulse signal with a frequency of 25 Hz and a pulse-width of $\frac{1}{16}$ is added to the counts in $\hat{z}_{\text{us}}$. The expression for a square pulse signal as a function of time, frequency, and pulse-width is given by

$$W_{\text{s}}(t, f_{\text{s}}, w) = \frac{1}{2}\left[1 + \operatorname{sgn}\left(\sin\left(2\pi f_{\text{s}}\left(t - \frac{2w+1}{4}\right)\right) - \sin\left((2w+1)\frac{\pi}{2}\right)\right)\right], \tag{2.8}$$

where t indicates time, f_{s} is the frequency in Hz, and $w \in [0, 1]$ is the pulse-width. Then the expression that describes the sensor output of the altimeter is given by

$$z_{\text{us}}|_{400\,\text{Hz}} = \text{round}\left[\left(\hat{z}_{\text{us}} + \mathcal{N}(t, \sigma_{\text{us}})\right)\Big|_{25\,\text{Hz}} + 3.28 \cdot 10^4 W_{\text{s}}\left(t, 400, \frac{1}{16}\right)\right]\Bigg|_{400\,\text{Hz}}, \tag{2.9}$$

where $\mathcal{N}(t, \sigma_{\text{us}}) \in \mathbb{R}$ is measurement noise, modelled as white noise with variance σ^2_{us} and sampled at 25 Hz. The measured altitude $\hat{z}$ is also sampled at 20 Hz, and W_{s} is sampled at 400 Hz. At instances between samples, the previously determined values are held until a new sample arrives.

ACCELEROMETER

The accelerometer measures the body-fixed accelerations of the drone, which can be expressed as:

$$\begin{aligned}\hat{a}_{\text{IMU}} =& R^\top \ddot{\rho} + R^\top \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \\ =& \frac{1}{m} B_f f + F_{\text{D}}(v),\end{aligned} \tag{2.10}$$

assuming that Coriolis effects are not measured by the accelerometer. Converting the body fixed accelerations $\hat{a}_{\text{IMU}}$ to sensor counts is achieved by

$$a_{\text{IMU}}|_{400\,\text{Hz}} = 4\,\text{round}\left[\frac{128\hat{a}_{\text{IMU}}}{g} + \mathcal{N}(t, \sigma_{\text{acc}})\right]\Bigg|_{400\,\text{Hz}}, \tag{2.11}$$

where g is the gravitational acceleration and $\mathcal{N}(t, \sigma_{\text{acc}}) \in \mathbb{R}^3$ is measurement noise, modelled as white noise with standard deviations $\sigma_{\text{acc}} \in \mathbb{R}^3$. In Appendix A.3 we note that the accelerometer signal is temperature dependent. We assume in our sensor model, however, that sensor package temperature is constant and compensated for by the correction as discussed in Appendix A.3.

GYROSCOPE

The gyroscope measures the angular velocities ω of the quad-rotor around its principle axes. Adding noise and converting to sensor counts results in the following expression for the gyroscope model:

$$\omega_{\text{IMU}}|_{400\,\text{Hz}} = \text{round}\left[\omega \frac{180}{\pi} \frac{2^{16}}{4000} + \mathcal{N}(t, \sigma_{\text{gyro}})\right]\Bigg|_{400\,\text{Hz}}, \tag{2.12}$$

where $\mathcal{N}(t, \sigma_{\text{gyro}}) \in \mathbb{R}^3$ is measurement noise, modelled as white noise with standard deviations $\sigma_{\text{gyro}} \in \mathbb{R}^3$. Similarly to the accelerometer, the gyroscope signal depends on the sensor package temperature. In this model, we assume constant temperature, which is corrected for as discussed in Appendix A.4.

MAGNETOMETER

The magnetometer measures the magnetic field in which the drone is placed (in most cases this is the geo-magnetic field) along its body fixed axes. For this model we assume the geo-magnetic field is measured and that it is constant for the period of simulation. We assume that the only affecting factor in the magnetometer measurement is the drone's orientation.

$$B_{\text{IMU}} = \text{round}\left[R^\top B_{\text{mat}} \frac{\sqrt{1.35 \cdot 10^4}}{\|B_{\text{mat}}\|_2} + \mathcal{N}(t, \sigma_{\text{mag}})\right]\Bigg|_{400\,\text{Hz}}, \tag{2.13}$$

where $\mathcal{N}(t, \sigma_{\text{mag}}) \in \mathbb{R}^3$ is measurement noise, modelled as white noise with standard deviations $\sigma_{\text{mag}} \in \mathbb{R}^3$. The vector B_{mat} is the geomagnetic field vector as calculated by Matlab, using the function `wrldmagm.m(.)`, and is converted from a NED (north-east-down) to a NWU (north-west-up) coordinate system. Note that the magnitude of B_{IMU} is not critical as only orientation components can be derived from this sensor, hence only the direction is of significance.

External camera

The external camera that is used in experiments is located above the test facility, facing downwards. This camera has a full HD resolution of 1920×1080 px and covers an area of approximately 11×7.5 m (Aert, 2016). Aert (2016) has developed a Matlab script that loads the images from this camera and extracts the drone's location in x- and y-directions, and the heading ψ using an LED strip identifier, mounted to the drone. The measured values are sent to the drone via a UDP connection. In order to simulate this behaviour, we first convert the drone's position to pixels, then round the result to simulate quantisation effects and, finally, convert back to metres in order to simulate the position. For the camera we assume very low noise levels, such that these can be neglected, and a sample rate of 10 Hz. We define:

$$x_{\text{ext}} = \frac{7.5}{1080}\,\text{round}\left[\frac{1080}{7.5}x\right] \tag{2.14}$$

$$y_{\text{ext}} = \frac{11}{1920}\,\text{round}\left[\frac{1920}{11}y\right] \tag{2.15}$$

$$\psi_{\text{ext}} = \text{atan}_2\left(\chi_2,\, \chi_1\right), \tag{2.16}$$

with $\chi = \begin{bmatrix}\chi_1\\ \chi_2\\ \chi_3\end{bmatrix} = \text{round}\left(\begin{bmatrix}\frac{1080}{7.5} & & \\ & \frac{1920}{11} & \\ & & 0\end{bmatrix} R \begin{bmatrix}0.3\\ 0\\ 0\end{bmatrix}\right)$.

The UDP input of the external camera which contains the xy-position of the drone and its heading is modeled as:

$$C_{\text{ext}}\big|_{10\,\text{Hz}} = \left.\begin{bmatrix} x_{\text{ext}}(t-\delta)\\ y_{\text{ext}}(t-\delta)\\ \psi_{\text{ext}}(t-\delta)\\ 0\end{bmatrix}\right|_{10\,\text{Hz}}, \tag{2.17}$$

where $\delta > 0$ denotes a constant time delay that is introduced by the time required for image processing and signal transmission.

Motor model

The motors each receive a PWM signal $P_{\text{W},i}$ that is converted to an angular velocity of the rotors, resulting in a thrust force and a drag torque. These forces and torques directly act upon the drone according to the following relation

$$\begin{bmatrix} f\\ \tau_x\\ \tau_y\\ \tau_z\end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1\\ l & -l & -l & l\\ -l & -l & l & l\\ 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix} f_1\\ f_2\\ f_3\\ f_4\end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 0\\ 1 & -1 & 1 & -1\end{bmatrix}\begin{bmatrix} \tau_1\\ \tau_2\\ \tau_3\\ \tau_4\end{bmatrix}. \tag{2.18}$$

The rotor angular velocity is given by

$$\omega_{\text{r}} = \text{diag}\left(A_{\text{r},\omega}\right) P_{\text{W}} + B_{\text{r},\omega}, \tag{2.19}$$

where $\omega_{\text{r}} = \left[\omega_{\text{r},1}, \omega_{\text{r},2}, \omega_{\text{r},3}, \omega_{\text{r},4}\right]^{\top} \in \mathbb{R}^4$ contains each rotor angular velocity, $A_{\text{r},\omega},\ B_{\text{r},\omega} \in \mathbb{R}^4$ hold coefficients for the linear relation between the rotor velocity and PWM signal, and $P_{\text{W}} = \left[P_{\text{W},1}, P_{\text{W},2}, P_{\text{W},3}, P_{\text{W},4}\right]^{\top} \in \mathbb{R}^4$ are the PWM signals going to each motor. The force and drag generated by each rotor is quadratically related to the PWM signals as follows:

$$f_{\text{r}} = \text{diag}\left(A_{\text{r},f}\right) P_{\text{W}} \circ P_{\text{W}} + \text{diag}\left(B_{\text{r},f}\right) P_{\text{W}} + C_{\text{r},f} \tag{2.20}$$

$$\tau_{\text{r}} = \text{diag}\left(A_{\text{r},\tau}\right) P_{\text{W}} \circ P_{\text{W}} + \text{diag}\left(B_{\text{r},\tau}\right) P_{\text{W}} + C_{\text{r},\tau}, \tag{2.21}$$

where $A_{r,f}$, $B_{r,f}$, $C_{r,f} \in \mathbb{R}^4$ and $A_{r,\tau}$, $B_{r,\tau}$, $C_{r,\tau} \in \mathbb{R}^4$ contain the coefficients for quadratic relations between PWM signals and thrust and torque, respectively. The coefficient values for these parameters are given in Appendix A.7. The methods for determining all model parameters are given in the following section.

2.3 Identification of system parameters

The model discussed in this chapter consists of several sub-models, which we can separate in three categories: sensors, dynamics, and actuators. In this section, we discuss how the parameters values and sub-models have been obtained for the AR.Drone 2.0. Let this section be a guide in how system parameters can be obtained for other quad-rotors, such that, once a Simulink connection is established, new control software can be implemented.

The first step of determination of all model parameters are the sub-models of the sensors on-board the quad-rotor, as some sensors are required to measure model parameters of the dynamics. Subsequently, we discuss how the model parameters of the dynamics subsystem can be determined. And finally, we discuss the actuators.

2.3.1 Sensor model parameters

The parameter estimation for the sensor models we discuss are the altimeter, accelerometer, gyroscope and magnetometer. The model parameters of the external top camera were obtained from Aert (2016).

By examining the output of the sensor block in Simulink, we notice that both the accelerometer and the gyroscope not only output each their three specific sensor signals, but also each include a temperature signal. This intuitively suggests a sensitivity to temperature. As both sensors are used for the determination of the quad-rotor's orientation, we desire to compensate for temperature induced biases, which could heavily influence the orientation estimation. Hence, we conducted sensor measurements while varying the ambient temperature (Appendix C.2) in order to relate the sensor values to temperature and compensate if necessary.

From the Parrot AR.Drone 2.0 documentation we learn that the gyroscope has a range of $[-2000,\ 2000]^\circ$/s, which is a total span of 4000 $^\circ$/s. In Simulink, the gyroscope signals are unsigned 16 bit integers, ranging $[0, 2^{16} - 1] \in \mathbb{N}$. Using both properties, we can derive relation (2.12), excluding noise.

The sensitivity of the accelerometer can be determined by experimentation. First, measure the sensor output when the quad-rotor is level and subsequently rotated upside-down. The difference in sensor signal ($\approx$1024 counts) coincides with body-fixed accelerations of ~2 g. We notice, however, that the sensor values are not defined in $\mathbb{N}$, but in $4\mathbb{N}$. Hence, sensor values only change by steps of four counts. These conclusions lead to relation (2.11) for the accelerometer behaviour.

The magnetometer measures the magnetic field in which the drone is located (geo-magnetic field), which we can simulate by rotation of a model of the geo-magnetic field vector, provided by Matlab. Conversion to sensor counts, however, can be achieved by following the measurements as described in Appendix A.5. The fourth element of β in (A.14) represents the squared radius of the sphere of measurement points. Hence, we can derive relation (2.13) for the magnetometer sensor behaviour, with $\beta_4 = 1.35 \cdot 10^4$.

The altitude of the drone can be measured using the ultrasonic altimeter. In order to relate the sensor counts to the true altitude in metres, measurements were conducted (Appendix C.5). Herein the drone

was held at a specific, measured, altitude while level, such that the altimeter was aimed perpendicular to the floor. We found a linear relation between sensor counts and altitude, with a lower bound of 0.275 m. In addition to the altitude, the altimeter is sensitive to orientation, specifically; the angle between $z^{\mathcal{B}}$ and the normal vector of the surface, $z^{\mathcal{I}}$, to be measured. Hence, additional measurements were conducted (Appendix C.6) to formulate a relation between the drone's attitude and sensor measurement. Using both relations, we can construct relation (2.9) for the altimeter sensor.

Finally, the sensor noise levels can be determined by a short sensor readout while the drone is stationary and analysing the signals in Matlab. Ideally, all sensor values should be constant as nothing changes in the drone's position and orientation, however, in reality all sensors are subject to noise. Using the Matlab command `var(.)`, outputs the variance σ^2 of the input signal.

Using the steps described in this sub-section, we realised the sensor sub-models described in Section 2.2 and found the model parameters. In the next subsection, we discuss how the model parameters of the dynamics sub-model can be determined.

2.3.2 Dynamics model parameters

In this subsection we discuss the identification of model parameters of the dynamics sub-model. Herein we explain how the mass, m, and mass moment of inertia matrix, J, were determined. We do not discuss the determination of the rotor mass moment of inertia, J_r, as this value was determined by Q. Li (2014). Also, as we assume relatively slow translational velocities for the main part of this report, we assume that air induced drag can be neglected and $C_{\mathrm{D}} = 0^{3\times3}$.

The mass of the AR.Drone 2.0 was determined simply by weighing it on a scale in two scenarios: equipped with the indoor hull and equipped with the outdoor hull. The mass moment of inertia was determined by the bifilar pendulum experiments described in Appendix C.3. The drone was suspended by two parallel wires and given an initial rotation prior to release, while the angular velocity was measured by the internal gyroscopes. The advantage of using the internal sensory equipment is that no external sensors that interfere with the bifilar pendulum dynamics are required. The angular velocity of a bifilar pendulum model was fitted to the actual measurement data. Using an optimisation script in Matlab, the minimum error between simulation and measurement could be found, leading to an accurate estimation of the mass moment of inertia.

2.3.3 Actuator model parameters

The only actuators available on the AR.Drone 2.0, are the four rotors, each providing a thrust force and a drag torque to the rotor's body as a result of angular velocity. Controlling the rotors, we can use four PWM signals going to the motor block in Simulink, receiving values ranging from 0 to 100. Measurements have been conducted to find the relations describing the motors' angular velocities $\omega_{\mathrm{r},i}$, thrusts $f_{\mathrm{r},i}$, and drag torques $\tau_{\mathrm{r},i}$ (Appendix C.7). In these measurements, the drone was mounted to a linear load cell and angular load cell, measuring thrust and torque, respectively. An optical tachometer was used to determine the angular velocity. Sweeps through PWM values have been made in order to relate the PWM value to the measured angular velocity, thrust force, and drag torque. The results can be found in Appendix A.7.

2.4 Summary

In this chapter we have discussed several sub-models in order to build the quad-rotor model depicted in Figure 2.1. Firstly in Section 2.1, we discussed the model of translating and rotating body in 3D space,

representing a quad-rotor's body. Subsequently in Section 2.2, since the quad-rotor's state cannot be determined direcly, only through use of sensors, we discussed the models that simulate the sensor signals under influence of the system states. Hence, we now have constructed a model that receives four PWM signals and outputs simulated sensor signals for the altimeter, accelerometer, gyroscope, magnetometer, and the external camera. These are the inputs and outputs that are available on the actual AR.Drone 2.0 in our test facility.

In the next chapter we use the quad-rotor model of Section 2.1 to build a cascaded controller for the AR.Drone 2.0 which enables the drone to follow a trajectory. The cascaded control system consists of an inner-loop, controlling the drone's attitude, and an outer-loop, controlling the position by generating a desired orientation and force. Later, in Chapter 4, we use the models discussed in Section 2.2 to reconstruct the state of the drone, based on the sensor signals. This is done by means of two observers: A Kalman-filter based orientation observer and a non-linear position and velocity estimator.

Chapter 3

Controllers

In the previous chapter we discussed a model of a quad-rotor, comprised of several sub-models. In this chapter we use the quad-rotor model of Section 2.1 to build a cascaded controller for the AR.Drone 2.0 which enables the drone to follow a trajectory. A schematic view of the control system is given in Figure 3.1. The control software is comprised of several sub-systems, each with their own purpose.

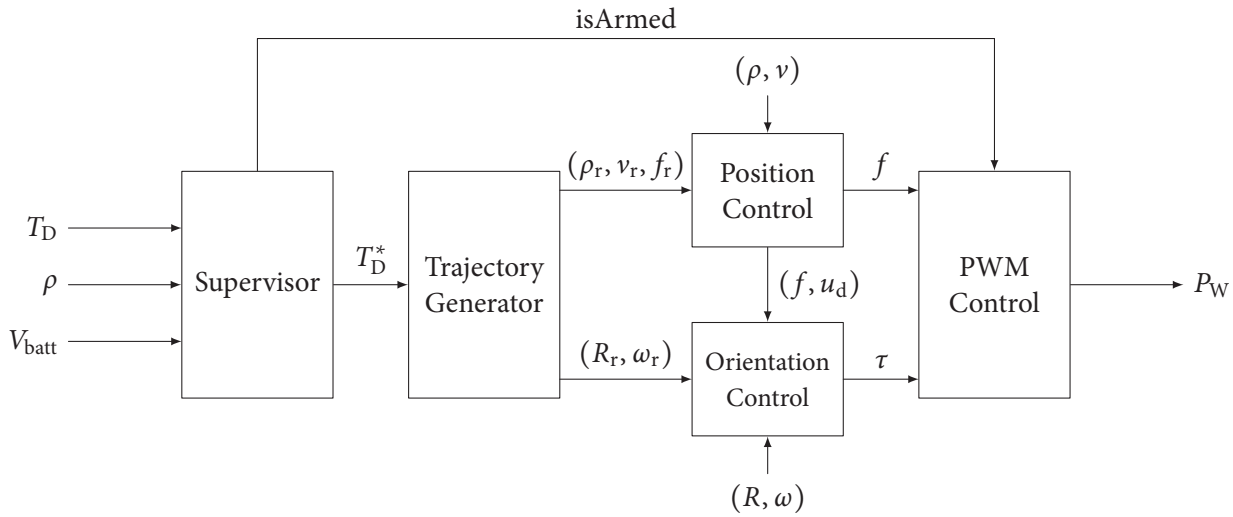

Figure 3.1: Schematic representation of the quad-rotor control structure.

In this control structure, the PWM control calculates the required PWM signals that the drone's interface can interpret, based on the desired force f and torques τ. The position and orientation controllers receive position and orientation feedback signals and aim to track the position, velocity, orientation, and angular velocity as defined by the trajectory generator. This trajectory generator contains several predefined trajectories and switches between them as the desired supervisor trajectory T_D^* is changed. The predefined trajectories are defined as position, velocity, and acceleration components in the inertial frame $\mathcal{I}$, which the trajectory generator converts to the desired ρ_r, v_r, R_r, ω_r, f_r, and τ_r such that the following set of equations is satisfied:

$$\begin{cases} \dot{\rho}_r = R_r v_r \\ \dot{v}_r = -S(\omega_r) v_r + M^{-1}\left(B_f f_r + G_r\right) \\ \dot{R}_r = R_r S(\omega_r) \\ J\omega_r = S(J\omega_r)\omega_r + \tau_r. \end{cases} \tag{3.1}$$

Originally, we aimed to use the position control law proposed by Choi and Ahn (2015). However, we found that their analysis of stability failed in the choice of control gains, proof is given in Appendix **??**. Hence,

in this chapter we discuss alternative position and orientation controllers that were designed by Van den Eijnden (2017), followed by a supervisor and PWM control. During the construction of these controllers, we assume that drag components as described in (2.5) can be neglected as we assume that velocities are relatively small. Since the drag term is quadratically dependent on the drone's velocity, during slow trajectories the influence on the dynamics is small. If faster trajectories are desired, however, this assumption no longer hold and an extension of the control law might be required to improve performance.

3.1 Position control by Van den Eijnden (2017)

In this section we discuss a position controller that is designed by Van den Eijnden (2017). We, however, only give the resulting controller. For the stability analysis and derivation of this controller see his work.

A new coordinate system $\mathcal{R}$ is defined, resulting from the desired position and orientation. These desired coordinates are calculated, based on the desired trajectory, using the model of the drone. We then can define error coordinates that describe the error of the drone's position and velocity with repect to the reference position and velocity by:

$$e_\rho = R^\top (\rho_\mathrm{r} - \rho) \tag{3.2}$$
$$e_v = v_\mathrm{r} - R_\mathrm{r}^\top R v, \tag{3.3}$$

leading to the following error dynamics:

$$\dot{e}_\rho = -S(\omega_\mathrm{r}) e_\rho + e_v \tag{3.4}$$
$$\dot{e}_v = -S(\omega_\mathrm{r})\, e_v + M^{-1}\left(B_f f_\mathrm{r} - R_\mathrm{r}^\top R B_f f\right). \tag{3.5}$$

By defining a new state vector $\left[e_\rho, e_v\right]^\top \in \mathbb{R}^6$, the error dynamics can be expressed as a linear time-varying state-space equation as

$$\dot{e} = Ae + Bu, \tag{3.6}$$

where $A = \begin{bmatrix} -S(\omega_\mathrm{r}) & I \\ 0 & -S(\omega_\mathrm{r}) \end{bmatrix}$, $B = \begin{bmatrix} 0 \\ I \end{bmatrix}$, and $u = M^{-1}\left(B_f f_\mathrm{r} - R_\mathrm{r}^\top R B_f f\right) \in \mathbb{R}^3$.

The errors in orientation and orientational velocity can be expressed as:

$$R_e = R_\mathrm{r}^\top R \tag{3.7}$$
$$J\omega_e = J\omega - JR^\top R_\mathrm{r} \omega_\mathrm{r}, \tag{3.8}$$

with corresponding error dynamics:

$$\dot{R}_e = R_\mathrm{r}^\top R S(\omega_e) \tag{3.9}$$
$$J\dot{\omega}_e = S(J\omega)\omega - JS(\omega_e)R^\top R_\mathrm{r}\omega_\mathrm{r} - JR^\top R_\mathrm{r}\dot{\omega}_\mathrm{r} + \tau. \tag{3.10}$$

As the angular velocity dynamics are fully actuated, for control purposes it is convenient to express ω_e in the body-fixed frame $\mathcal{B}$.

The controller that has been designed by Van den Eijnden (2017) considers the system as defined in (3.6) in closed-loop with desired control input

$$u_\mathrm{d} = R_\mathrm{r}^\top (k_z z + k_w w) - \frac{k_1}{\sqrt{1 + \|\overline{e}_\rho\|^2}}\overline{e}_\rho - \frac{k_2}{\sqrt{1 + \|\overline{e}_v\|^2}}\overline{e}_v, \tag{3.11}$$

where $\overline{e}_\rho = e_\rho + R_r^\top w$, $\overline{e}_v = e_v + R_r^\top z$, and w and z are solutions of the artificial subsystem

$$\begin{cases} \dot{w} = z \\ \dot{z} = -k_z z - k_w w + k_w R_r \sigma_n(\overline{e}_\rho), \end{cases} \tag{3.12}$$

with $(w(t_0), z(t_0)) = (0, 0)$, and $\sigma_n(\overline{e}_\rho)$ a smooth saturation function, defined as:

$$\sigma_n(\overline{e}_\rho) = \frac{\overline{e}_\rho}{\left(1 + (\overline{e}_\rho^\top \overline{e}_\rho)^{\frac{n}{2}}\right)^{\frac{1}{n}}}, \tag{3.13}$$

with $n \in 2\mathbb{N}^+$. The control parameters need to be chosen such that $k_1 > k_w\sqrt{2} > 0$ and $k_2, k_z, k_w > 0$, and we assume that ω_r is continuous and bounded for all $t \geq t_0$. Under these assumptions, the zero solution $e = 0$ of the closed-loop system is uniformly asymptotically stable for all $\|(w(t_0), z(t_0))\| < \sqrt{1 + \left(\frac{k_w+1}{k-z}\right)^2}$. For the controller derivation and proof of stability, see the work of Van den Eijnden (2017).

3.2 Orientation control

Using the desired control input of the position controller (3.11), we can rewrite expression (3.6) as:

$$\dot{e} = Ae + Bu_d + B(u - u_d). \tag{3.14}$$

The input vector $u - u_d$ can be separated in terms of an input vector $F = \left(R_r^\top R B_f\right) f$ and a desired input vector $F_d = B_f f - M u_d$, such that $u - u_d = M^{-1}(F_d - F)$, which are the actual and desired thrust vectors of the quad-rotor, expressed in $\mathcal{R}$. By letting F converge to F_d, the drone is steered to the desired orientation, corresponding to the desired trajectory.

In order to achieve this convergence, the thrust force f can be used to ensure $\|F\| = \|F_d\|$. As the magnitude of a vector is invariant under rotation, it holds that $\|F\| = f$. The force input f can therefore be chosen as:

$$f = \|F_d\|_2 = \left\|B_f f_r - M u_d\right\|_2, \tag{3.15}$$

assuming $0 < f_{\min} \leq f_r(t) \leq f_{\max}$, as the quad-rotor's actuators cannot provide negative thrust and the maximum achievable thrust is limited. In order to enforce $\|F_d\|_2 > 0$, a condition on the stabilising function is imposed:

$$\|u_d\|_2 \leq k_z + \frac{(k_w + 1)^2}{k_z} + k_1 + k_2 \leq \frac{f_{\min}}{m}. \tag{3.16}$$

If k_w, k_z, k_1, and k_2 are positive and chosen such that condtition (3.16) is satisfied and the initial conditions of 3.12 are chosen such that $\|(w(t_0), z(t_0))\| < \sqrt{1 + \left(\frac{k_w+1}{k-z}\right)^2}$, then $f > 0$ can be guaranteed.

Next, the orientation of F is steered, such that it aligns with the desired thrust vector F_d. Hereto, two normalised thrust vectors are defined:

$$\hat{f} = \frac{F}{\|F\|} = R_e B_f \tag{3.17}$$

$$\hat{f}_d = \frac{F_d}{\|F_d\|} = R_d B_f. \tag{3.18}$$

Additionally, a new coordinate system, $\mathcal{D}$, is defined: the heading direction frame. The origin of $\mathcal{D}$ coincides with the body-fixed frame $\mathcal{B}$ and the rotation matrix $R_d \in SO(3)$ maps the desired heading reference

frame $\mathcal{D}$ to the tracking reference frame $\mathcal{R}$.

The rotation matrix R_d is defined as

$$R_\mathrm{d} = \begin{bmatrix} \frac{r_{1,\mathrm{d}}}{\|r_{1,\mathrm{d}}\|} & \frac{r_{2,\mathrm{d}}}{\|r_{2,\mathrm{d}}\|} & \hat{f}_\mathrm{d} \end{bmatrix}, \tag{3.19}$$

where $r_{1,\mathrm{d}} = r_{2,\mathrm{d}} \times \hat{f}_\mathrm{d}$, $r_{2,\mathrm{d}} = \hat{f}_\mathrm{d} \times \hat{i}$, and $\hat{i} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$. As we aim to align $\hat{f}$ with $\hat{f}_\mathrm{d}$, the definition of these vectors in (3.17) and (3.18) allows for a new set of intuitive error coordinates

$$e_f = R^\top R_\mathrm{d} \left(\hat{f}_\mathrm{d} - \hat{f} \right) = \left(\hat{R}_e - I \right) B_f \tag{3.20}$$

$$J e_\omega = J R^\top R_\mathrm{r} \left(\omega_\mathrm{d} + \omega_\mathrm{r} \right) - J\omega, \tag{3.21}$$

expressed in $\mathcal{B}$, where $\hat{R}_e = \left(R^\top R_\mathrm{r} \right) R_\mathrm{d}$ and $\omega_\mathrm{d} = \frac{F_\mathrm{d} \times \dot{F}_\mathrm{d}}{\|F_\mathrm{d}\|^2}$.

The attitude dynamics now can be expressed as

$$\dot{\hat{R}}_e = S(e_\omega) \hat{R}_e \tag{3.22}$$

$$J\dot{\omega} = -JS(\omega_e) R^\top R_\mathrm{r} \left(\omega_\mathrm{d} + \omega_\mathrm{r} \right) + J R^\top R_\mathrm{r} \left(\dot{\omega}_\mathrm{d} + \dot{\omega}_\mathrm{r} \right) - J\dot{\omega}, \tag{3.23}$$

where $\omega_e = \omega - R^\top R_\mathrm{r} \omega_\mathrm{r}$ is the angular velocity error between $\mathcal{B}$ and $\mathcal{R}$, expressed in $\mathcal{B}$.

Now, the control input τ can be defined as

$$\tau = S(\omega) J\omega + J(\lambda + \mu), \tag{3.24}$$

with

$$\lambda = -S(\omega_e) R^\top R_\mathrm{r} \left(\omega_\mathrm{d} + \omega_\mathrm{r} \right) + R^\top R_\mathrm{r} \left(\dot{\omega}_\mathrm{d} + \dot{\omega}_\mathrm{r} \right) \tag{3.25}$$

$$\mu = -c_1 \sum_{i=1}^{3} \sigma_i \left(s_i \times \hat{R}_e s_i \right) - c_2 e_\omega, \tag{3.26}$$

where s_i, $i \in [1, 2, 3]$ are the respective columns of the identity matrix I, σ_i are distinct positive numbers, and c_1, $c_2 > 0$ are positive control gains. The control law (3.24) almost-globally uniformly asymptotically stabilises the solutions of the closed-loop system (3.22) - (3.24) to $\hat{R}_e = I$ and $e_\omega = 0$. All solutions, with the exception of solutions starting in a set $\mathcal{M} \subset SO(3) \times \mathbb{R}^3$ with zero Lebesque measure, converge ot the desired equilibrium $(I, 0)$. Therefore, F converges to F_d asymptotically for any initial condition not in this subset. For an in-depth derivation of this controller and proof of stability, see the work of Van den Eijnden (2017).

3.3 SUPERVISOR

A supervisory controller is used to engage different desired trajectories, and ensure safe and proper functioning of the quad rotor. For proper operation, we require a sufficient battery voltage and non-blocked rotors. A schematic view of the supervisor is given in Figure 3.2. Other safety requirements, such as upright starting orientation or limited rotations, could be added to the supervisor for additional safety guarantees. Below, we give a description of the supervisor behaviour during normal operation.

When the controlling software is initiated on the AR.Drone 2.0, the supervisor starts in *Initialise* and

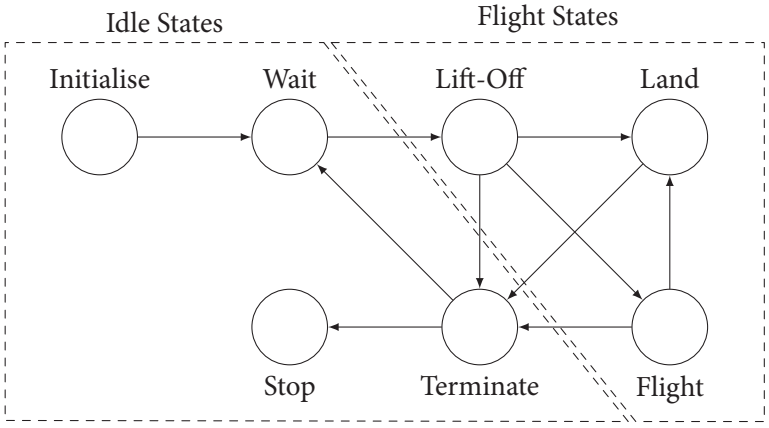

Figure 3.2: Schematic view of the supervisory control structure.

remains there for 10 seconds, during which sensors are measured and recalibrated such that drift is eliminated. Subsequently, the *Wait* is reached, where the supervisor remains if no trajectory is desired, or the desired trajectory T_D is to wait. Once a trajectory is requested by the user, the supervisor initially reaches *Lift-Off*, ordering an adequate lift-off trajectory from the trajectory generator prior to reaching *Flight*, where the actual desired trajectory is passed to the trajectory generator. Once the user commands the drone to land, the supervisor state continues to *Land*, where the altitude is decreased gradually until the quad-rotor hovers only centimetres above the ground, then the state reaches *Terminate*, where the motors are shut down and disarmed. Once these steps are complete, the state proceeds to *Wait* during normal operation.

For safe operation, the battery voltage should be greater than a critical value. When this critical voltage is surpassed, this is an indication that the battery is nearly depleted, hence prolonged safe flight is no longer possible. In such situation, the supervisor reaches *Land*, ordering the trajectory generator to engage a landing sequence, regardless of the user desired trajectory. Subsequently, the *Terminate* and *Stop* states are reached once the landing sequence is completed.

In addition to the behaviour described above, any flight state can reach *Terminate* directly in case of an emergency. One example is blockage of one of the rotors, in this case the rotors are instantaneously disarmed and the drone is not able to re-arm them until the software is restarted. Additional requirements can be implemented for higher safety standards. During any idle state, all rotors are disarmed such that they are disabled, regardless of the desired control force f or torques τ. This ensures that the rotors cannot rotate unexpectedly.

3.4 PWM calculation

The position and orientation controllers output the desired force f and torques τ for position tracking. The Parrot AR.Drone 2.0, however cannot receive these force and torques directly, they need conversion to PWM values for each rotor. Using the motor model (2.18) from Section 2.2, we can relate the desired f and τ to the corresponding $P_{W,i}$, $i \in [1, 2, 3, 4]$. Hereto, we assume a linear relation between the rotor's delivered thrust and torque:

$$\tau_z = \begin{bmatrix} c_{d,1} & -c_{d,2} & c_{d,3} & -c_{d,4} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}, \tag{3.27}$$

where $c_{\mathrm{d},i}$ are the rotors drag coefficients. Using (3.27), we can express (2.18) as

$$\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ l & -l & -l & l \\ -l & -l & l & l \\ c_{\mathrm{d},1} & -c_{\mathrm{d},2} & c_{\mathrm{d},3} & -c_{\mathrm{d},4} \end{bmatrix} \begin{bmatrix} f \\ \tau_x \\ \tau_y \\ \tau_z \end{bmatrix}, \tag{3.28}$$

relating the required rotor thrusts f_i to the desired force f and torques τ. Now, using (2.20) and the quadratic formula, we can extract the corresponding PWM values for each rotor by

$$P^*_{\mathrm{W},i} = \frac{-b_{\mathrm{r}f,i} + \sqrt{b^2_{\mathrm{r}f,i} - 4a_{\mathrm{r}f,i}\left(c_{\mathrm{r}f,i} - f_i\right)}}{2a_{\mathrm{r}f,i}}. \tag{3.29}$$

As the rotors should only be active when armed, we define the final PWM values for each rotor as

$$P_{\mathrm{W},i} = \begin{cases} P^*_{\mathrm{W},i} & \text{if isArmed} \\ 0 & \text{otherwise.} \end{cases} \tag{3.30}$$

3.5 SUMMARY

In this chapter we have given the control structure of the software to be implemented on the AR.Drone 2.0. We discussed the position and orientation controllers as designed by Van den Eijnden (2017) that stabilise the drone to a desired trajectory and outputting the desired thrust and torques. Next, we gave a simple supervisory controller for proper operation and safety. Finally, we discussed how the desired thrust and torques, output by the position and orientation controllers, are converted to the PWM values the drone can receive by using the model discussed in Section 2.2.

Now that we are able to steer the drone using the controllers discussed in this chapter, in the following chapter we discuss how the sensory equipment available on the drone is used to reconstruct the drone's states, such that feedback of the position, velocity, orientation, and angular velocity can be used by the controllers discussed in this chapter.

Chapter 4

State reconstruction from sensor data

In this chapter we determine the orientation based on the data provided by the inertial measurement unit (IMU). This set of sensors provides body-fixed acceleration, angular velocity, and geomagnetic field vectors that can be used to determine the rotation between the earth-fixed reference coordinate frame and the body-fixed coordinate frame. Additionally we determine the position of the drone with respect to a point (marker) on the floor using the on-board camera and the calculated orientation.

Whereas the accelerometer and magnetometer sensors are subject to noise and disturbances, we also provide observers that minimise these effects. Kalman filters are used to estimate the true gravity and magnetometer vectors by means of sensor fusion of the gyroscope and accelerometer for the gravity vector and gyroscope, magnetometer, and camera for the magnetic field vector. Since the velocity of the drone cannot be measured directly, a velocity estimator is required.

In this chapter we firstly discuss how the orientation can be determined if estimates for the gravitational and magnetic field vectors in the body-fixed frame are given, followed in Section 4.2 by how the internal camera can be used to determine the position of the drone. In the second half of this chapter, Section 4.3, we discuss the orientation observer that is used for estimation of the gravity and magnetic field vectors in the body-fixed frame using the sensory equipment. This observer additionally uses the derived method of Section 4.1 to calculate the resulting rotation. Then, as no velocity information is provided by the sensory equipment, we briefly discuss the velocity observer as designed by Van den Eijnden (2017), in order to estimate not only the velocity, but also estimate the position in-between sensor measurements for the low-frequency position measurements.

4.1 Determination of the orientation

In this section we discuss how the drone's orientation can be determined using the estimated gravity and geo-magnetic field vectors with respect to the body fixed frame. The actual estimation of these vectors is covered in Section 4.3.

Given the body-fixed gravity and geomagnetic field vectors, $g^{\mathcal{B}}$ and $B^{\mathcal{B}}$, the rotation between the earth-fixed coordinate frame $\mathcal{I}$ and the body-fixed frame $\mathcal{B}$ can be calculated. Firstly, using the gravity vector, we can determine the roll and pitch angles:

$$\phi = \operatorname{atan}_2\left(g_y^{\mathcal{B}},\, g_z^{\mathcal{B}}\right) \tag{4.1}$$

$$\theta = -\operatorname{atan}_2\left(g_x^{\mathcal{B}},\, \sqrt{(g_y^{\mathcal{B}})^2 + (g_z^{\mathcal{B}})^2}\right). \tag{4.2}$$

Then, we rotate the measured magnetic field vector in (4.3), such that we subsequently can determine the yaw angle, ψ, from the resulting vector.

$$B^{\mathcal{B}*} = R_y(\theta)R_x(\phi)B^{\mathcal{B}}. \tag{4.3}$$

$$\psi = -\operatorname{atan}_2\left(B_y^{\mathcal{B}*}, B_x^{\mathcal{B}*}\right). \tag{4.4}$$

Using (4.1), (4.2), and (4.4), the rotation matrix describing the rotation from the boxy-fixed frame to the earth-fixed coordinate frame can be expressed as:

$$R = R_z(\psi)R_y(\theta)R_x(\phi), \tag{4.5}$$

where

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

This gives us a method to determine the rotation matrix that describes the rotation from the body-fixed frame $\mathcal{B}$ to the inertial frame $\mathcal{I}$. In next section we discuss how the on-board, down facing camera can be used to determine the position of the drone with respect to some marker.

4.2 Position measurement using the on-board camera

The body-fixed, down-facing, internal camera of the AR.Drone 2.0 can be used to determine the position of the drone with respect to some marker. Firstly, we isolate the marker from the camera image and subsequently we can correct for perspective distortion caused by orientation changes of the drone and, therefore, the camera. We use a marker that is comprised of a blue and red LED, which enables us not only to determine the drone's position with respect to this marker, but additionally the drone's heading. This section is divided in two parts. In the first part we discuss how the marker location can be extracted from the camera image, in the second part of this section, we use geometry to project the marker location from the image plane to the floor in the real world, enabling us to determine the location of the drone with respect to the marker.

4.2.1 Marker detection

The internal down-facing camera can be used to track a marker's position, as long as it remains in the camera's view. The internal camera of the AR.Drone 2.0 outputs an image vector, containing information for three colour channels defined in a $YCbCr$ colour format. All three channels are of the unsigned 8-bit

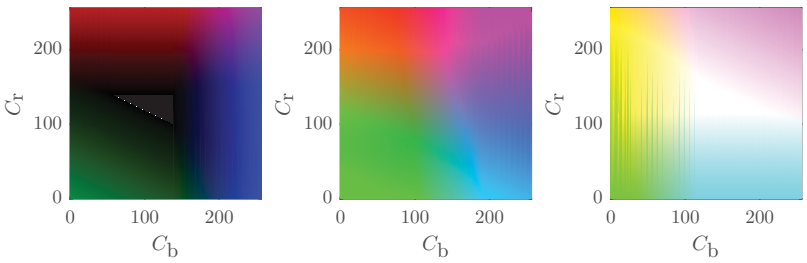

Figure 4.1: The C_bC_r plane at constant luminance values $Y = 0$ (left), $Y = 128$ (centre), and $Y = 255$ (right).

integer type and, therefore, range in $[0, 255]$. In this colour format, the luminance channel (Y-channel) represents the image brightness and the two chrominance channels (C_b and C_r) contain the colour differences. Figure 4.1 shows the C_bC_r plane at three constant luminance values. We can use this colour format in our advantage by choosing a two-tone marker with a red and a blue LED as these two colours lie far apart on the C_bC_r plane.

Figure 4.2: The complete camera image that is used to isolate the two-LED marker (left) and a zoomed-in section of this image (right).

An example camera image of the down-facing camera, aimed at the marker, is given left in Figure 4.2, with a zoomed-in section for clarity at the right. This image is used to determine the location of the blue and red LEDs. In order to isolate the LED locations, we create two binary images: one has a value of 1 at the blue LED and 0 otherwise, one has a value of 1 at the red LED and 0 otherwise. These binary images can be used in a blob-detection algorithm to determine the locations of the LEDs.

To create the binary image for the red LED, we use the C_r channel. We define a threshold such that pixels with a sufficiently high C_r value yield a value of 1 in the binary image, and 0 otherwise. The C_b channel is used similarly for the blue LED.

$$[B_b]_{i,j} = \begin{cases} 1 & \text{if } [C_b]_{i,j} > ((1-0.3)\max(C_b) + 0.3\min(C_b)) \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

$$[B_r]_{i,j} = \begin{cases} 1 & \text{if } [C_r]_{i,j} > ((1-0.3)\max(C_r) + 0.3\min(C_r)) \\ 0 & \text{otherwise.} \end{cases} \tag{4.7}$$

Herein are B_b and B_r the blue and red binary image matrices, and the subscript $_{i,j}$ indicates a specific matrix element, with $i \in [1, 2, \ldots, 240]$ and $j \in [1, 2, \ldots, 160]$. Figure 4.3 shows grey-scale images that represent the value of the chrominance channels, where dark an light areas indicate low and high values, respectively. Additionally, the corresponding binary images, generated by (4.6) and (4.7), are given in this figure.

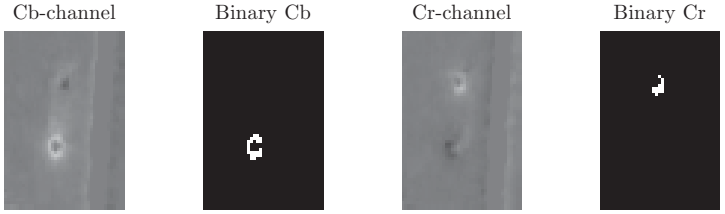

Figure 4.3: The camera C_b and C_r channels with their corresponding binary images.

Once we obtain these binary images, in Simulink, we can use the blob detection block to determine the centroid positions blue and red regions of the image. We assume that the largest blob, in terms of area, is the result of the coloured LED. Note, however, that the C_b and C_r channels are only half resolution matrices of 160 × 240, whereas the Y channel has a resolution of 320 × 240. Hence, resulting centroid position requires a multiplication of 2 in y-direction. Also note that the brightest C_b and C_r pixels are not exactly at the position of the LEDs, but encircle the LED locations. This can mean that the output of the blob detection algorithm not necessarily is the best approximation of the LED location in the image. Hence, additionally, we can use the Y camera channel, which represents the brightness of the image, as an indication for the actual LED locations. Using (4.8) the Y channel can be converted to a binary image that can be used by the blob detection block in Simulink. The resulting binary image is given in Figure 4.4.

$$[B_Y]_{i,j} = \begin{cases} 1 & \text{if } Y_{i,j} > ((1-0.1)\max(Y) + 0.1\min(Y)) \\ 0 & \text{otherwise.} \end{cases} \tag{4.8}$$

Note that, whereas the Y channel is a full resolution image of 320 × 240 pixels, we choose to reduce this

Figure 4.4: The camera half resolution Y channel with corresponding binary image.

resolution to the same size as the C_b and C_r channels to reduce the computational resource requirement in the blob detection algorithm significantly, such that the AR.Drone 2.0 is able to perform this operation at a sufficient sample rate.

The blob detection block might output more blobs than there are LEDs visible in the Y channel of the image. Also, from the Y channel alone, no distinction between a blue and red LED can be made. Hence, we use the blob locations from the C_b and C_r channels to determine which blob in the Y channel is a red or blue LED.

Let us define L_b^* and L_r^* the centroid locations of the largest blobs in the blue and red binary images, respectively and $L_{Y,k}^*$ the centroid location of the k^{th} blob in the binary image B_Y. We then can use Algorithm 4.1 to determine a more accurate location of the red and blue LEDs in the image. In the blob distinction algorithm of Algorithm 4.1, n is the number of blobs in the Y channel, $L_{Y,k}^*$ is the centroid of blob k in binary image B_Y, $d_{b,k}$ and $d_{r,k}$ are the distances from $L_{Y,k}^*$ to L_b^* and L_r^*, respectively, and L_b and L_r are the locations of the blue and red LEDs in the image.

This now gives us the locations of the red and blue LEDs in the image frame. These locations, however, are measured in pixels an are subject to perspective distortions when the camera is not aimed perfectly vertically to the floor. Hence, in the next subsection we discuss how the actual position of the drone, in metres, can be extracted from the marker locations in the image, measured in pixels.

Algorithm 4.1 Blob distinction algorithm.

1: **for** $k = 1 \rightarrow n$ **do**
2: $\quad d_{b,k} = \|L^*_{Y,k} - L^*_{\mathrm{b}}\|_2$
3: $\quad d_{r,k} = \|L^*_{Y,k} - L^*_{\mathrm{r}}\|_2$
4: **end for**
5: $k_b = \mathrm{find}\,(k \,|\, d_{b,k} = \min(d_b))$
6: $k_r = \mathrm{find}\,(k \,|\, d_{r,k} = \min(d_r))$
7: $L_{\mathrm{b}} = L^*_{Y,k_b}$
8: $L_{\mathrm{r}} = L^*_{Y,k_r}$

4.2.2 Perspective correction

In this subsection we discuss how the drone's location can be determined using the marker location as determined in the previous subsection. To achieve this, we first convert the marker location in pixels to metres and then project the marker on the image plane to the xy-plane of the inertial frame $\mathcal{I}$.

Consider a known point p on the xy-plane of the inertial frame $\mathcal{I}$ fixed to the world and its projection p' on an image plane, where the image plane is a plane parallel to the xy-plane of the body(camera)-fixed frame $\mathcal{B}$ at a distance f (focal distance). Note that the focal distance can be chosen arbitrarily. Here, however we choose the focal distance such that the image plane intersects drone's projection on the $x^{\mathcal{I}}$, $y^{\mathcal{I}}$-plane, easing the projection from a point on the image plane to the $x^{\mathcal{I}}$, $y^{\mathcal{I}}$-plane. A graphical representation is given in Figure 4.5. Given the drone's altitude, h, and the rotation matrix $R^\top$ that describes

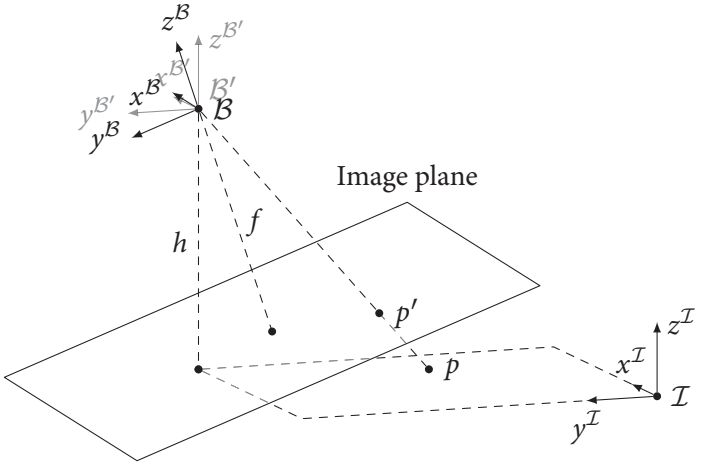

Figure 4.5: The frame $\mathcal{B}'$ is a frame that is only translated to the drone position, such that $\mathcal{B}$ and $\mathcal{B}'$ share a common origin. The video images are placed upon the focal plane, which is parallel to the xy-frame of $\mathcal{B}$ at a distance f in negative $z^{\mathcal{B}}$-direction. The point p on the xy-plane of $\mathcal{I}$ is projected to the image plane.

the rotation from $\mathcal{B}'$ to $\mathcal{B}$, the focal distance, f, can be determined as follows:

$$f = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R^\top \begin{bmatrix} 0 \\ 0 \\ h \end{bmatrix}. \tag{4.9}$$

Using the focal distance f, we can convert the location of a point p' on the image frame in pixels to metres. Hereto we use the properties that the camera has a field of view (f.o.v.) of 64° and a resolution of 320×240

pixels. Usually the f.o.v. is measured diagonally, which enables us to convert pixels to metres if we know the distance from our object. Since we place the image on the focal plane of Figure 4.5, the distance fro m our object is known: f. Hence the position of p' in pixels can be converted to metres by

$$p'_{[m]} = 2f \tan\left(\frac{64^\circ}{2}\right)\left(\sqrt{240^2 + 320^2}\right)^{-1} p'_{[px]}. \tag{4.10}$$

If we have a point of interest in our image (marker), the coordinates of that point, p', can be expressed in the body-fixed frame as

$$p'^{\mathcal{B}} = \begin{bmatrix} p'_x & p'_y & -f \end{bmatrix}^\top, \tag{4.11}$$

where $p'B$ is a vector containing the coordinates of p' with respect to the body-fixed frame, and p'_x and p'_y are the x- and y-coordinates of this point in the image, respectively. If we now transform these coordinates to the $\mathcal{B}'$ frame,

$$p'^{\mathcal{B}'} = R^\top p'^{\mathcal{B}}, \tag{4.12}$$

and scale (4.12) according to (4.13), such that $p^{\mathcal{B}'}(3) = h$, the result is a vector with the coordinates of point p with respect to $\mathcal{B}'$.

$$p^{\mathcal{B}'} = \frac{h}{p'^{\mathcal{B}'}(3)} p'^{\mathcal{B}'}. \tag{4.13}$$

The drone's position with respect to the inertial frame can be determined by

$$\rho = p_r - p^{\mathcal{B}'}, \tag{4.14}$$

where p_r is the vector describing the location of the marker in the inertial frame.

In this section we discussed means to determine the position of the drone using the AR.Drone 2.0's internal sensory equipment. We firstly determine the location of a two-LED marker in the camera image, subsequently converting the marker measurement to pixels, followed by a perspective correction and projection to the xy-plane of the inertial frame $\mathcal{I}$. This calculated marker position is with respect to the body fixed-frame and can be used to determine the quad-rotor's position ρ in the inertial frame $\mathcal{I}$.

4.3 Orientation observer

Previously, we discussed how the estimated gravity and magnetic field vectors can be used to estimate the drone's orientation. In this section, we discuss an observer that uses signals from several sensors and fuses them to accurately estimate gravity and magnetic field vectors, such that methods described in Section 4.1 can be used to estimate the orientation accurately.

Since sensors are subject to noise, disturbances, and signal bias due to, for example, temperature effects which cannot be cancelled completely, one can combine measurement data from several sensors to determine estimates for the gravitation and geo-magnetic field vectors. Whereas the noise of the three-axis gyroscope is relatively small, it is not recommended to simply integrate the body-fixed angular velocities in order to determine the relative orientation with respect to the earth frame, as a drift may occur due to gyroscope signal biases.

In this section, we firstly discuss the general structure of the observer. Subsequently, we discuss how the signals of the accelerometer and gyroscope are fused into the estimation of the gravity vector. Lastly, we discuss how we not only fuse the magnetometer with the gyroscope measurements, but also add the heading as determined by the camera and markers, for more accurate heading estimation when the magnetometer data is compromised.

4.3.1 Observer structure

The observer that is used is based on two linear Kalman filters in parallel to filter the accelerometer and magnetometer measurements with the gyroscope data, as proposed by Zhang, Yu, Liu, Yuan, and Liu (2016). Additionally, an R-adaptation algorithm (W. Li & Wang, 2012; Sabatini, 2011) is implemented, which modifies the Kalman gain in the presence of sensor disturbances. However, since we use the camera in addition to the magnetometer and gyroscope to determine the heading of the drone, the structure of the orientation observer changes slightly. A schematic representation is given in Figure 4.6.

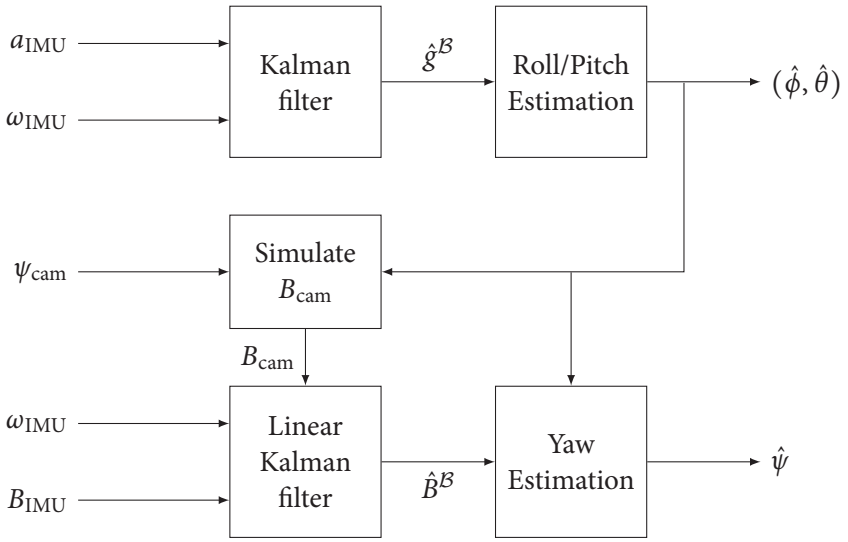

Figure 4.6: Structure of the adapted orientation observer, comprised of two linear Kalman filters used to estimate the gravity vector and geomagnetic field vector in $\mathcal{B}$, which can be used to determine the drone's orientation. The signals a_{IMU}, ω_{IMU}, and B_{IMU} are the measured accelerations by the accelerometer, angular velocities as measured by the gyroscope, and the measured magnetic field vector respectively and ψ_{cam} is the heading of the drone as measured by the camera. The signals $\hat{g}^{\mathcal{B}}$ and $\hat{B}^{\mathcal{B}}$ are the estimated gravity and geomagnetic field vectors in the body fixed frame and $\hat{\phi}$, $\hat{\theta}$, and $\hat{\psi}$ are the estimated roll, pitch, and yaw angles, respectively.

The unified discrete-time dynamic equations, describing the rotation of a vector in the body-fixed coordinate system, can be expressed as:

$$\begin{cases} x^k = \Phi x^{k-1} + \mu^{k-1} \\ z^k = H x^k + v^k, \end{cases} \tag{4.15}$$

where μ and v are the process and measurement noise vectors, the measurement matrix H depends on the number of sensors and what they measure. Specific measurement matrices are given in the following subsections, where sensor measurements are fused. Note that this system is defined in discrete-time where the controller and model are defined in continuous-time. During implementation, however, sensor signals are sampled at a rate of 400 Hz. Hence, a discrete-time filter is used to estimate the gravity and magnetic field vectors. In (4.15), Φ is defined as

$$\Phi = \exp\left(- \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} t_s \right). \tag{4.16}$$

where p, q, and r are the angular rates around the x, y, and z-axes of the body fixed frame, respectively. The filter algorithm is given in Algorithm 4.2.

Algorithm 4.2 Kalman filter algorithm.

1: $\tilde{x}^k = \Phi_k \hat{x}^{k-1}$
2: $P_k^- = \Phi_k P_{k-1} \Phi_k^\top + Q_k$
3: $K_k = P_k^- H^\top \left(H P_k^- H^\top + \lambda R_z \right)^{-1}$
4: $\hat{x}^k = \tilde{x}^k + K_k \left(z^k - H \tilde{x}^k \right)$
5: $P_k = \frac{1}{\lambda} \left(I - K_k H \right) P_k^-$

Herein $Q = L R_g L^\top$ is the process noise covariance matrix with

$$R_g = \text{diag}\begin{bmatrix} \sigma_p^2 & \sigma_q^2 & \sigma_r^2 \end{bmatrix}, \qquad L = \begin{bmatrix} 0 & x_3 & -x_2 \\ -x_3 & 0 & x_1 \\ x_2 & -x_1 & 0 \end{bmatrix}$$

where σ_p^2, σ_q^2, and σ_r^2 are the variances of p, q, and r. The values are determined by determination of the variance of a sensor readout, as per Appendix C.1. The matrix R_z is the sensor covariance matrix.

4.3.2 Fusion of accelerometer and gyroscope

In order to estimate the roll and pitch angles ϕ and θ accurately using (4.1) and (4.2), the acceleration vector as a result of gravity needs to be estimated accurately. Hence, we fuse the accelerometer and gyroscope readings using the Kalman filter of Algorithm 4.2. The sensor matrix for this fusion system is defined as $H = I^{3\times3}$.

When the drone is subject to accelerations, we desire to rely more on the gyroscope, as the accelerometer in such situation no longer represents the gravity vector. Hence, we use the R-adaptation algorithm by W. Li and Wang (2012) to modify the Kalman gain temporarily.

Suppose $\alpha_{\text{Acc}} = |\|a_{\text{IMU}}\| - \|g\||$, then:

$$R_{\text{Acc}} = \begin{cases} R_{\text{Acc,0}} & \text{for } \alpha_{\text{Acc}} \leq \xi_{\text{a}} \\ R_{\text{Acc,0}} + k\alpha^2 I^{3\times3} & \text{for } \xi_a < \alpha_{\text{Acc}} \leq \xi_b \\ \gamma I^{3\times3} & \text{for } \alpha_{\text{Acc}} > \xi_{\text{b}} \end{cases} \tag{4.17}$$

where $R_{\text{Acc,0}} = \text{diag}\begin{bmatrix} \sigma_{\text{Acc},x}^2 & \sigma_{\text{Acc},y}^2 & \sigma_{\text{Acc},z}^2 \end{bmatrix}$ is the accelerometer noise covariance matrix, ξ_a is a threshold value that is suggested to equal $\xi_{\text{a}} = \sqrt{\sigma_{\text{Acc},x}^2 + \sigma_{\text{Acc},y}^2 + \sigma_{\text{Acc},z}^2}$, k is some tunable constant and ξ_{b} is a threshold value that needs to be determined by simulations and experiments. The tuning parameter γ is chosen such that the Kalman gain reduces to zero in the extreme case where $\alpha_{\text{Acc}} > \xi_{\text{b}}$. Ideally, the value of this parameter is chosen to be infinity. In practice, however, γ is chosen very large ($\gamma = 10^{10}$, for example), as infinite elements might introduce calculation difficulties.

Whereas this works satisfactory in simulations, even with the introduction of noise, in practice, disturbances are introduced by rotor imbalance, for example. Such imbalance introduces high acceleration disturbances, which disturb the accelerometer gravity vector measurement. Hence, the parameter λ in the gravity vector estimator, which usually is chosen in the order of magnitude of 1, has been raised to a value of $\lambda = 100$. By raising the value of λ, the Kalman gain is reduced such that, essentially, the gyroscopes are being integrated in order to estimate the gravity vector in the body-fixed frame. The value of λ, however, is chosen sufficiently low in order to compensate for low frequency variations, such as drift

introduced by gyroscope bias terms.

The bias in the gyroscope signals is temperature dependent and is removed by calibration as described in Appendix A.4. As this calibration might not be perfect, remaining biases are removed by measuring a signal average during an initialisation period and subtracting that value from the incoming signal. Using these techniques, not only are the roll and pitch angles more accurate than before, the angle noise is reduced significantly as well, as the gyroscope's noise is smaller than that of the accelerometer disturbances introduced by the propeller imbalance since these are linear accelerations in the xy-plane of the body-fixed frame.

4.3.3 Fusion of magnetometer, camera and gyroscope

In order to estimate the heading angle ψ accurately using (4.4), the magnetic field vector needs to be estimated accurately. Hence, we fuse the magnetometer, camera, and gyroscope readings using the Kalman filter of Algorithm 4.2. The sensor matrix for this fusion system is defined as:

$$H = \begin{bmatrix} I^{3\times 3} \\ I^{3\times 3} \end{bmatrix}. \tag{4.18}$$

Given the estimated roll and pitch angles from the accelerometer $\hat{\phi}$ and $\hat{\theta}$, and the heading angle from the camera, ψ_{cam}, we can transform the earth's magnetic field vector, $B_{\text{o}}^{\mathcal{I}}$, from the inertial frame to the body-fixed frame:

$$B_{\text{cam}}^{\mathcal{B}} = R(\hat{\phi}, \hat{\theta}, \psi_{\text{cam}}) B_{\text{o}}, \tag{4.19}$$

where B_{o} is obtained from Matlab's magnetic field model `wrldmagm(.)`. We define the observation state vector $z = \begin{bmatrix} z_1 & z_2 \end{bmatrix}^\top$ with

$$z_1 = B_{\text{IMU}}^{\mathcal{B}}, \tag{4.20}$$

$$z_2 = B_{\text{cam}}^{\mathcal{B}}, \tag{4.21}$$

where $B_{\text{IMU}}^{\mathcal{B}}$ is the magnetic field vector as measured by the magnetometer and $B_{\text{cam}}^{\mathcal{B}}$ is the estimated magnetic field vector, based on the heading measurement from the camera, as expressed in (4.19).

Since the magnetometer is only reliable in circumstances where the earth's magnetic field is not disturbed and the camera is only reliable if a marker is detected, we need a manner to select which sensor to use in order to determine the heading ψ. Suppose $\alpha_{\text{mag}} = \left| \|B_{\text{IMU}}\| - \|B_{\text{o}}\| \right|$ and we define R_{z_1} and R_{z_2} as

$$R_{z_1} = \begin{cases} \text{diag}\begin{bmatrix} \sigma_{\text{mag},x}^2 & \sigma_{\text{mag},y}^2 & \sigma_{\text{mag},z}^2 \end{bmatrix} & \text{if } \alpha_{\text{mag}} \leq \xi_{\text{mag}} \\ \gamma I^{3\times 3} & \text{otherwise,} \end{cases} \tag{4.22}$$

$$R_{z_2} = \begin{cases} \text{diag}\begin{bmatrix} \sigma_{\text{cam},x}^2 & \sigma_{\text{cam},y}^2 & \sigma_{\text{cam},z}^2 \end{bmatrix} & \text{if } \exists \psi_{\text{cam}} \\ \gamma I^{3\times 3} & \text{otherwise,} \end{cases} \tag{4.23}$$

then we can control which sensors are used. By setting $R_{z_i} = \infty I^{3\times 3}$, we ensure that the Kalman gain for sensor $i \in [1, 2]$ goes to zero, preventing false data to be used in the heading estimation. The tuning parameter γ is chosen such that the Kalman gain reduces to zero in the case where no (accurate) measurements are available. Ideally, the value of this parameter is chosen to be infinity. In practice, however, γ is chosen very large ($\gamma = 10^{10}$, for example), as infinite elements might introduce calculation difficulties.

In the case of z_1, the magnetometer data is only used if the norm of the measured magnetic field deviates at most ξ_{mag} from the modelled earth's magnetic field. In the case of z_2, the camera data is only

used when a marker is actually detected and when the angle is not exactly equal to the previous sample to eliminate repeated image frames from influencing the yaw estimate.

Using the fusion algorithms discussed in this section, we can estimate the gravity and geo-magnetic field vector accurately in the body-fixed frame $\mathcal{B}$. Then we can use the orientation determination, discussed in Section 4.1 to construct an accurate estimation of the orientation, which is required for the controlling software. In the next section, we discuss a position and velocity observer that is proposed by Van den Eijnden (2017), in order to filter and estimate the position, based on low frequency camera measurements, and reconstruct the drone's velocity based solely on position measurements and the modeled dynamics.

4.4 Position and velocity observer

Whereas the position of the drone can be measured using either the down-facing internal camera, or an external camera, the sampling rate of these cameras is significantly lower than that of the other internal sensors. Also, it is not possible to directly measure the velocity of the drone. Hence, we require an observer that not only upsamples the position measurements, but also estimates the drone's velocity based on the position measurement and the model of the drone. Van den Eijnden (2017) has designed such observer, which we discuss below.

Consider the linear time-varying system (3.6), describing the error dynamics, and assume no velocity information is available. A new translational error system is defined as

$$\begin{aligned} \dot{e} &= Ae + Bu \\ y &= Ce \end{aligned}, \tag{4.24}$$

where $C = [I, 0]$. The output of the translational tracking error system is denoted $y \in \mathbb{R}^3$ and corresponds to the linear transformation of measured states to tracking errors. A full state proportional-integral observer of the Luenberger-type is proposed as

$$\begin{aligned} \dot{\hat{e}} &= A\hat{e} + Bu + L_p(y - \hat{y}) - By_I \\ \dot{y}_I &= -S(\omega_r)y_I - L_I(y - \hat{y}) \\ \hat{y} &= C\hat{e}, \end{aligned} \tag{4.25}$$

where $L_p = \begin{bmatrix} L_1 & L_2 \end{bmatrix}^\top \in \mathbb{R}^{6\times 3}$ and $L_I \in \mathbb{R}^{3\times 3}$. As the reference trajectory is known for any time t, the system matrix A is known for all t. Additionally, as the orientation is reconstructed from the sensor measurements, R is available, and since f can be directly manipulated, the control input u is known for all t. Choosing $L_1 = l_1 I$, $L_2 = l_2 I$, and $L_p = l_3 I$, where $l_1, l_2, l_3 > 0$ and $l_1 l_2 > l_3 > 0$ leads to a globally exponentially stable observer that estimates the error $\hat{e}$.

The position of a quad-rotor is mostly determined by using cameras, which generally are sampled at significantly lower frequencies compared to the IMU sensory equipment. In addition to this lower sample rate, delays could be introduced as image processing is demanding of computational resources. Position measurements, therefore, are not continuously available, such that the output of (4.24) needs to be reformulated as

$$y(t) = Ce(t_k), \tag{4.26}$$

where $t \in [t_k, t_{k+1})$, and t_k, $k \in \mathbb{N}$ denotes the time instant at which the position is sampled. Using these measurements directly in the observer (4.25) would result in non-smooth error estimates, in particular if

the sample rate is low or measurements are lost. Hence, a new observer that smooths the state estimates based on the expected system behaviour. The extended observer dynamics is defined as

$$\begin{aligned} \dot{\hat{e}} &= A\hat{e} + Bu + L_p\left(\varphi(t) - \hat{y}\right) - By_I \\ \dot{y}_I &= -S(\omega_r)y_I - L_I\left(y - \hat{y}\right) \\ \hat{y} &= C\hat{e}, \end{aligned} \tag{4.27}$$

where $\varphi(t)$ is an inter-sample predictor of the system output, which is continuous on the interval $t \in (t_k,\, t_{k+1})$ and is updated with the available output at time t_k. The output predictor is defined as a hybrid system of the form

$$\begin{cases} \dot{\varphi}(t) = -S\left(\omega_r\right)\hat{e}_\rho + \hat{e}_v \\ \varphi(t_k^+) = y(t_k). \end{cases} \tag{4.28}$$

By analysing the observer behaviour, it follows that the extimated states globally uniformly exponentially converge to the actual state, provided that the maximum sample interval $t_{k+1} - t_k$ remains within a certain bound. The derivation and proof of stability of this observer is given by Van den Eijnden (2017).

4.5 Summary

In this chapter we have discussed several methods that are used to determine the state of the quad-rotor based on sensor measurements. An orientation observer that is comprised of two parallel Kalman filters is used to fuse the information of several sensors in the IMU, such that accuracy is improved and drift reduced greatly, compared to not fusing the sensor data. Several changes have been made to the orientation observer by Zhang et al. (2016), as more sensors were available and sensor perturbations required tuning of the Kalman filters. Subsequently, we discussed how the altimeter and internal camera can be used in determination of the quad-rotor's position with respect to some marker and that, if a two-tone marker is used, also the heading can be extracted from the down-facing camera image. As only position information is available while the controller requires both position and velocity feedback, a velocity observer is used to estimate the position and velocity errors $\hat{e}_\rho$ and $\hat{e}_v$. Herein an inter-sample predictor is used that gives measurement estimates between samples based on the modelled dynamics.

In the next chapter we discuss the implementation of the controllers of Chapter 3 and the state reconstruction of this chapter. We give a schematic view of the control software structure, discuss simulations and experiments, and compare the results.

Chapter 5

Implementation

In previous chapters we discuss the quad-rotor model, the control laws we want to implement on the drone, and how the quad-rotor state is reconstructed from sensor data. In this chapter we combine the results for implementation. Firstly, the observers and controllers are implemented in a simulation environment, and subsequently embedded within the commercially available Parrot AR.Drone 2.0. This part of this work has been done in collaboration with colleague student S.J.A.M. van den Eijnden. First a simulation study is conducted to analyse the closed-loop system behaviour within a discrete-time environment, and in the presence of sensor noise and model uncertainties. Hereto, the models discussed in Chapter 2 are used. Since in this simulation, sensor and actuator models are included, it provides a realistic simulation environment.

5.1 Simulink structures

Prior to flying the quad-rotor, simulations have been conducted in Simulink to verify the controller's behaviour. A schematic representation of the Simulink model structure for simulation is given in Figure 5.1. Herein are V_{batt}, Gains, and T_{D} the input battery voltage, control and observer gains, and the desired trajectory identifier, respectively. During simulations, we included a 5% mass error between the mass in the model and control law, mainly to verify the usefulness of adding an integral action to the control law.

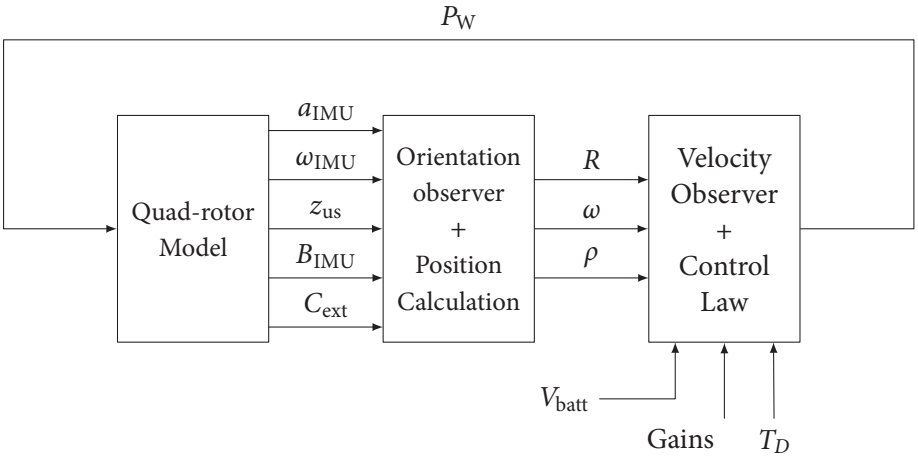

Figure 5.1: Schematic representation of the Simulink structure in simulation.

The block *Quad-rotor Model* contains the quad-rotor model as discussed in Section 2.1 and additionally

includes the sensor and motor models as discussed in Section 2.2. One alteration to the accelerometer sensor model has been made, however. As we assume zero drag, the accelerometer would only output accelerations in $z^{\mathcal{B}}$-direction and therefore cannot be used for attitude estimation. In reality, however, the accelerometer signal also includes components in the body-fixed x and y-directions. Hence, we alter the accelerometer model in simulations where drag is neglected to

$$\hat{a}_{\text{IMU}} = R^{\top} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \tag{5.1}$$

and use (2.11) for discretisation and noise as before. The accelerometer model now only represents the rotated gravitational vector. This assumption is valid only for small body accelerations and is sufficient for simulation purposes.

The *Orientation observer + Position Calculation* block contains the two-Kalman filter orientation observer that is discussed in Section 4.3 in order to fuse the information of multiple sensors into more accurate gravity and magnetic field vector estimates, such that the orientation can be estimated using the methods given in Section 4.1. Additionally, in this block, the position of the drone is calculated using the external camera input and the altimeter sensor signal.

The *Velocity Observer + Control Law* block includes the position and velocity observer, discussed in Section 4.4, and the controllers and trajectory generator as discussed in Chapter 3, receiving the position and velocity from the position and velocity observer and orientation from the orientation observer.

Once the simulation results are satisfactory, the control software can be implemented in the AR.Drone 2.0. The control software is designed in Simulink and subsequently compiled to C-code using Microstoft Visual C++ Professional (C) compiler. The resulting C-code is built to a binary file that can be run on the operating system of the drone. Manuals for using Simulink with the AR.Drone 2.0 are given in Appendix D.

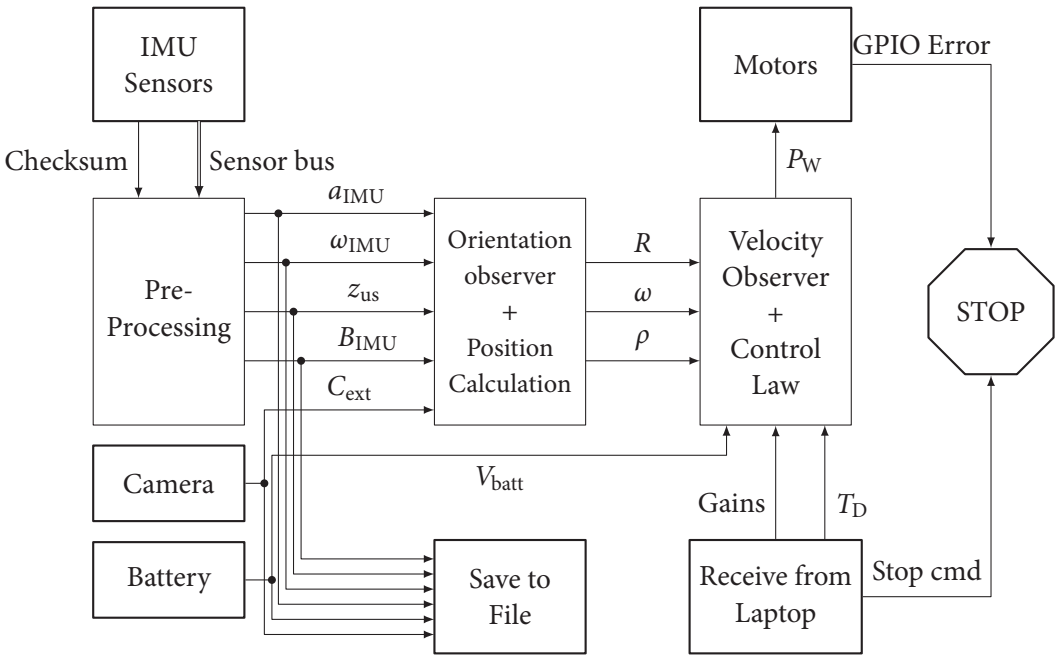

Figure 5.2: Schematic representation of the Simulink structure in the uploaded controlling software.

The structure of the software to be implemented is similar to the simulation structure, a schematic representation is given in Figure 5.2. Herein represent the blocks with a thick border the blocks giving access to the AR.Drone hardware.

The *Camera* block receives the camera output of the external camera from the laptop via a UDP connection. The *Battery* block outputs the drone's battery voltage V_{batt}, the *Receive from Laptop* receives the control and observer gains, the desired trajectory identifier T_{D} and a stop command from the laptop over a UDP connection. The *Motors* block receives the P_{W} signal from the controller and sends these to the motor drivers. Also, it outputs a *GPIO Error* signal, which is 1 if the rotors are blocked by an object, and 0 otherwise. This signal is used to terminate the control software in case such blockage occurs by the *STOP* block, which also terminates the software in case a stop command is received from the laptop. The *Save to File* block is used to save the sensor readings to a `.mat` file on the AR.Drone 2.0, that can be downloaded to the laptop for off-line analysis.

The *IMU Sensors* block outputs the all other sensor signals via the *Sensor bus* and provides a checksum signal, which is 1 if a measurement error occurs and 0 otherwise. Both signals are used in the *Pre-Processing* block where the individual sensor signals are extracted from the *Sensor bus* and erroneous measurements are blocked. Additionally, the *Pre-Processing* block also corrects the accelerometer and gyroscope measurements for temperature, as discussed in Appendix A.

5.2 Simulation results

In this section, several simulation studies are conducted and the results are presented accordingly. The main purpose of these simulations is as follows. Although it has been shown by Van den Eijnden (2017) that the controller is able to stabilise a quad-rotor in continuous time using the control laws discussed in Chapter 3, stability preservation is not guaranteed in discrete-time implementation. Additionally, external disturbances, saturations, measurement noise, and time-delays might significantly affect the closed-loop behaviour. Using the simulation model, discussed in Section 5.1, the performance of the control law in the presence of such effects can be analysed. In addition, since the integrating action in the position control has been implemented in a relatively unconventional manner, the effectiveness of this approach is illustrated through a numerical experiment.

For all simulations, it was chosen to let the quad-rotor track a three-dimensional elliptical trajectory, described by

$$\rho_{\text{r}} = \begin{bmatrix} \cos(t) \\ \sin(t) \\ 1.5 + \sin(t) \end{bmatrix}. \tag{5.2}$$

The reference velocities and accelerations, and corresponding forces and torques are determined by the reference generator, as discussed in Chapter 3. The controller and observer parameters for all simulations are given in Table 5.1 and the initial conditions for all simulations are given by

$$\rho_{\text{o}} = \begin{bmatrix} -0.5 & 0 & 0 \end{bmatrix}^{\mathsf{T}}, \qquad v_{\text{o}} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathsf{T}}, \qquad R_{\text{o}} = I, \qquad \omega_{\text{o}} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^{\mathsf{T}}.$$

Table 5.1: Control and observer parameters used in simulations.

Parameter	Value	Description
k_w	0.4	Integral gain
k_z	1	Integral gain
n	2	Saturation level
k_1	3	Position gain
k_2	2	Velocity gain
c_1	70	Orientation gain
c_2	30	Orientation gain
l_1	15	Observer gain
l_2	30	Observer gain
l_3	10	Observer gain

The simulation results are presented in figures 5.3 and 5.5. Note that Figure 5.3 additionally presents the results of a simulation in which the integral action of the controller is disabled (i.e. $k_w = k_z = 0$).

The effect of the integrating action immediately shows in the altitude z in figures 5.3 and 5.4. In case no integral action is added, the quad-rotor consistently flies at a higher altitude than desired. This results from the fact that since the expected mass is greater than the actual mass (due to, for example, measurement error), an excessive reference thrust is generated. As expected, the steady-state error is removed if the integral action is enabled. Recall that for position tracking it was desired to align the two normalised vectors $\hat{f} = R_{\mathrm{r}}^{\top} R B_f$ and $\hat{f}_{\mathrm{d}} = R_{\mathrm{d}} B_f$, which represent the actual and desired thrust vectors of the quad-rotor, expressed in the tracking reference frame $\mathcal{R}$.

In order to align these vectors, as a control objective, it was chosen to let $R_{\mathrm{r}}^{\top} R$ converge to R_{d}. In Figure 5.5 the orientation $R_{\mathrm{r}}^{\top} R$ and the desired orientation R_{d} are characterised in terms of roll, pitch, and yaw angles and expressed with respect to the tracking reference frame. From this figure, we observe that in the presence of non-exact measurements, the quad-rotor is able to track the desired orientation accurately. Furthermore, as expected, the desired orientation is small, as the quad-rotor is within a small vicinity to the desired trajectory. Therefore, large angle manoeuvres are not required to track the position.

Comparing the actual position with the reference position in Figure 5.3, we can observe that the quad-rotor is able to track the reference trajectory. The response, however, is somewhat delayed compared to the reference. We found that this is a direct result of the delayed position measurements. As a consequence to the delay, the estimated states converge to the measurements, rather than the actual states. The control action, therefore, compensates for previous positions such that an absolute periodic error remains. It is to be expected that in case of fast, aggressive reference trajectories, time-delays gain influence on the closed-loop performance. These observations motivate the idea to (partially) incorporate the time-delay in the observer. Moreover, one could examine the possibilities to reduce the time-delay, for example, by means of adopting a more efficient image processing algorithm, or more computational resources. The overall simulation results are as expected: in case the system is subject to small constant disturbances (e.g. erroneous mass estimations) and sampled, delayed, and disturbed measurements, the closed-loop stability is maintained and the corresponding tracking errors converge to some region near the origin.

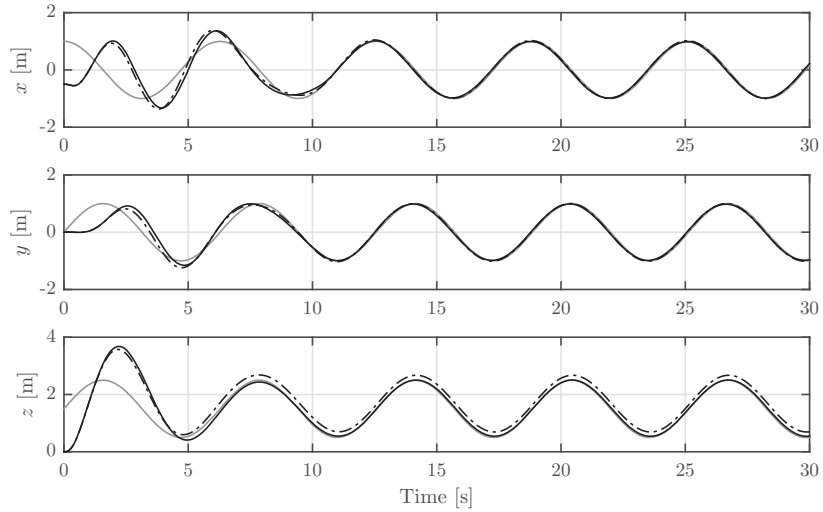

Figure 5.3: Position results ρ without integral control (dashed), with integral control (solid black), and reference position ρ_r (solid gray).

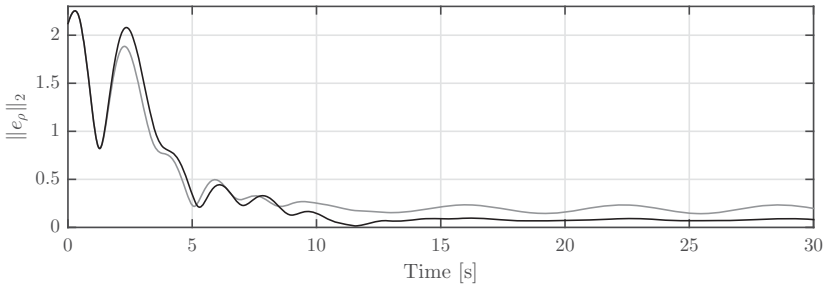

Figure 5.4: Position error norm $\|e_\rho\|_2$ without integral control (gray) and with integral control (black).

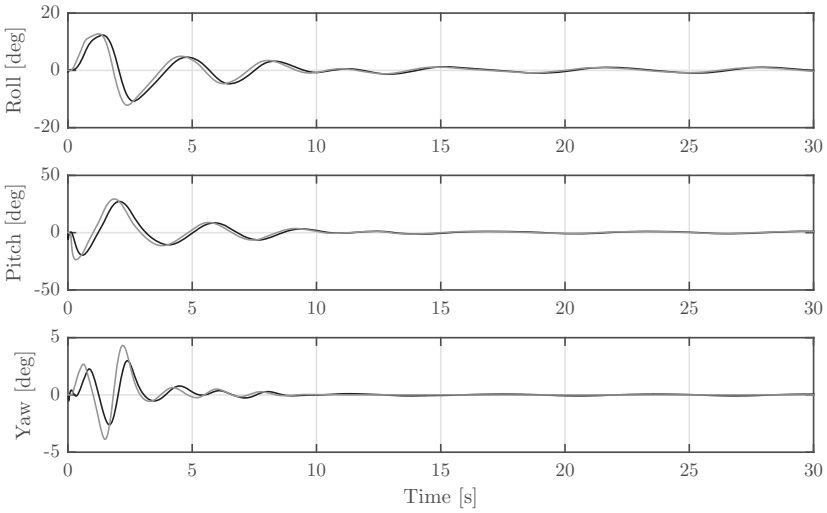

Figure 5.5: Orientation of the quad-rotor $R_r R$ (black) and the desired orientation R_d (gray), characterised in terms of roll, pitch, and yaw angles and expressed with respect to the tracking frame of reference.

5.3 MATLAB TOOLBOX IMPROVEMENTS

The toolbox used for experiments is based on the AR.Drone 2.0 toolbox by Daranlee and Slovak194 (n.d.). This toolbox provides several Simulink S-functions, based on C-code, that give access to the sensory equipment of the AR.Drone 2.0.

It was observed (Aert, 2016; Patel, 2016) that after a certain period of time, the magnetometer stops working and only outputs its last recorded signals. This issue only could be solved by restarting the entire control software. We solved this issue by altering the source code of the S-functions such that the IMU is reinitialised if the magnetometer outputs do not change over a period of 1 second. Note that this is a viable assumption as the magnetometer usually is subject to noise, hence outputting constantly changing signals.

In preparation for experiments, we noticed that it was hardly possible to use the internal down-facing camera on the drone. Sample rates of 1 fps required more computational resources than were available. This was solved by not running the Simulink in external mode with the drone, but in a standalone mode. Herein the Simulink model is compiled and uploaded to the drone, while no Simulink connection is required during execution of the control software. This significantly improved performance, resulting in the possibility to increase the sample rate of the camera block to 60 Hz. As the specified camera sample rate is 60 fps, we would expect image signals that vary at each sample due to noise. This, however, was not the case, the camera seemed to update only at 30 Hz. Also, a significant delay was observed. It seemed, however, not possible to increase the camera's sample rate from 30 to 60 fps in the source code for the camera S-function.

The C-code that compile into the Simulink camera S-function was altered, such that a frame buffer was reduced in size from 4 to 1 frame. This results in a significantly reduced delay between the actual camera sample and the camera signal in the control software. This frame-buffer stores several camera images and pushes the oldest image to the control software, introducing delay. Hence, reducing the frame-buffer length results in a smaller delay.

The alterations mentioned in this section improved the performance of the control software and added functionality in a sense that sensors that previously were unreliable or not usable now can be used reliably and at reasonable sample frequencies.

5.4 EXPERIMENTAL RESULTS

Experiments were conducted at the RoboCup soccer field at Eindhoven University of Technology. In order to conduct safe flights, this area is surrounded by netting, such that the drone remains within a confined area. A fixed right-handed inertial frame is defined at the centre of the soccer field and serves as the inertial frame of reference $\mathcal{I}$.

In this section, the results from various experimental flights are presented. In order to solve all problems regarding software and implementation, and to verify proper functioning of the sensors and hardware, we initially consider a simple hovering trajectory, where we desire the drone to hover at the centre of the field at an altitude of 1 metre. During the hovering experiments, similar control parameters were used as during simulations (Table 5.1), with exception of the integral gains, these were set at $k_w = 0.3$ and $k_z = 0.4$. The results are presented in figures 5.6 and 5.7.

From the position measurements, it can be seen that the AR.Drone 2.0 is able to reach the desired position through a stable manoeuvre. Overshoot is limited and the quad-rotor settles after approximately 20

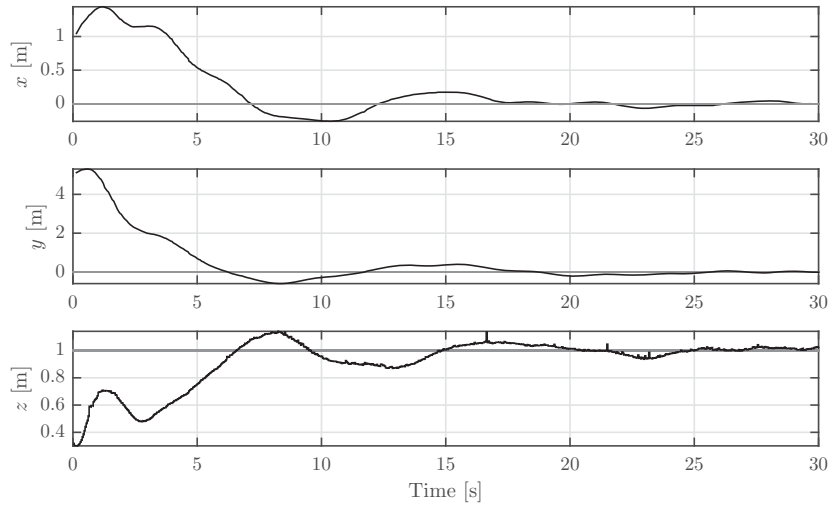

Figure 5.6: Measured position ρ (black) and reference position ρ_r (gray) over time.

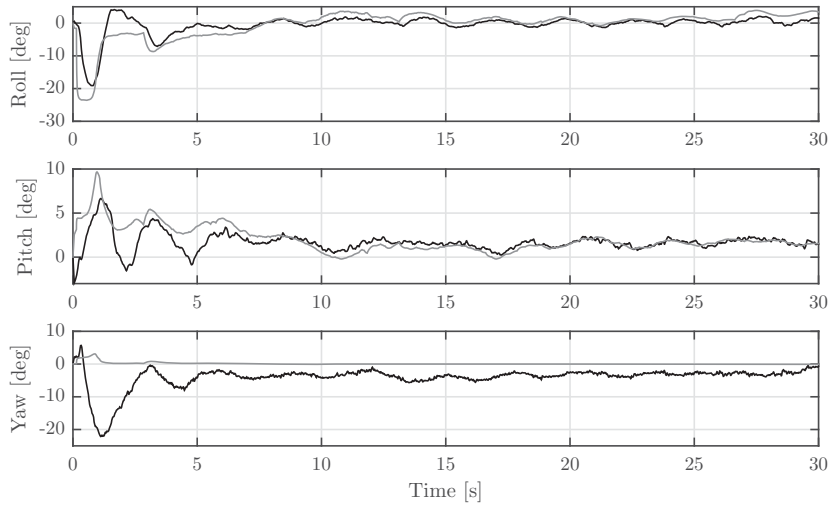

Figure 5.7: Reconstructed orientation $R_r^\top \hat{R}$ (black) and the desired orientation R_d (gray) characterised in terms of roll, pitch, and yaw angles and expressed with respect to the reference tracking frame $\mathcal{R}$.

seconds and slightly oscillates around the desired point. These oscillations result from small deviations in the quad-rotors estimated orientation and desired orientation, as can be seen in Figure 5.7. Note that in case of hovering $R_r = I$ such that the roll, pitch, and yaw angles can be considered with respect to the inertial frame of reference. Moreover, the desired orientation is calculated retroactively. It is remarkable that the desired and measured pitch angle of the quad-rotor remain positive, whereas the expected pitch would oscillate around zero. This difference might be caused by a misalignment of the sensors or propellers, such that a constant disturbing force acts in the horizontal direction. The position integral controller compensates for such constant disturbances, which results in a positive desired pitch angle, preventing positional drift.

Since the hovering trajectory is static, the control action and tracking error dynamics are time-invariant. In order to verify the effectiveness of the controller in case tracking of more aggressive, time-varying

manoeuvres is desired, an experiment was conducted. Herein, the desired trajectory was the ellipsoid as discussed in Section 5.2, where $\rho_r = \begin{bmatrix}\cos(t) & \sin(t) & 1.5 + \sin(t)\end{bmatrix}^\top$.

The control parameters used for these experiments were similar as presented in Table 5.1, with exception of the integral gains, which were set at $k_w = 0.3$ and $k_z = 1$. Note the difference with the choise for integral gains in the hovering experiment. A larger integral action appeared to have a more oscillatory response. The resulting flight trajectory is shown in figures 5.8 and 5.9.

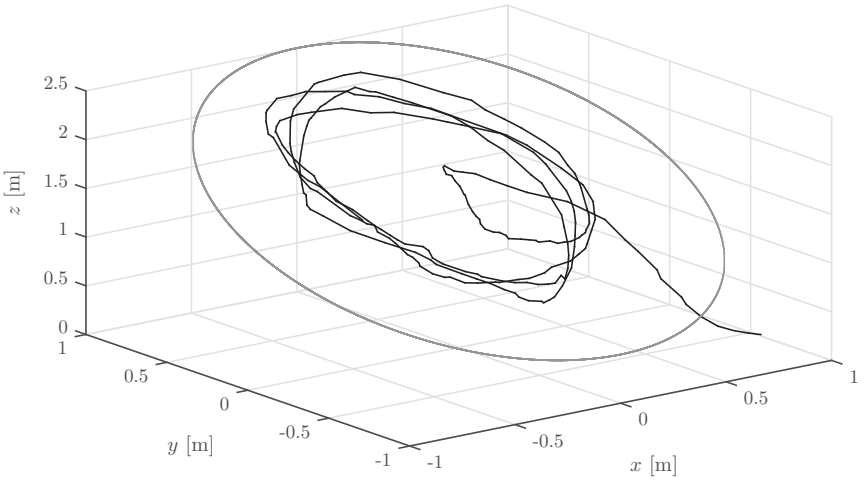

Figure 5.8: Spatial representation of the flight trajectory of the Parrot AR.Drone 2.0 (black) and the desired reference trajectory (gray).

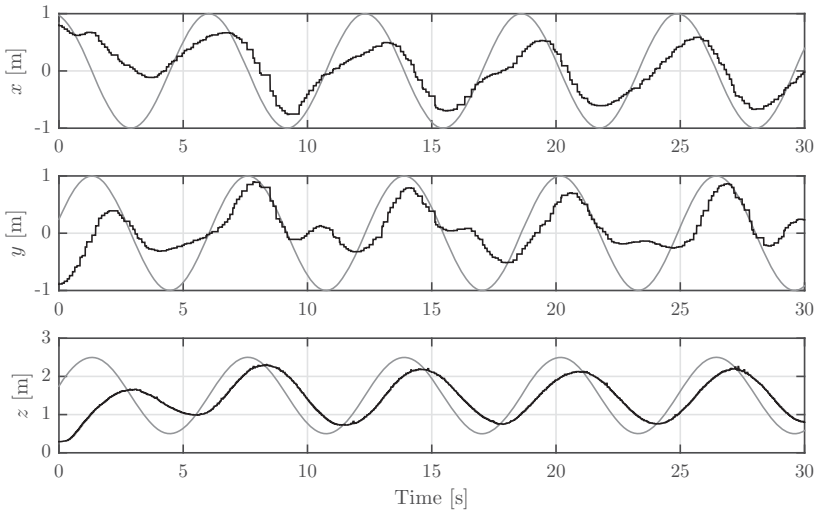

Figure 5.9: Measured position ρ (black) and the reference position ρ_r (gray) as a function of time.

Two direct observations can be made from the results in Figure 5.9. Firstly, we can conclude that the closed-loop system is able to perform a stable three-dimensional circular manoeuvre. Secondly, it can be seen that the quad-rotor remains within a significant distance from the reference trajectory, that is, the

amplitude of the flown trajectory is significantly smaller than desired. This intuitively suggests some form of non modelled damping is present in the system, which we discuss in the next section.

5.5 Non-modelled behaviour

As concluded in the previous section, some forms of non-modelled behaviour is present in the real-world system. One of these effects is expected to be air resistance. In order to examine this hypothesis, an additional set of simulations was conducted where a generalised, motion dependent damping force $F_{\mathrm{D}}(v)$ is applied to the dynamics as was modelled in Chapter 2. The damping coefficients are estimated from experimental data, in which the quad-rotor performed some manoeuvres (such as navigation to desired hovering location and the circle trajectory of previous section), and were found to be $c_{\mathrm{D},x} = 2.5$, $c_{\mathrm{D},y} = 1.3$, and $c_{\mathrm{D},z} = 1.3$. These values were found by conducting simulations, varying the damping coefficients, and minimise the difference between the measurement and simulation responses.

Note the large difference between $c_{\mathrm{D},x}$ and $c_{\mathrm{D},y}$, where intuitively one would expect similar values as the frontal area of the quad-rotor is not vastly different in these two directions. As the damping force is considered as a generalised drag force, other non-modelled dynamics might strongly affect motion in the x-direction (or y-direction). The density of the surrounding air is estimated at $\rho_{\mathrm{Air}} = 1.2\,\mathrm{kg/m^3}$.

In Figure 5.10 we give the results from a simulation with drag, without drag, and actual measurements. From this figure, we can clearly see that adding a generalised drag force to the system, the simulation resembles the actual measurements more closely. This result confirms the expectation: the closed-loop performance is largely affected by non-modelled, motion dependent dynamics. We recommend to incor-

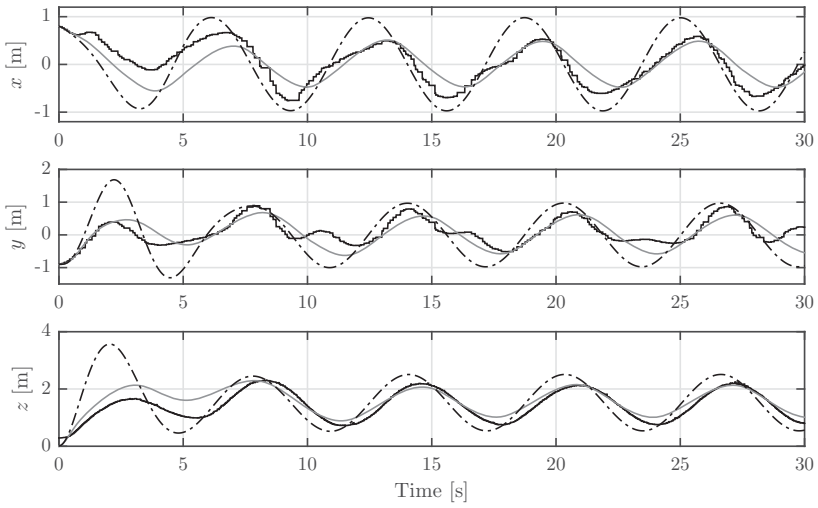

Figure 5.10: Comparison between the experimental results (solid black), simulation results without drag (dashed) and simulations with drag (solid gray).

porate these effects in the quad-rotor's model and expand the controller such that these damping effects can be compensated for. Hereto, an algorithm could be used to estimate the disturbance on-line, for example. A feed-forward structure can subsequently be added to the controller to compensate for these effects. We expect that such an approach can significantly increase the tracking performance of time-varying trajectories. Additionally, we remark that since the damping effects are motion dependent, the tracking performance is ultimately limited by the hardware. If the quad-rotor is desired to track a very fast trajectory, damping force might become large enough such that the rotors are incapable of generating

the required thrust or torque. Since the amplitude of the quad-rotor's flight trajectory during experiments approximately ranges between 0.5 and -0.5 metres, and the period time was 2π seconds, the velocity in x, y, and z-directions can be estimated as 0.5 metres per second. This would suggest that the damping force would be of order 0.4 Newton, which is well within the physical limits of the thrust.

So far, only damping effects in the translational motion of the system have been identified as a source for the affected tracking performance. The motion of the quad-rotor, however, is heavily dependent on its orientation. We therefore examine the orientation behaviour during flight in more detail. In Figure 5.11 the estimated orientation and the desired orientation necessary for position tracking are shown in terms of roll, pitch, and yaw angles.

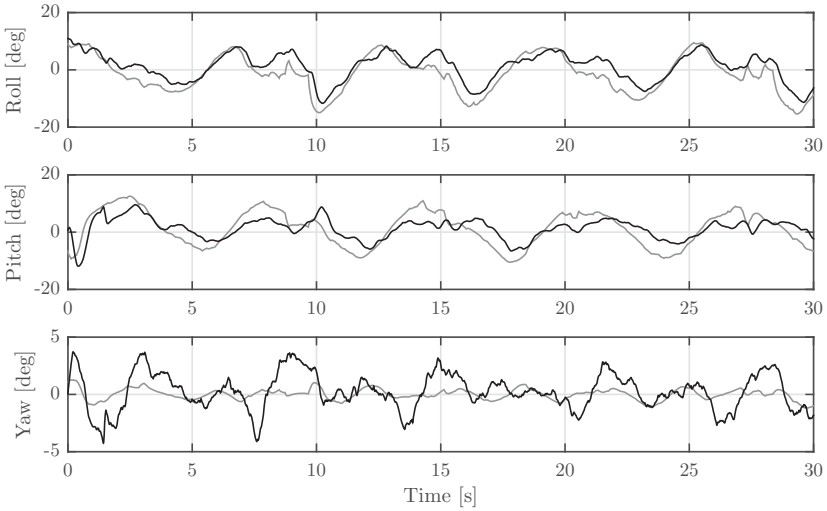

Figure 5.11: Estimated orientation $R_r^\top \hat{R}$ (black) and the desired orientation R_d (gray) characterised in terms of roll, pitch, and yaw angles and expressed with respect to the tracking reference frame $\mathcal{R}$.

It can be seen that the orientation follows the desired behaviour to some extent. The overall roll, pitch, and yaw angles, however, appear to be somewhat suppressed. A possible cause for this effect could be the damping in the orientation sub-dynamics. The protective hull of the AR.Drone 2.0 is constructed of lightweight expanded polystyrene, which is not very rigid. Also, its connection to the drone's frame is not rigid. Both sources of non-rigidity could be a source of damping, especially during high angular acceleration. An other source of rotational damping could be air resistance during rotational motion, causing an additional drag torque in the system.

Additionally, we note that, whereas motor-rotor measurements have been conducted for modelling purposes, these measurements have only been conducted in a stationary situation. During flight, however, motor performance might be different due to varying airflow, which could influence the performance of the AR.Drone 2.0 in time-varying trajectories.

Although including a generalised drag force into the model improves the accurate, there are still significant differences visible in Figure 5.10, particularly in y-direction. It can be seen that in the descending part of the quad-rotor's trajectory, the y-displacement somewhat remains constant; the quad-rotor only makes a downward motion. This might be caused by the fact that, during descend, the quad-rotor has to move through turbulent air generated by its rotors. Such turbulence affects the rotors thrust and torque,

making tracking significantly more difficult, hence losing positional accuracy.

5.6 Summary

In this chapter we implemented the controlling software in realistic simulations and experiments with the aim to verify the closed-loop behaviour in an environment where signals are not perfect and discretised and unconsidered dynamics. Hereto, the models of a quad-rotor and its sensors and actuators, discussed in Chapter 2, was used. Simulations were conducted, where the drone was to track a three-dimensional ellipsoid trajectory. The results show that the closed-loop system remained stable and sufficient tracking performance was achieved. Furthermore, we observed that delayed position measurements have some significant effect on the tracking performance as the response was somewhat delayed.

Subsequently, the controller was implemented in the AR.Drone 2.0. In order to validate the usefulness of the available software and hardware, first an experiment was conducted in which the drone was desired to hover at a fixed location. From these experiments it followed that the controlled quad-rotor was capable of hovering within a small vicinity of this desired location. A constant deviation in the pitch angle can be observed, which possibly is caused by misalignment of the rotors, compared to the body of the drone. In a second experiment, the drone was desired to track a time-varying, three-dimensional trajectory, similar to simulations. From the flight results we concluded that the drone was able to perform a stable elliptical manoeuvre, however, a significant deviation between the quad-rotor's position and the desired trajectory was observed. Several possible causes for the reduced tracking performance are discussed. Firstly, during fast motion, it appears that a translational damping effect is becoming more dominant in the system. This effect has been further analysed in simulations, where a generalised damping force was modelled as given in (2.5). We observe that the simulation results are strongly in accordance with the experimental results when this drag is introduced in the model. Secondly, damping in the orientation dynamics might cause loss of positional accuracy. This angular damping is possibly caused by the protective hull and its attachment mechanism to the body structure of the AR.Drone 2.0 as both are not very rigid, possibly introducing damping effects during high angular acceleration. Also, air resistance in rotational directions may introduce rotational damping, similar to the linear damping effects in (2.5). Lastly, several other effects, such as turbulence of the surrounding air, decreasing performance of the rotors in moving air, and damaged propellers or other hardware could contribute to the reduced performance.

For future research we recommend to add an algorithm to the controller that estimates the diminishing damping effects on-line, such that these can be compensated for by means of a feed-forward structure.

Chapter 6

Conclusions and recommendations

Drones are becoming more readily available and the extent of their possible applications seems to grow endlessly. This thesis is part of a larger research project that aims to provide an additional purpose to the already vast number of applications; sound imaging. The prime objective of this thesis was to autonomously fly a single AR.Drone 2.0, using control software designed in Simulink. In this chapter we present the conclusions and recommendations of this thesis.

6.1 Conclusions

In Chapter 2 we discussed several sub-models that comprise a more elaborate model that accurately describes the quad-rotor's behaviour, including the actuators and sensory equipment. The first sub-model is a model that describes the equations of motion for a translating and rotating body in 3D space. Subsequently, we derived the sensor and actuator models, based on measurements. This elaborate model receives PWM signals for motor actuation and outputs discrete, quantised sensor signals in a similar fashion to the actual AR.Drone 2.0. This elaborate model can be used for both control design and verification.

Subsequently, in Chapter 3, we discussed the control laws to be implemented on the AR.Drone 2.0. We initially discussed the cascaded structure in which the position and orientation controllers are ordered before discussing the specific control laws. Then, we indicated that the proposed position control by Choi and Ahn (2015) is not valid as its stability analysis fails due to poorly chosen conditions on two control gains. As this control law is not proven to be stable, a different set of controllers, designed by colleague student S.J.A.M. van den Eijnden (2017), is discussed and chosen for implementation. Additionally, we proposed a simple supervisory controller that aims to guarantee safe operation of the quad-rotor in several senses. Firstly, when one or more rotors are blocked when rotating, the supervisor instantly shuts down all rotors and terminates the controlling software, thereby preventing accidental restart of the rotors. Secondly, the supervisor disables the drone from initiating flight trajectories when the battery is low, and initiates a landing sequence when such event happens during flight. Lastly, the supervisor ensures the proper order of trajectories (lift-off before the desired trajectory, and landing before flight termination) to be executed during flight. Finally, in this chapter, we discussed a method for calculation of the required PWM signals going to the motors, corresponding to the desired force and torques by the position and orientation controllers.

Then, in Chapter 4, we discussed methods for state reconstruction based on the sensors available on the AR.Drone 2.0. Hereto, we started by calculating the drone's orientation, based on the gravity and geomagnetic field vectors in the body-fixed frame. Subsequently, we discussed how the internal camera of the drone can be used to obtain position information. Hereto, we used a two-LED marker that we can identify in the camera image. We gave a method to project the marker from the image to the real world by using the estimated orientation and altitude. From this projection, we can calculate the drone's position,

relative to this marker. As sensors are subject to noise and drift, we discussed an orientation observer that fuses several signals into a gravity and magnetic field vector. This reduces noise and improves accuracy, compared to solely using the accelerometer and magnetometer for estimation of the gravity and geomagnetic field vectors. Lastly, in this chapter, we discussed a position and velocity observer (Van den Eijnden, 2017) that uses low-frequency position measurements and high-frequency orientation estimates in addition to the drone model dynamics to estimate the drone's position in the inertial frame and the body-fixed velocity at high sample rates. Using the methods described in this section, we estimated the full state of the drone, required for the controller.

Finally, in Chapter 5, the separate elements discussed in previous chapters; the model, control, and state reconstruction were united in simulations and experiments. We started by giving an explanation of the Simulink structures that were used to arrange all components in order to run simulations, and later, experiments. Simulations where the drone was desired to track a ellipsoid trajectory in 3D space showed that the closed-loop behaviour of the quad-rotor system was stable, and the drone was able to track the reference trajectory. This verified the control law's stability in discrete-time measurements that are subject to noise. The simulation results additionally showed that time delay on the position measurement, lead directly to delayed position estimation, which affects the closed-loop tracking performance in a sense that the drone lagged behind the reference trajectory. The effects of time-delay on the position measurements are expected to more significantly influence the tracking performance for fast-changing trajectories.

As flight simulations were conducted successfully, we proceeded to implementing the control software on the actual drone and conducting experiments. Initially, we conducted an experiment where the drone was desired to hover at a fixed location. This illustrated the performance of the controlling software in the case of a time-invariant reference trajectory. Furthermore, we found that mass estimate errors and rotor performance errors due to, for example, bent propellers or shafts were compensated for by the integral action of the control law. In following experiments, the drone was desired to track the same ellipsoid trajectory as during simulations. From the results, we can conclude that the drone is capable of autonomously performing a stable three-dimensional, time-varying trajectory. The results, however, also show a large deviation from the reference trajectory in a sense that the drone was not able to achieve the desired amplitude of the desired ellipsoid. Non-modelled translational damping was identified as the main source of this effect. This suspicion was confirmed by extending the simulation model by including a velocity-dependent damping force, and compare the new simulation results to the measurements. Furthermore, we noticed that the effects of such damping is less significant in slow time-varying trajectories, such as hovering. Through analysis of the orientation response, damping in rotational directions was suspected, affecting orientation tracking performance. Since the translational motion of the drone is heavily dependent on its orientation, rotational damping could be an additional cause of reduced tracking performance. The attachment system of the hull to the body and drag of the indoor hull of the AR.Drone 2.0 ware indicated as possible sources of rotational damping. Furthermore, we remarked several other effects, such as air turbulence due to the drone's rotors and reduced rotor performance in flowing air, for possible sources that influence tracking performance.

6.2 RECOMMENDATIONS FOR FURTHER RESEARCH

One goal that has not been achieved in this thesis, is the use of the internal camera for position determination in practice, making it possible for the drone to fly without the need of an external data connection. We have been able to use the internal camera for isolating the two-LED marker and projection to the real world using the orientation estimation and altitude measurement. Successful implementation for position determination during flight experiments, however, was not achieved due to time limitations.

Additionally, in this thesis, all control software runs on the CPU of the AR.Drone 2.0. However, this drone is also equipped with a DSP specialised for image processing. It, therefore, might be interesting to use this DSP in conjunction with image processing algorithms for higher efficiency. It might lead to the possibility to execute more resource heavy, but higher performance image processing utilities for position estimation.

From flight experiments it appeared that the model, as discussed in Chapter 2, with the exclusion of translational damping was insufficient for fast time-varying trajectories. In order to improve the accuracy of the model, we recommend to further investigate the non-modelled dynamics such as translational and rotational damping and include these effects in the model. Additionally, these effects can be used to improve the tracking performance by extending the controller by means of a feed-forward structure, for example.

During experiments, the external camera was used for position determination of the drone. During large angle manoeuvres, the LEDs fixed to the top of the drone occasionally were not clearly identifiable by this camera. Hence, during such manoeuvres no position feedback was received, reducing tracking performance. This could be improved by implementing additional external measures for position measurement, or the use of both internal camera's for accurate position measurement. Also, the position measurements of the external camera are subject to time delay as the image processing algorithm used to isolate the drone in the external camera image runs directly in Matlab. The image processing algorithm is relatively resource demanding as it involves large matrix operations (1920×1080×3 elements) at ideally 30 Hz. While Matlab code is easy to comprehend, it is not as efficient as other programming languages, such as C. A more efficient language might not only reduce the delay in the position measurement, also the sample rate might improve.

Bibliography

Aert, B.C.M.V. (2016). *Control and coordination algorithms for autonomous multi-agent quadrotor systems* [CST report, No. 2016.070] (Master's thesis, Eindhoven University of Technology, Control Systems Technology Group, Department of Mechanical Engineering).

Apvrille, L., Tanzi, T., & Dugelay, J.-L. (2014, August). Autonomous drones for assisting rescue services within the context of natural disasters. In *2014 XXXIth URSI general assembly and scientific symposium (URSI GASS)* (pp. 1–4). Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/ursigass.2014.6929384

Atkinson, K. (1989, January 11). *An introduction to numerical analysis*. New York: JOHN WILEY & SONS INC.

Choi, Y.-C. & Ahn, H.-S. (2015, June). Nonlinear control of quadrotor for point tracking: actual implementation and experimental tests. *IEEE-ASME Transactions on Mechatronics*, *20*(3), 1179–1192. doi:10.1109/tmech.2014.2329945

Daranlee & Slovak194. (n.d.). Simulink ar drone target. Retrieved September 1, 2015, from https://github.com/darenlee/SimulinkARDroneTarget

Jardin, M.R. & Mueller, E.R. (2009, May). Optimized measurements of unmanned-air-vehicle mass moment of inertia with a bifilar pendulum. *Journal of Aircraft*, *46*(3), 763–775. doi:10.2514/1.34015

Jeurgens, N.L.M. (2016). *Implementing a Simulink controller in an AR.Drone 2.0* [DC report, No 2016.005] (Internship report, Eindhoven University of Technology, Dynamics and Control Group, Department of Mechanical Engineering).

Li, Q. (2014, August). *Grey-box system identification of a quadrotor unmanned aerial vehicle* (Master's thesis, Delft University of Technology, Faculty of Mechanical, Maritime and Materials Engineering (3mE)).

Li, W. & Wang, J. (2012, July). Effective adaptive Kalman filter for MEMS-IMU/magnetometers integrated attitude and heading reference systems. *Journal of Navigation*, *66*(01), 99–113. doi:10.1017/s0373463312000331

Oczipka, M., Bemmann, J., Piezonka, H., Munkabayar, J., Ahrens, B., Achtelik, M., & Lehmann, F. (2009, September). Small drones for geo-archaeology in the steppe: locating and documenting the archaeological heritage of the orkhon valley in Mongolia. In U. Michel & D.L. Civco (Eds.), *Remote sensing for environmental monitoring, GIS applications, and geology IX*. SPIE-Intl Soc Optical Eng. doi:10.1117/12.830404

Ozyagcilar, T. (2013). *Calibrating an eCompass in the presence of hard and soft-iron interference*. Application Note. Freescale Semiconductor. Retrieved from http://www.nxp.com/files/sensors/doc/app_note/AN4246.pdf

Patel, J. (2016). *Tracking a moving object using multiple drones* [CST report, No. 2016.093] (Master's thesis, Eindhoven University of Technology, Control Systems Technology Group, Department of Mechanical Engineering).

Pleban, J.-S., Band, R., & Creutzburg, R. (2014, February). Hacking and securing the AR.drone 2.0 quadcopter: investigations for improving the security of a toy. In R. Creutzburg & D. Akopian (Eds.),

Mobile devices and multimedia: enabling technologies, algorithms, and applications 2014 (90300L). Proceedings of SPIE. SPIE-Intl Soc Optical Eng. doi:10.1117/12.2044868

Sabatini, A.M. (2011, September). Kalman-filter-based orientation determination using inertial/magnetic sensors: observability analysis and performance evaluation. *Sensors*, *11*(12), 9182–9206. doi:10.3390/s111009182

Scholte, R. (2008). *Fourier based high-resolution near-field sound imaging* (Doctoral dissertation, Eindhoven University of Technology). doi:10.6100/IR639528

Sorama. (2015). Visualising sound and vibrations. Retrieved December 21, 2016, from https://www.sorama.eu

Tripicchio, P., Satler, M., Dabisias, G., Ruffaldi, E., & Avizzano, C. (2015, July). Towards smart farming and sustainable agriculture with drones. In *Intelligent environments (ie), 2015 international conference on* (pp. 140–143). Institute of Electrical and Electronics Engineers (IEEE). doi:10.1109/IE.2015.29

van den Eijnden, S.J.A.M. (2017). *Cascade Based Tracking Control of Quadrotors* [DC report, No 2017.012] (Master's thesis, Eindhoven University of Technology, Dynamics and Control Group, Department of Mechanical Engineering).

Vlasenko, D. (n.d.). BusyBox: The Swiss Army Knife of Embedded Linux. Retrieved December 20, 2016, from https://www.busybox.net/about.html

Zhang, S., Yu, S., Liu, C., Yuan, X., & Liu, S. (2016, February). A dual-linear Kalman filter for real-time orientation determination system using low-cost MEMS sensors. *Sensors*, *16*(2), 264. doi:10.3390/s16020264

Appendices

Appendix A

AR.Drone 2.0 hardware

The AR.Drone 2.0 is a quad-rotor, with an on-board computer running BusyBox, a small executable that combines many tiny versions of common UNIX utilities and provides a fairly complete environment for embedded systems (Vlasenko, n.d.). The AR.Drone 2.0 creates its own access point to which a laptop can be connected for interaction. In order to determine its position and orientation, the AR.Drone is equipped with a variety of sensors (Pleban, Band, & Creutzburg, 2014). It contains the following sensors:

- ultrasonic distance sensor (range: 6m)
- 3-axis accelerometer (±50 mg precision)
- 3-axis gyroscope (±2000 °/s precision)
- 3-axis magnetometer (6 ° precision)
- barometer (10 Pa precision)
- front-facing camera (1280×720 px @ 30 fps, fov: 90 ° diagonal)
- bottom-facing camera (320×240 px @ 60 fps, fov: 64 ° diagonal).

In order to actuate the quad-rotor, only four actuators are present: the rotors, each providing thrust in body-fixed z-direction and torque (either positive or negative, depending on the rotor) around the body-fixed z-axis. The front-facing camera and the barometer are not used in this project and are therefore not discussed further.

A.1 AR.Drone 2.0 system parameters

Table A.1: Parameters of the Parrot AR.Drone 2.0 model.

Parameter	Indoor hull	Outdoor hull	Unit
m	0.445	0.475	[kg]
l	0.125	0.125	[m]
I_x	$2.7{\cdot}10^{-3}$	$2.2{\cdot}10^{-3}$	[kg m^2]
I_y	$2.9{\cdot}10^{-3}$	$2.5{\cdot}10^{-3}$	[kg m^2]
I_z	$5.3{\cdot}10^{-3}$	$4.5{\cdot}10^{-3}$	[kg m^2]
J_r	$2.03{\cdot}10^{-5}$	$2.03{\cdot}10^{-5}$	[kg m^2]
$C_{D,x}$	2.5	N/A	[-]
$C_{D,y}$	1.3	N/A	[-]
$C_{D,z}$	1.3	N/A	[-]

A.2 Ultrasonic altimeter

The ultrasonic altimeter is used to determine the drone's altitude. This is a down-facing, body-fixed sensor that emits and receives ultrasonic pulses at a rate of 25Hz. The time between transmission and reception of such pulse is used to determine the distance between the sensor (drone) and an object (floor). The output of the sensor, however,

is not the altitude in metres directly, hence a conversion is required.

Measurements have been conducted where the drone's orientation was parallel to the floor surface in order to obtain accurate measurements. The separate measurements are given in Figure A.1. This figure also shows a linear relation

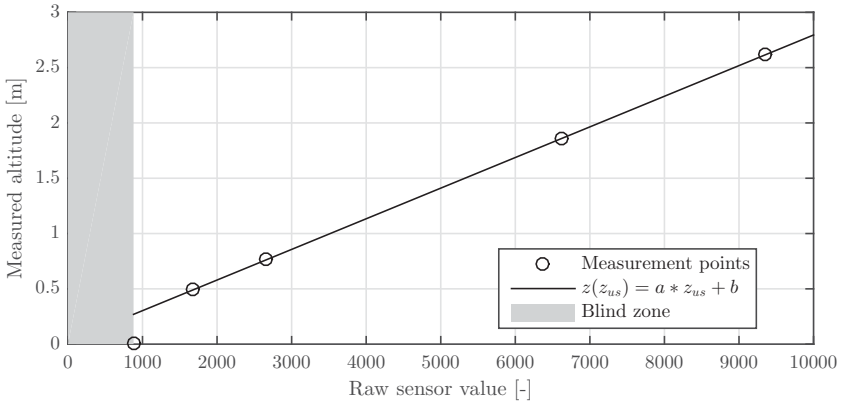

Figure A.1: The black circles show altitude measurements at several altitudes with the corresponding raw sensor values, the black solid line shows a linear relation, fitted through the measurement data, and the grey area shows a blind zone: Where the altimeter outputs constant values, independent of the altitude.

that is fitted through the measurement points, and a blind zone where the altimeter outputs constant values, regardless of the distance to an object. As can be seen, apart from the measurement at altitude $z = 0\text{m}$, all measurements lie on the line $\tilde{z}(z_{\text{us}}) = az_{\text{us}} + b$, where a, b are constants and z_{US} is the raw sensor value. The found values for a and b are:

$$a = 2.8 \cdot 10^{-4}, \qquad b = 2.6 \cdot 10^{-2}.$$

In addition to the altitude, the orientation w.r.t. the floor significantly influences the sensor readings. We assume

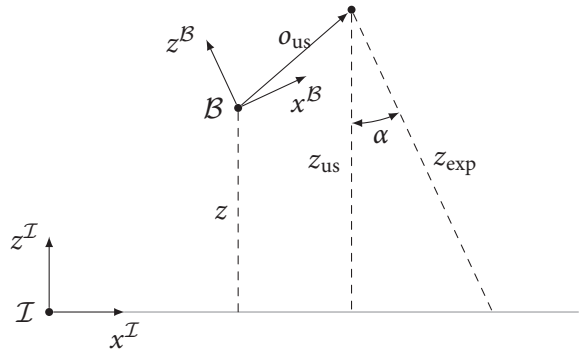

Figure A.2: Visual representation of the altimeter measurement. The altimeter is fixed to the body-fixed coordinate frame $\mathcal{B}$ and is aimed in the negative $z^{\mathcal{B}}$-direction. The expected measured distance and the true altitude are given as z_{exp} and z, respectively. The angle between the earth- and body-fixed vertical axes, $z^{\mathcal{I}}$ and $z^{\mathcal{B}}$, is defined as α.

that the drone is positioned above a horizontal floor with no obstructions that interfere with the altitude measurements. Measurements have been conducted by fixing the drone's location and varying the angle α between the earth-bound normal vector and the body-fixed z-vector:

$$\alpha = \arccos\left(\frac{\left(z^{\mathcal{I}}\right)^{\top} R z^{\mathcal{I}}}{|z^{\mathcal{I}}|^2}\right), \tag{A.1}$$

where $z^{\mathcal{I}}$ is the normal vector of the floor and R is the rotation matrix describing the rotation from the earth-fixed inertial frame $\mathcal{I}$ to the body-fixed coordinate frame $\mathcal{B}$. The expected sensor reading, if the sensor acutally measures the distance to a point precisely in front of this sensor, is expressed in (A.2).

$$z_{\text{exp}} = \frac{z_{\text{us}}}{\cos\alpha} = \frac{z + \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R^{\top} \vec{o}_{\text{us}}}{\cos\alpha}, \tag{A.2}$$

where z_{exp} is the expected altitude measurement and z is the true altitude, the vector o_{us} is the sensor offset from the body-fixed frame. A visual representation of the coordinate systems and expected altitude measurement is given in Figure A.2.

The measured altitude, however, does not correspond with the expected measurement z_{exp}, as can be seen in Figure A.3. This figure shows the expected altitude measurement and the actual altitude measurement in the top graph,

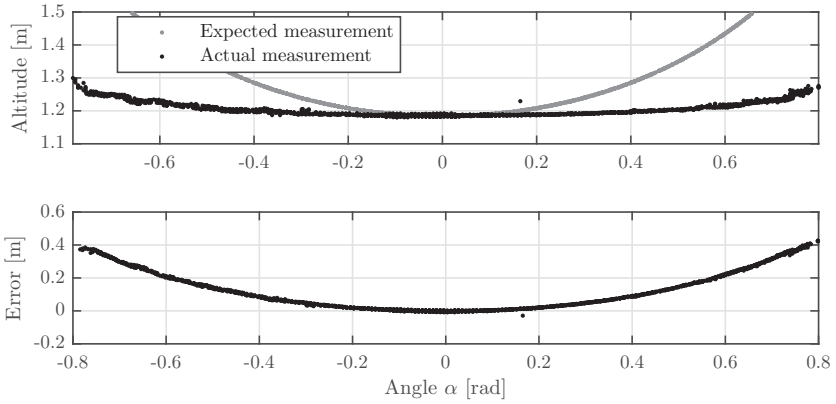

Figure A.3: Actual sensor measurements compared to what would be expected if the sensor measured the distance of a point that it was aimed at under varying angles (top), and the difference between the expected and actual measurements (bottom).

the bottom graph shows the error $e_z = z_{\text{exp}} - z$. Fitting a function through the error enables us to relate the measured altitude $\hat{z}$ to the angle α and the true altitude z. A function that seems to fit the error data is given by

$$e_z = a_1(z)\left(\cos\alpha - 1\right) + a_2(z)\left(\cos\alpha - 1\right)^2, \tag{A.3}$$

where the coefficients $a_1(z)$ and $a_2(z)$ are altitude dependent. An estimate for the measured altitude $\hat{z}$ then can be expressed as:

$$\begin{aligned} \hat{z} &= z_{\text{exp}} - e_z \\ &= \frac{z + \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R^{\top} o_{\text{us}}}{\cos\alpha} - a_1(z)\left(\cos\alpha - 1\right) - a_2(z)\left(\cos\alpha - 1\right)^2. \end{aligned} \tag{A.4}$$

In Figure A.4 the values for the coefficients $a_1(z)$ and $a_2(z)$ are shown for measurements at different altitudes. The coefficients $a_1(z)$ and $a_2(z)$ seem a linear function of the altitude, as expressed in (A.5) and (A.6).

$$a_1(z) = c_{11} z + c_{12} \tag{A.5}$$

$$a_2(z) = c_{21} z + c_{22}, \tag{A.6}$$

with

$$c_{11} = -0.94, \qquad c_{12} = 0.057, \qquad c_{21} = 1.5, \qquad c_{22} = -0.69.$$

Using equations (A.1), (A.4), (A.5), and (A.6), and bringing the altitude terms to the left side of the equation gives us a relation for the altitude as a function of the measured altitude $\hat{z}$ and the orientation matrix R:

$$z(\hat{z}, R) = \frac{\hat{z} + c_{12}\left(\cos\alpha - 1\right) + c_{22}\left(\cos\alpha - 1\right)^2}{\frac{1}{\cos\alpha} - c_{11}\left(\cos\alpha - 1\right) - c_{21}\left(\cos\alpha - 1\right)^2} - \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} R^{\top} o_{\text{us}}. \tag{A.7}$$

If we apply (A.7) to the measurement data of Figure A.3, we can see a significant improvement in the resulting altitude as is demonstrated in Figure A.5, where the grey and black dots represent the uncorrected and corrected altitude measurements, respectively.

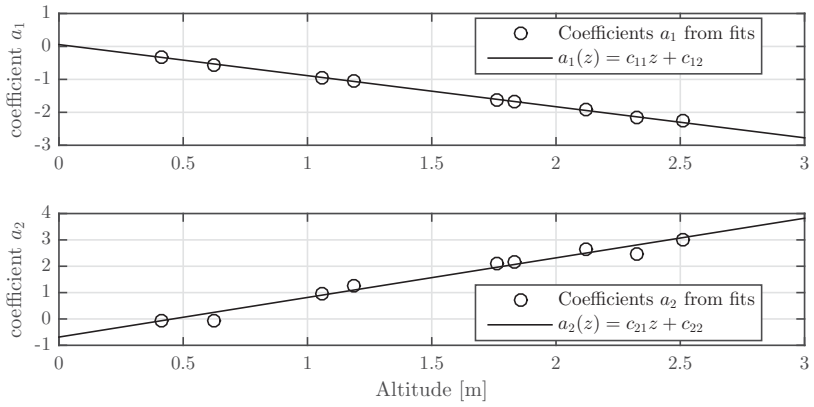

Figure A.4: Dependency of coefficients $a_1(z)$ and $a_2(z)$ on the altitude (top and bottom respectively). The circles are the individual coefficient values at each measurement, the solid black line represent functions fitted through these points.

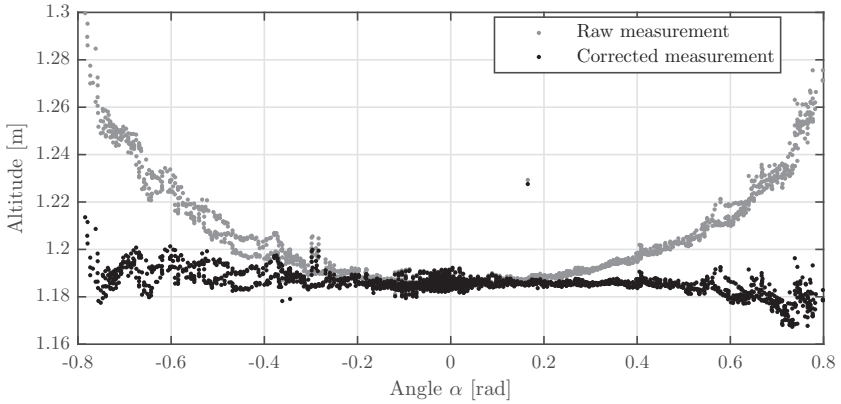

Figure A.5: A comparison between uncorrected (grey) and corrected (black) altitude measurements.

A.2.1 Filter false readings

Since the floor of the testing facility is covered in fabric, which absorbs (part of) the ultrasonic pulses transmitted by the altimeter, occasionally the altimeter returns an altitude of zero. In order to compensate, such that the position observer receives fewer and smaller erroneous altitude measurements, we implement a simple filter that uses the accelerometer measurement to estimate the altitude.

Firstly, the vertical acceleration of the quad-rotor is determined by:

$$a_z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} (R a_{\text{IMU}} - g), \tag{A.8}$$

where R is the rotation matrix, a_{IMU} is the measured body-fixed acceleration by the accelerometer (this includes gravity) and g is the gravity vector. We introduce the state vector $u = \begin{bmatrix} z & v_z & a_z \end{bmatrix}^\top$, where z is the altitude, v_z is the vertical velocity, and a_z is the vertical acceleration, all in the earth-fixed inertial frame. Then we can estimate the current altitude based on the accelerometer measurement if no valid measurement is available. The filter algorithm is given in Algorithm A.1. A comparison between the raw, unfiltered altimeter measurement and filtered measurement is given in Figure A.6. As can be seen, the measurement peaks are eliminated completely. Note that it is not necessarily useful to implement a different, more accurate filter on the altimeter measurement, since this signal is used only in the position and velocity observer.

Algorithm A.1 Altitude filtering algorithm

1: For each time step k:
2: $u_k = \begin{bmatrix} 1 & t_s & 0 \\ 0 & 1 & t_s \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_{k-1}(1) \\ u_{k-1}(2) \\ a_{z,k} \end{bmatrix}$;
3: $n = n + 1$;
4: **if** new z-measurement & $|u_k(1) - z_k| \leq \text{Tol}$ **then**
5: $v_z = \frac{z_k - z_{k-n}}{n t_s}$;
6: $u_k = \begin{bmatrix} z_k & v_z & a_{z,k} \end{bmatrix}^\top$
7: $n = 0$
8: **end if**

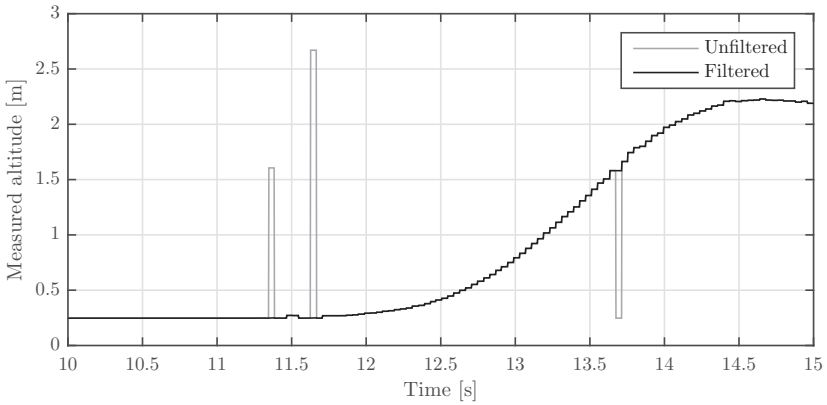

Figure A.6: Comparison between the raw, unfiltered altitude measurement (grey) and the filtered altitude measurement (black).

A.3 Accelerometer

As described by Jeurgens (2016), the accelerometer outputs four signals: three accelerations in the body-fixed frame and a temperature signal. All of these signals are unsigned 16 bit integers in Simulink. They also state that the raw sensor signal not only depends on the body acceleration and orientation, but also on temperature of the sensor package. They have conducted measurements in order to identify the relation between the sensor acceleration output and temperature. They, however, suggest to re-conduct these measurements using a larger temperature range in order to better model the sensor behaviour.

We have conducted new measurements using the same AR.Drone with a greater ambient temperature range, ranging from ~10 °C to ~60 °C. Note that the temperature range is not very precisely known. This is, however, not required since only the relation between temperature counts and acceleration counts is to be determined.

Figure A.7 shows the moving average acceleration counts versus the temperature counts (grey). Both signals were averaged using the Matlab function `smooth(signal,400)` in order to ease visualisation of the relation. Figure A.7 also shows a function that was fitted through the original (non-smoothed) measurement data of the acceleration and temperature signals. The chosen function for this particular relation is

$$\tilde{a}_{\text{IMU}}(T_{\text{acc}}) = a_2 T_{acc}^2 + a_1 T_{\text{acc}} + a_0 + \text{diag}\left(A_{\text{acc}} \frac{\text{sgn}(T_{\text{acc}} - S_{\text{acc}}^\top) + 1}{2}\right). \tag{A.9}$$

$$a_2 = \begin{bmatrix} 3.7 \cdot 10^{-4} \\ 3.6 \cdot 10^{-5} \\ 2.3 \cdot 10^{-4} \end{bmatrix} \quad a_1 = \begin{bmatrix} -9.2 \cdot 10^{-2} \\ 3.7 \cdot 10^{-3} \\ -2.3 \cdot 10^{-1} \end{bmatrix} \quad a_0 = \begin{bmatrix} 5.2 \cdot 10^{2} \\ 5.1 \cdot 10^{2} \\ 4.5 \cdot 10^{2} \end{bmatrix}$$

$$A_{\text{acc}} = \begin{bmatrix} 0.16 & 1.7 & -2.8 & 1.4 & 1.1 \\ 0 & 0 & 0 & 0 & -1.4 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$S_{\text{acc}} = \begin{bmatrix} 105 & 112 & 127 & 144 & 175 \\ 0 & 0 & 0 & 0 & 127 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Relations for the other axes are similar, but may include more or fewer sgn(.) functions. Figure A.8 shows the

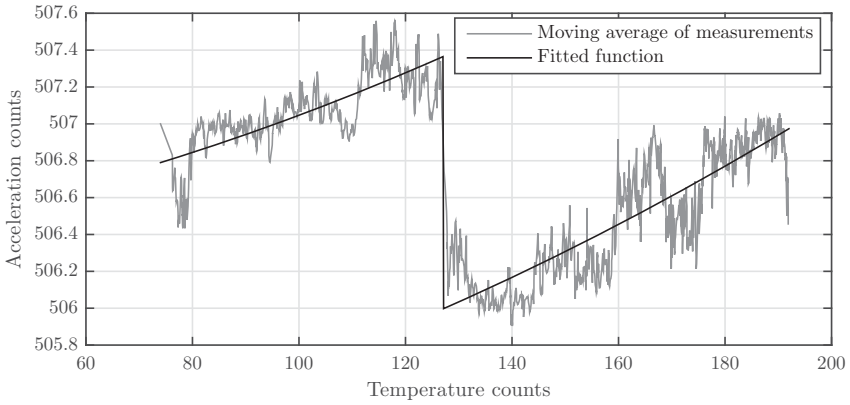

Figure A.7: Moving average of the raw measurement data (dash-dot) where the acceleration counts are plotted against the temperature counts, and the fitted function (solid) that estimates the relation between temperature and acceleration counts.

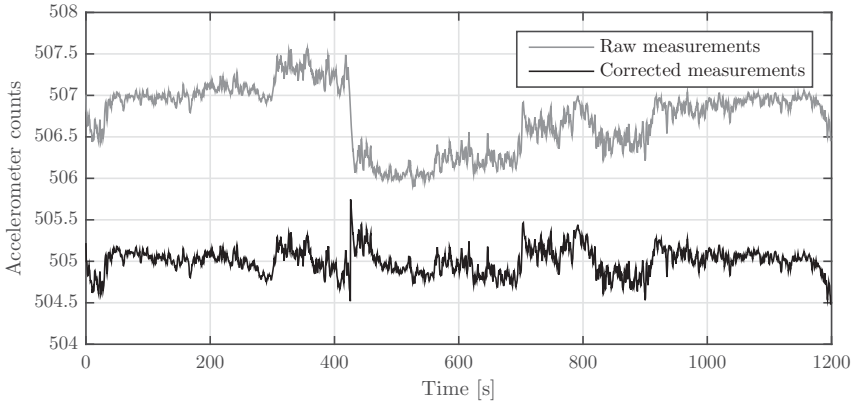

Figure A.8: Moving average of the raw measurement data (grey) over time and the corrected moving average (black), shifted vertically for visual purposes.

moving average of the raw measurement data and the corrected measurement that is shifted vertically for visual purposes. As can be seen, the compensated signal is significantly more level, compared to the raw, uncorrected acceleration measurement.

After temperature correction, the body-fixed acceleration can be calculated by:

$$\dot{v} = \frac{a_{\text{IMU}} - \tilde{a}_{\text{IMU}}(T_{\text{acc}})}{512} g, \tag{A.10}$$

where g is the standard gravity ($g = 9.80665\ \text{m/s}^2$). Note that, whereas a sensitivity of 512 counts per g might suggest a resolution of 1/512 g, the accelerometer actually outputs steps of four, resulting in a lower resolution of 1/128 g.

A.4 Gyroscope

Similarly to the accelerometer, the gyroscope outputs four signals: three accelerations in the body-fixed frame and a temperature signal. All of these signals are unsigned 16 bit integers in Simulink. Jeurgens also states that the raw sensor signal depends not only on the body rotational velocity, but also on the sensor package temperature. They have conducted measurements in order to identify the relation between the sensor angular velocity output and temperature. They, however, suggest to re-conduct these measurements using a larger temperature range in order to better model the sensor behaviour.

We have conducted new measurements using the same AR.Drone with a greater ambient temperature range, ranging from ~10 °C to ~60 °C. Note that the temperature range is not very precisely known. This is, however, not required since only the relation between temperature counts and acceleration counts is to be determined.

Figure A.9 shows the angular velocity counts versus the temperature counts (grey) and a fifth degree polynomial that has been fitted through the measurement data, in order to relate the angular velocity counts to the sensor temperature.

$$\tilde{\omega}_{\text{IMU}}(T_{\text{gyro}}) = a_5 T_{\text{gyro}}^5 + a_4 T_{\text{gyro}}^4 + a_3 T_{\text{gyro}}^3 + a_2 T_{\text{gyro}}^2 + a_1 T_{\text{gyro}} + a_0, \tag{A.11}$$

where

$$a_5 = \begin{bmatrix} -4.7 \cdot 10^{-19} \\ 2.0 \cdot 10^{-20} \\ -5.8 \cdot 10^{-21} \end{bmatrix} \quad a_4 = \begin{bmatrix} 1.2 \cdot 10^{-13} \\ -5.2 \cdot 10^{-15} \\ 1.5 \cdot 10^{-15} \end{bmatrix} \quad a_3 = \begin{bmatrix} -1.2 \cdot 10^{-8} \\ 5.4 \cdot 10^{-10} \\ -1.5 \cdot 10^{-10} \end{bmatrix}$$

$$a_2 = \begin{bmatrix} 6.0 \cdot 10^{-4} \\ -2.7 \cdot 10^{-5} \\ 7.5 \cdot 10^{-6} \end{bmatrix} \quad a_1 = \begin{bmatrix} -1.5 \cdot 10^{1} \\ 7.0 \cdot 10^{-1} \\ -1.9 \cdot 10^{-1} \end{bmatrix} \quad a_0 = \begin{bmatrix} 1.5 \cdot 10^{5} \\ -7.2 \cdot 10^{3} \\ 2.1 \cdot 10^{3} \end{bmatrix}.$$

Note that (A.11) is only valid in the temperature range ($4.4{\cdot}10^4$—$6.2{\cdot}10^4$) [counts], as only in this temperature range

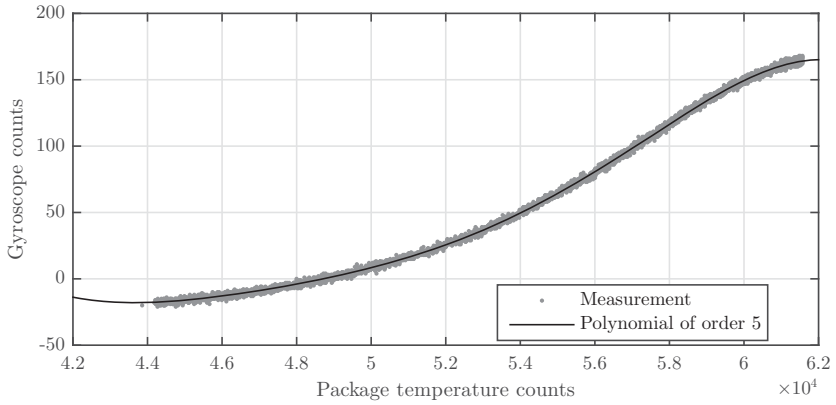

Figure A.9: Raw measurement data (grey) where the angular velocity counts are plotted against the temperature counts, and the fitted polynomial that estimates the relation between temperature and angular velocity counts.

measurements have been conducted. Using (A.11) together with the property that the gyroscope signal can be converted to degrees per second by multiplying with $4000/2^{16}$, the angular velocity in radians per second can be calculated using (A.12)

$$\omega = \frac{\pi}{180} \frac{4000\left(\hat{\omega}_{\text{IMU}} - \tilde{\omega}_{\text{IMU}}(T_{\text{gyro}})\right)}{2^{16}}, \tag{A.12}$$

where $\hat{\omega}_{\text{IMU}}$ is the raw gyroscope measurement in counts. Figure A.10 shows the raw measurement data (grey) with the corrected angular velocity (in counts). As can be seen, this fifth degree polynomial compensates the temperature induced error significantly.

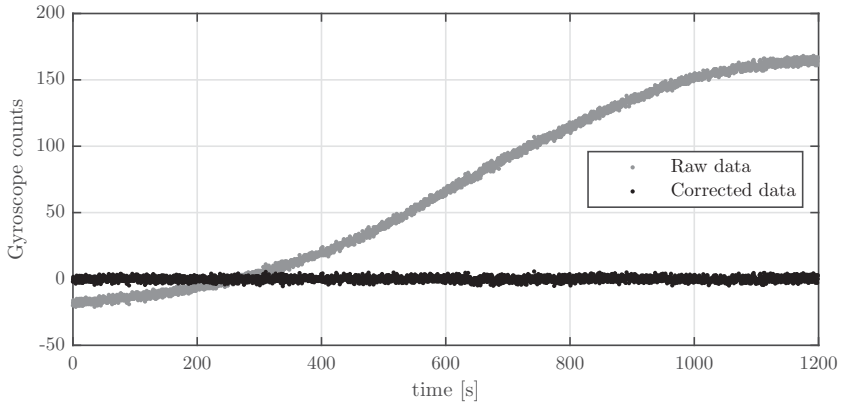

Figure A.10: Raw measurement data (grey) over time and the corrected (black) angular velocity.

A.5 Magnetometer

Unlike the accelerometer and gyroscope, the three-axis magnetometer (compass) outputs only three signals: the three orthogonal elements of the (geo)magnetic field vector in the body-fixed frame. As is the case with previous sensors, these three signals are defined as 16-bit integers.

The x-, y-, and z-components of the magnetic field, however, are offset, leading to false heading estimates. When the drone is rotated around all its axes, the measurement points of the magnetometer should all lie on a spherical surface with its center in the origin. The left graph of Figure A.11 displays the x, y components of the magnetometer signal. As can be seen, the measurements indeed lie within a circle, however, the centre of this circle does not coincide with the coordinate system origin. According to Ozyagcilar (2013), offsets in the magnetometer due to hard iron effects can be calculated by (A.13), fitting a sphere through a set of measurement data, using the least squares method.

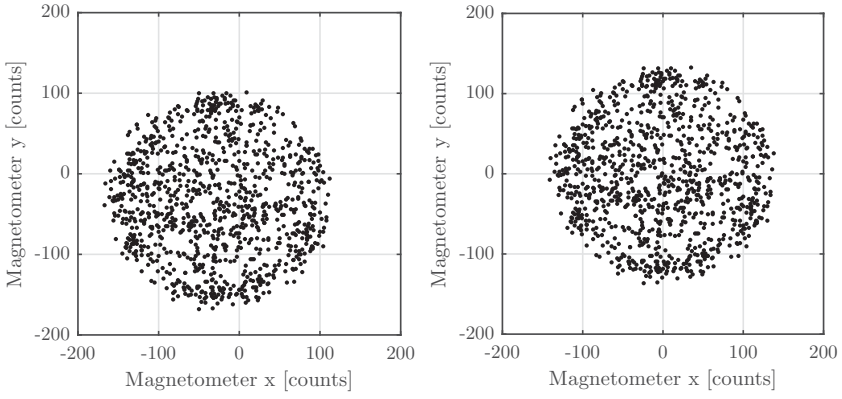

Figure A.11: Raw magnetometer x, y data of the rotating drone (left) and the same data shifted such that the measurement data lies on a sphere with its centre at the orgin (right).

We define the magnetometer field offset as

$$\tilde{B}_{\text{IMU}} = \frac{1}{2}\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}, \tag{A.13}$$

where

$$\beta = \left(X^\top X\right)^{-1} X^\top Y \in \mathbb{R}^4 \tag{A.14}$$

with

$$Y = \begin{bmatrix} B_{x,1}^2 + B_{y,1}^2 + B_{z,1}^2 \\ B_{x,2}^2 + B_{y,2}^2 + B_{z,2}^2 \\ \vdots \\ B_{x,N}^2 + B_{y,N}^2 + B_{z,N}^2 \end{bmatrix}, \qquad X = \begin{bmatrix} B_{x,1} & B_{y,1} & B_{z,1} & 1 \\ B_{x,2} & B_{y,2} & B_{z,2} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ B_{x,N} & B_{y,N} & B_{z,N} & 1 \end{bmatrix}$$

leading to the corrected magnetometer signals:

$$B_{\text{IMU}} = \hat{B}_{\text{IMU}} - \tilde{B}_{\text{IMU}} = \hat{B}_{\text{IMU}} - \frac{1}{2}\begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}. \tag{A.15}$$

The corrected x and y components of the magnetometer signal are displayed in the right graph of Figure A.11. As can be seen, the centre of the circle in which the measurement points are located now indeed coincides with the origin of the coordinate system. The calculated values for β are:

$$\beta = \begin{bmatrix} -51.9 \\ -63.1 \\ 21.6 \\ 1.35 \cdot 10^4 \end{bmatrix}.$$

The magnetometer is sensitive to disturbances in the geomagnetic field. The motors of the AR.Drone 2.0 are brushless DC motors, which have a two-pole permanent magnet as a rotor. The position of this rotor can influence the magnetic field in the vicinity of the magnetometer sensor significantly. Figure A.12 displays the raw x, y magnetometer signals where the drone is stationary. At time $t = 5$ s one of the motors is rotated 180 °. As can be seen in

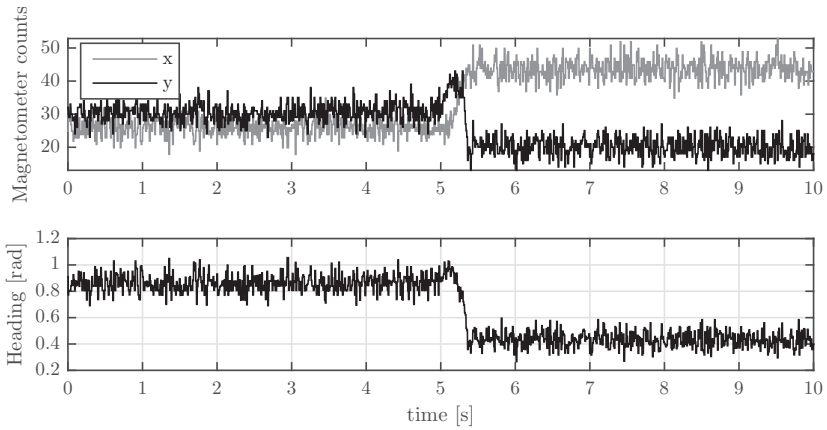

Figure A.12: Raw magnetometer data in x- and y-direction of the stationary drone where at $t = 5$ s one of the motors is rotated 180° (top), and the resulting calculated heading (bottom).

this figure, after rotation the two signals deviate significantly from their values before rotation. The bottom graph of Figure A.11 shows the resulting calculated heading. The change in the estimated heading is approximately 0.4 radians, whereas the true heading has not changed.

When the motors are rotating, the magnetic field is constantly disturbed by the motors. This disturbance, however, is not necessarily disastrous for the usability of the magnetometer for heading calculation. When the drone is in flight, the magnetic disturbance is too quick for the magnetometer to register, and can therefore be seen as a constant offset in the geomagnetic field.

In his thesis, Aert (2016) advises not to use the magnetometer of the AR.Drone 2.0 for heading estimation as it is not reliable. The sensor fails after some time (which then requires a reinitialisation of the software) and might be subject to magnetic disturbances induced by high current draw of the motors during flight. From our measurements, however, we concluded that the magnetic field vector does not change noticeably by the electrical current drawn by the motors. The magnetometer, however, indeed fails to supply new measurements after some time and

needs to be reinitialised in order for it to operate again. This has been solved by analysis of the magnetometer output and reinitialisation of the IMU board when the sensor outputs the (exact) same value for too long. Hence, only the sensor module is reinitialised when the magnetometer fails to output new measurements and the drone can continue to operate. In order to achieve this, changes have been made in the C-code that is used to build the Simulink S-functions.

A.6 Camera

The down-facing camera is used to determine the position with respect to a reference point on the floor (marker). The camera has a resolution of 320×240 px, with a supposed frame-rate of 60 fps and a field of view of 47.5°. Whereas it is possible to update the camera-block in Simulink with higher rates than 60 fps, it seems that the camera images are being updated at 30 Hz. The image defined in the $YCbCr$ color space, where Y channel is a matrix that contains the greyscale image data and the Cb and Cr channels are half-resolution (160×240) matrices that contain colour information of the image. The video signal is an unsigned 8-bit array of 153600 elements:

$$C_{\text{int}} = \begin{bmatrix} Y_1 & Cb & Y_2 & Cr \end{bmatrix}, \tag{A.16}$$

where each element in C_{int} has 38400 elements, and Y_1 and Y_2 together form the full-resolution grey-scale image.

A.7 Motors

The Parrot AR.Drone 2.0 has four motors that drive four rotors, respectively, providing thrust and torque to the body of the drone. Two rotors spin in counter clockwise direction (rotor 1 and 3 in Figure 2.2), and two in clockwise direction (rotor 2 and 4) as viewed from above. The motors can be controlled using a PWM (pulse width modulation) signal, ranging from 0 to 100, that is fed into the Motor block in Simulink. The actual motors are controlled by an electronic speed controller (ESC) that receives the PWM signal and drives the motor to the desired speed. The ESC also receives a back emf signal from the motor windings, which it uses to control the angular velocity of the motor.

Experiments have been conducted in order to find relations between the PWM signal and rotor thrust, torque and angular velocity. The measurement plan and results are given in Appendix C. Figure A.13 displays three measurement sets of the angular velocity of the rotors as a function of the PWM setting. Also, it displays a linear function that is fitted through these points. Each motor-rotor combination is slightly different, hence the linear functions

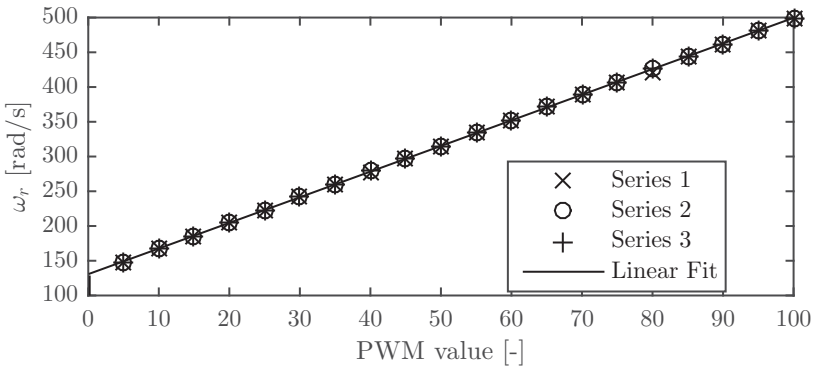

Figure A.13: Relation between the PWM settings and the actual rotor velocity, ω_r. Three measurement series are displayed along with a linear fit.

relating the angular velocity to each PWM setting differ. The linear relations are given in (A.17) which holds for

PWM values greater than 0.2. For smaller values, the rotor velocity equals 0.

$$\begin{bmatrix} \omega_{r,1} \\ \omega_{r,2} \\ \omega_{r,3} \\ \omega_{r,4} \end{bmatrix} = \begin{bmatrix} 3.7503 P_{W,1} + 132.7387 \\ 3.7123 P_{W,2} + 131.5018 \\ 3.6891 P_{W,3} + 130.7137 \\ 3.7380 P_{W,4} + 132.2209 \end{bmatrix} \tag{A.17}$$

For modelling purposes, the thrust and torque are required as functions of the PWM signals. Hence, these parameter values were also recorded during the measurement sessions. The results are displayed in Figure A.14. As

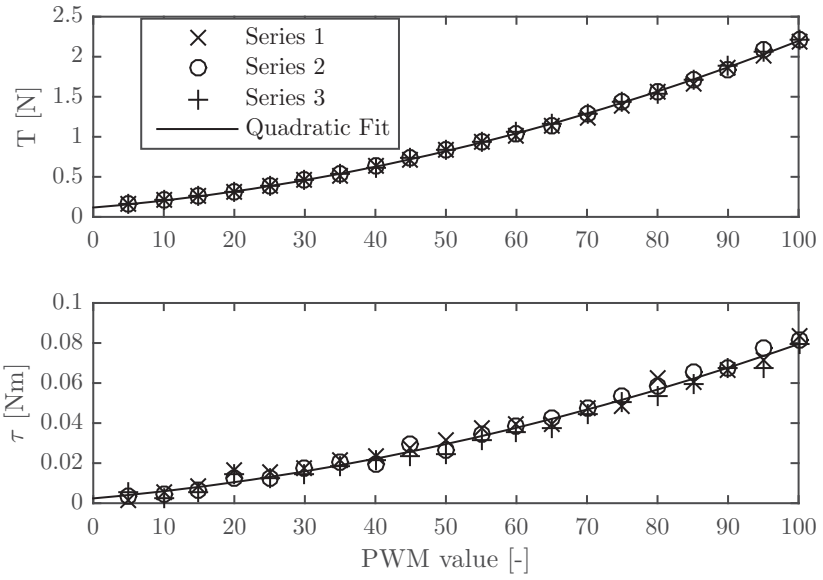

Figure A.14: Thrust (top) and torque (bottom) measurements for three different measurement series along with quadratic fits through the data points, describing relations between PWM and thrust, and PWM and torque.

can be seen, the generated thrust and torque both seem to follow quadratic trends. Hence, quadratic functions are fitted through the measurement points in order to find a description relating the thrust and torque to the PWM values. These functions are given in (A.18) and (A.19) for thrust and torque, respectively. As stated before, the motor rotor combinations differ from another leading to different results for each rotor.

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \begin{bmatrix} 1.5618 \cdot 10^{-4} P_{W,1}^2 + 1.0395 \cdot 10^{-2} P_{W,1} + 0.13894 \\ 1.8150 \cdot 10^{-4} P_{W,2}^2 + 8.7242 \cdot 10^{-3} P_{W,2} + 0.14425 \\ 1.3478 \cdot 10^{-4} P_{W,3}^2 + 7.3295 \cdot 10^{-3} P_{W,3} + 0.11698 \\ 1.4306 \cdot 10^{-4} P_{W,4}^2 + 5.7609 \cdot 10^{-3} P_{W,4} + 0.13362 \end{bmatrix} \tag{A.18}$$

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \end{bmatrix} = \begin{bmatrix} 3.4103 \cdot 10^{-6} P_{W,1}^2 + 1.9283 \cdot 10^{-4} P_{W,1} + 1.3043 \cdot 10^{-3} \\ 2.9756 \cdot 10^{-6} P_{W,2}^2 + 3.8093 \cdot 10^{-4} P_{W,2} + 5.1923 \cdot 10^{-3} \\ 4.5795 \cdot 10^{-6} P_{W,3}^2 + 3.1301 \cdot 10^{-4} P_{W,3} + 2.3996 \cdot 10^{-3} \\ 7.2886 \cdot 10^{-6} P_{W,4}^2 + 1.2779 \cdot 10^{-4} P_{W,4} + 4.6184 \cdot 10^{-3} \end{bmatrix} \tag{A.19}$$

Additionally, the relation between thrust and torque directly may be useful, in order to calculate the required PWM signals from the controller signals. Figure A.15 displays the thrust and torque in a graph. As can be seen, the measured points seem to follow a straight line, suggesting a linear relation between thrust and torque. Hence a linear function, crossing the origin, has been fitted to the measurement points. The relation is given in (A.20).

$$\begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \\ \tau_4 \end{bmatrix} = \begin{bmatrix} c_{\tau,1} f_1 \\ c_{\tau,2} f_2 \\ c_{\tau,3} f_3 \\ c_{\tau,4} f_4 \end{bmatrix} = \begin{bmatrix} 2.9107 \cdot 10^{-2} f_1 \\ 2.7543 \cdot 10^{-2} f_2 \\ 3.6171 \cdot 10^{-2} f_3 \\ 4.0559 \cdot 10^{-2} f_4 \end{bmatrix} \tag{A.20}$$

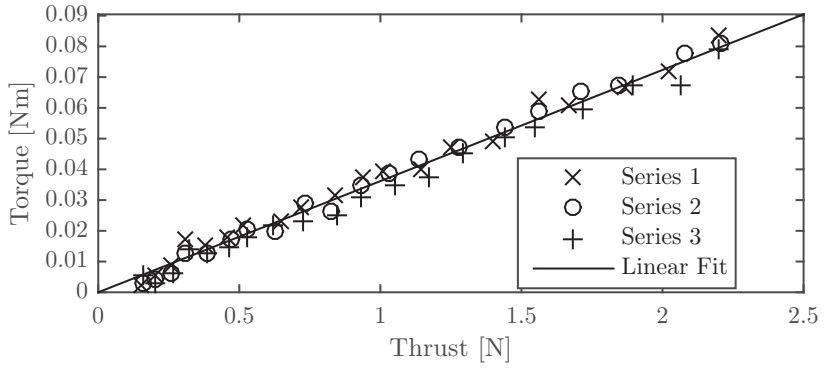

Figure A.15: Thrust and torque measurements plotted in a single graph, along with a linear function following the linear trend of the measured data.

As can be seen in Figures A.14 and A.15, the spreading in measurement data is greater for torque than for thrust. This is caused by the fact that the thrust was measured by a load cell with a sensitivity of 10 newton per volt. Since the range of the thrust during measurements was around 2 newtons, the maximum voltage range was around 0.2 volts. The used torque sensor, however, has a sensitivity of 4 Nm per volt. The range of generated torque was in the order of 0.1 Nm during measurements, giving a maximum voltage range of around 0.025 volt, which is a significantly smaller range. This results in less accurate measurements and introduces more measurement noise. The measurements can be improved by using sensors that have a greater sensitivity (smaller load range) such that the noise is reduced and accuracy is improved.

During testing several propellers have been replaced. Since the replacements may vary in performance from the original propellers, an additional calibration can be executed. For this calibration we hover the drone at a fixed location and save the PWM values.

Using:

$$\begin{bmatrix} f \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ l & -l & -l & l \\ -l & -l & l & l \\ c_{\tau,1} & c_{\tau,2} & c_{\tau,3} & c_{\tau,4} \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}, \tag{A.21}$$

with the knowledge that at hovering:

$$\begin{bmatrix} f \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} mg \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{A.22}$$

we can find expected values for the required Thrust of each rotor:

$$\begin{cases} \hat{f}_1 = \hat{f}_3 = -mg \frac{c_{\tau,2}+c_{\tau,4}}{2(c_{\tau,1}-c_{\tau,2}+c_{\tau,3}-c_{\tau,4})} \\ \hat{f}_2 = \hat{f}_4 = \quad mg \frac{c_{\tau,1}+c_{\tau,3}}{2(c_{\tau,1}-c_{\tau,2}+c_{\tau,3}-c_{\tau,4})} \end{cases} \tag{A.23}$$

From the measurement, we know the mean PWM values and, hence, can determine the corresponding estimated thrusts by

$$\tilde{f}_i = a_{2,i} P_{\mathrm{W},i}^2 + a_{1,i} P_{\mathrm{W},i} + a_{0,i} \tag{A.24}$$

$$\Delta f_i = \hat{f}_i - \tilde{f}_i \tag{A.25}$$

$$\hat{a}_{1,i} = a_{1,i} + \frac{\Delta f_i}{P_{\mathrm{W},i}}. \tag{A.26}$$

Appendix B

Error in stability analysis

Initially, the idea was to implement the position controller given by Choi and Ahn (2015). In this Appendix, however, we show how their proof of stability fails. Hence, we did not implement the postion control law proposed in their paper.

In their paper, the position errors e_9 to e_{12} are defined as

$$e_9 = x - x_d \tag{B.1}$$
$$e_{10} = \dot{x} - \dot{x}_d + k_9 e_9 \tag{B.2}$$
$$e_{11} = y - y_d \tag{B.3}$$
$$e_{12} = \dot{y} - \dot{y}_d + k_{11} e_{11}, \tag{B.4}$$

where x, y are the drone's position in $\mathcal{I}$ and x_d, y_d are the desired position components. Next, they consider a Lyapunov candidate function

$$V_3 = \frac{1}{2}\left(\zeta_1 e_9^2 + \zeta_2 e_{10}^2 + \zeta_3 e_{11}^2 + \zeta_4 e_{12}^2\right), \tag{B.5}$$

where $\zeta_i > 0,\ i \in [1, 2, 3, 4]$. The time derivative of (B.5) along (B.1)–(B.4) is given by

$$\begin{aligned}\dot{V}_3 =& \zeta_1 e_9 \dot{e}_9 + \zeta_2 e_{10} \dot{e}_{10} + \zeta_3 e_{11} \dot{e}_{11} + \zeta_4 e_{12} \dot{e}_{12} \\ =& \zeta_1 e_9 (e_{10} - k_9 e_9) + \zeta_3 e_{11} (e_{12} - k_{11} e_{11}) \\ &+ \zeta_2 e_{10} (\ddot{x} - \ddot{x}_d + k_9 (e_{10} - k_9 e_9)) \\ &+ \zeta_4 e_{12} (\ddot{y} - \ddot{y}_d + k_{11} (e_{12} - k_{11} e_{11})),\end{aligned} \tag{B.6}$$

where

$$\ddot{x} = (\cos\phi \sin\theta \cos\psi + \sin\phi \sin\psi)\frac{U_1}{m} = \gamma_1 \frac{U_1}{m} \tag{B.7}$$
$$\ddot{y} = (\cos\phi \sin\theta \sin\psi - \sin\phi \cos\psi)\frac{U_1}{m} = \gamma_2 \frac{U_1}{m}. \tag{B.8}$$

Note that ϕ, θ, and ψ herein describe the quad-rotor's orientation in Euler angles. If we assume $\ddot{x}_d = \ddot{y}_d = 0$, which is valid in case of hovering, then (B.6) can be expressed as

$$\begin{aligned}\dot{V}_3 =& \zeta_1 e_9 (e_{10} - k_9 e_9) + \zeta_3 e_{11} (e_{12} - k_{11} e_{11}) \\ &+ \zeta_2 e_{10} (\gamma_1 \frac{U_1}{m} + k_9 (e_{10} - k_9 e_9)) + \zeta_4 e_{12} (\gamma_2 \frac{U_1}{m} + k_{11} (e_{12} - k_{11} e_{11})).\end{aligned} \tag{B.9}$$

Choi and Ahn (2015) suggest choosing $\zeta_1 = \zeta_2 k_9^2$ and $\zeta_3 = \zeta_4 k_{11}^2$, such that the Lyapunov stability criterion $\dot{V}_3 < 0$ can be expressed as

$$\dot{V}_3 = \zeta_2 e_{10} \gamma_1 \frac{U_1}{m} + \zeta_2 k_9 e_{10}^2 - \zeta_2 k_9^3 e_9^2 + \zeta_4 e_{12} \gamma_2 \frac{U_1}{m} + \zeta_4 k_{11} e_{12}^2 - \zeta_4 k_{11}^3 e_{11}^2 < 0. \tag{B.10}$$

Subsequently, they claim that if k_9 and k_{11} are chosen sufficiently large, then

$$k_9 e_{10}^2 - k_9^3 e_9^2 < 0 \implies k_9 > \left| \frac{e_{10}}{e_9} \right| \tag{B.11}$$

$$k_{11} e_{12}^2 - k_{11}^3 e_{11}^2 < 0 \implies k_{11} > \left| \frac{e_{12}}{e_{11}} \right|. \tag{B.12}$$

Thus, the following condition should be ensured to satisfy the stability:

$$\dot{V}_3 = \zeta_2 e_{10} \gamma_1 \frac{U_1}{m} + \zeta_4 e_{12} \gamma_2 \frac{U_1}{m} < 0, \tag{B.13}$$

where m is the drone mass, ζ_2 and ζ_4 are controller parameters, and U_1 is the altitude control input (all positive). The position controllers γ_1 and γ_2 can be chosen such that (B.13) holds.

However, if we substitute (B.2) in (B.11), the result is:

$$k_9 (\dot{x} - \dot{x}_d)^2 + 2k_9^2 e_9 (\dot{x} - \dot{x}_d) + k_9^3 e_9^2 - k_9^3 e_9^2 < 0 \tag{B.14}$$

where the first term is non-negative and the last two terms cancel, leading to

$$2k_9 e_9 (\dot{x} - \dot{x}_d) < -(\dot{x} - \dot{x}_d)^2. \tag{B.15}$$

and similar results for (B.12). For situations where $\text{sgn}(e_9) = \text{sgn}(\dot{x} - \dot{x}_d)$ and $\text{sgn}(e_{11}) = \text{sgn}(\dot{y} - \dot{y}_d)$ these conditions do not hold. Hence, we cannot use the stability criterion of (B.13) to create stabilising controllers, proving failure of the stability analysis.

Appendix C

Measurement plans

This appendix contains the measurement plans for executing the experiments conducted in this report. Firstly, the measurement plan for temperature sensitivity of the accelerometer and gyroscope are discussed. Secondly, the measurement plan for determination of the mass moments of inertia of a multi-rotor are discussed.

C.1 Sensor noise

The goal of this experiment is to map the noise levels of all used sensors for modelling purposes. This is a simple experiment where only sensor signals are measured during a short period of time. The items required for conducting this experiment are:

- Laptop equipped with Matlab/Simulink 2014b and the AR.Drone 2.0 toolbox installed.
- AR.Drone 2.0.

Experiment procedure

1. Create a Simulink model that reads the desired sensors and saves their signals to a file. Do not forget to include a measure to stop the programme, either by means of a timer, or an external signal. An unclean termination of the software leads to a corrupt `.mat` file and loss of data.
2. Place the drone such that it remains still for the duration of the sensor measurement.
3. Turn on the drone, connect to its WiFi access point, and build the Simulink model and wait for the programme to initiate on the drone and finish its measurements.
4. Download the `.mat` file containing sensor data to the laptop using FTP. (located in `/update` on the AR.Drone 2.0.)
5. The `.mat` file contains the saved sensor measurements, loading it into the Matlab workspace and using the built-in function `var(.)` outputs the sensor noise variance. The function `lillietest(.)` can be used to verify whether the sensor noise has a certain distribution.

C.2 Temperature sensitivity of sensors

In a previous part of this research project (Jeurgens, 2016), we have seen that the accelerometer and gyroscope are sensitive to temperature changes. The sensor packages of these sensors are equipped with internal temperature sensors that measure the package temperature. In this experiment we measure the sensors under influence of a varying temperature, such that a relation between the sensor temperature counts and sensor signal counts can be established. This relation then can be used to compensate for temperature fluctuations during operation of the AR.Drone 2.0. The items required for conducting this experiment are:

- Laptop equipped with Matlab/Simulink 2014b and the AR.Drone 2.0 toolbox installed
- AR.Drone 2.0 with charged battery.

- Chilled room such as refrigerator.
- Heat source such as a space heater.
- Confined space that can accommodate both the AR.Drone 2.0 and the heat source.
- Thermocouple.

EXPERIMENT PROCEDURE

1. Mount the thermocouple inside the drone, such that the temperature of the main board can be measured.
2. Place the AR.Drone 2.0 inside the chilled room until a steady state temperature is reached. Use the thermocouple to check the temperature.
3. Meanwhile, prepare a Simulink model that reads the sensor signals, including the temperature signals, and saves them to a file. Do not forget to include a measure to stop the programme by means of an external signal. An unclean termination of the software leads to a corrupt `.mat` file and loss of data.
4. When a steady-state temperature is reached, remove the drone form the chilled room, connect the battery such that it boots, and place it in the confined space with the heater.
5. Connect the laptop to the WiFi access point and build the Simulink model and wait for the programme to initiate on the drone.
6. Start the heater and monitor the drone's temperature using the thermocouple.
7. When a steady-state temperature is reached, turn off the heater and let the drone cool down, monitoring the temperature using the thermocouple.
8. When again a steady-state temperature is reached, the measurement can be terminated by sending a stop command to the programme running on the drone.
9. Download the `.mat` file containing the sensor data to the laptop using FTP.
10. Analyse the sensor signals and relate the sensor temperature counts to the sensor signal counts.

C.3 MASS MOMENT OF INERTIA

One of the system parameters of the quad-rotor model discussed in Chapter 2 is the mass moment of inertia. In this experiment we aim to measure this parameter value using the internal sensory equipment. The method for determination of the mass moment of inertia is based on the bifilar pendulum experiment by Jardin and Mueller (2009). A schematic view of the bifilar pendulum experiment set-up is given in Figure C.1. The items required for conducting this experiment are:

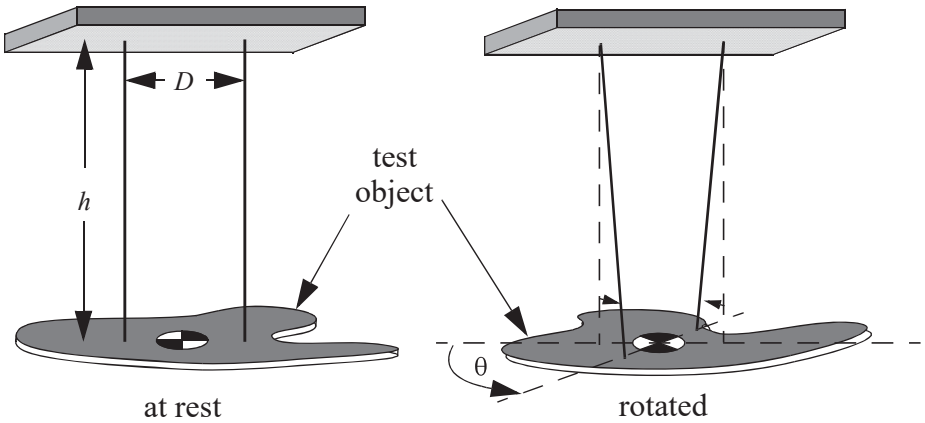

Figure C.1: Bifilar pendulum diagrams of the experiment setup. Source: Jardin and Mueller (2009).

- Laptop equipped with Matlab/Simulink 2014b and the AR.Drone 2.0 toolbox installed
- AR.Drone 2.0 with charged battery.
- Two equal-length wires and a rigid frame to suspend the drone on.

Experiment procedure

1. Create a Simulink model that reads the gyroscope signals and saves them to a file. Do not forget to include a measure to stop the programme, either by means of a timer, or an external signal. An unclean termination of the software leads to a corrupt `.mat` file and loss of data.
2. Suspend the drone by two vertical, parallel wires, attached near the rotors at the drone end and at the rigid frame above it and connect the battery. Note the wire length h and distance between wires D.
3. Connect the laptop to the WiFi access point of the AR.Drone 2.0 and build the Simulink model and wait for the programme to initiate on the drone.
4. Give the drone an initial rotation θ_0 around the vertical axis, and release.
5. Let the drone oscillate until the oscillation is (nearly) damped out and stop the measurement.
6. Download the `.mat` file containing the sensor data to the laptop using FTP.

Now that we have sensor measurements of the gyroscope, we can use the equations of motion that describe the bifilar pendulum to determine the mass moment of inertia. The equations of motion for this system are given by

$$\begin{cases} x_1 = \theta \\ x_2 = \dot{\theta} \end{cases} \tag{C.1}$$

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = -\left(\frac{K_D}{I}\left|x_2\right| + \frac{C}{I}\right) x_2 + \frac{mgD^2}{4Ih} \frac{1}{\sqrt{1-\frac{1}{2}\left(\frac{D}{h}\right)^2(1-\cos x_1)}} \sin x_1, \end{cases} \tag{C.2}$$

where K_D and C are aerodynamic and viscous friction parameters, I is the moment of inertia around the measured rotation axis, m and g are the mass of the quad-rotor and the gravitational acceleration, and D and h are the distance between the two wires and the wire length, respectively.

The set of equations (C.2) can be simulated in Matlab, using the same sample rate as the gyroscope in the measurement, providing $\dot{\theta}_{\text{sim}}$. A minimizer searching function, such as `fminsearch(.)`, can be used to find the values for K_D, C, and most importantly I that correspond to the minimum error between the measured and simulated angular velocities $\left\|\dot{\theta}_{\text{sim}} - \omega_{\text{IMU}}\right\|$. This experiment can be done several times, such that the mass moment of inertia is determined for each principle body-fixed axis, for each hull.

C.4 Magnetometer

The goal for this experiment is to determine the hard-iron offset of the magnetometer. In order to determine this, the quad-rotor is rotated randomly around all axes, such that all magnetometer measurements should lie on the surface of a sphere. As the position of the motors can influence the magnetic field that is measured, we run the motors during this experiment. The rotating motors generate such quickly changing magnetic fields that it no longer is measured by the magnetometer. As during flight the motors are running, this measurement is more representative for flight when conducted with active motors. The items required for conducting this experiment are:

- Laptop equipped with Matlab/Simulink 2014b and the AR.Drone 2.0 toolbox installed.
- AR.Drone 2.0.
- A tool to remove the clips from the rotor axles.

EXPERIMENT PROCEDURE

1. Remove the rotors, large gears, and rotor axles from the motor assemblies by removing the clips that hold them in place.
2. Create a Simulink model that reads the magnetometer signals and saves them to a file and turns on the motors at 50% throttle. Do not forget to include a measure to stop the programme, either by means of at timer, or an external signal. An unclean termination of the software leads to a corrupt `.mat` file and loss of data.
3. Connect to the WiFi access point of the AR.Drone and build the Simulink model.
4. While the motors are running, rotate the drone randomly around al axes multiple times, away from sources of magnetic interference.
5. When the measurement is complete, download the `.mat` file tot the laptop using FTP.

Methods discussed in Appendix A.5 can be used to determine and compensate for the hard-iron offset.

C.5 ALTIMETER ALTITUDE

The goal for this experiment is to determine the relation the ultrasonic altimeter sensor signal to the actual altitude in metres, such that this sensor can be used in the controller. Hereto, we measure the sensor output at several predetermined altitudes. The items required for conducting this experiment are:

- Laptop equipped with Matlab/Simulink 2014b and the AR.Drone 2.0 toolbox installed.
- AR.Drone 2.0.
- A tape-measure, or alternative manner of distance measurement.

EXPERIMENT PROCEDURE

1. Create a Simulink model that reads the altimeter *ultrasound* signal and saves it to a file. Do not forget to include a measure to stop the programme, either by means of a timer, or an external signal. An unclean termination of the software leads to a corrupt `.mat` file and loss of data.
2. Connect to the WiFi access point of the AR.Drone and build the Simulink model.
3. Position the drone level, such that the altimeter is aimed perpendicular to the floor.
4. Measure the distance from the floor to the drone.
5. Stop the software on the drone and download the `.mat` file for off-line analysis.
6. Repeat steps 2–5 for different altitudes.

C.6 ALTIMETER ATTITUDE

The attitude of the drone compared to the floor can be of influence of the altimeter measurement. This experiment aims to relate the angle between the drone and the floor to the measured altitude, such that this can be compensated for. The items required for conducting this experiment are:

- Laptop equipped with Matlab/Simulink 2014b and the AR.Drone 2.0 toolbox installed.
- AR.Drone 2.0.
- A tape-measure, or alternative manner of distance measurement.

Experiment procedure

1. Create a Simulink model that reads the altimeter *ultrasound* and accelerometer signals and saves them to a file. Do not forget to include a measure to stop the programme, either by means of a timer, or an external signal. An unclean termination of the software leads to a corrupt `.mat` file and loss of data.
2. Connect to the WiFi access point of the AR.Drone and build the Simulink model.
3. Position the drone level, such that the altimeter is aimed perpendicular to the floor.
4. Measure the distance from the floor to the drone.
5. Slowly rotate the drone, while maintaining the same altitude.
6. Stop the software on the drone and download the `.mat` file for off-line analysis. The accelerometer can be used to determine the orientation of the drone, assuming the drone's location is fixed and only its orientation varies.
7. Repeat steps 2–6 for different altitudes.

C.7 Motors

This measurement plan describes the steps and set up to experimentally determine the thrust, torque, current, and angular velocity of the rotors, depending on motor commands. In order to conduct these measurements, the following items are required:

- Windows laptop/PC, equipped with a WiFi adapter, with LabVIEW SignalExpress installed and Simulink configured correctly to connect to the AR.Drone 2.0 and control the four rotors individually,
- Variable power supply with built in voltage and current displays (seperate voltage and current sensors can be added in case the power supply is not equipped with these),
- Sensor interface,
- Force transducer (AST KAP-E/50N),
- Torque transducer (Lorenz D-2553),
- Manual optical RPM counter (Voltcraft DT-1 opto),
- Mounting hardware (bolts, nuts, mounting plates and bracket),
- Cable to provide power to the AR.Drone 2.0.

Measurement setup

Figure C.2 shows the test setup. The AR.Drone is mounted upside-down to a bracket in order to reduce the ground effect, which could cause erroneous results. The bracket is mounted to the force transducer, which is mounted with an adapter plate to the torque transducer. This whole assembly then is mounted to the table. The sensor interface is connected to the force and torque transducers in order to read the sensor voltages. Also, it provides extra power to the amplifier of the torque transducer. The variable power supply is connected to the battery connector of the AR.Drone 2.0, in place of the battery, such that it can operate without introducing voltage drop due to battery depletion. Hence, this is a more stable source of power, with the additional benefit that the voltage can be controlled and the current draw can be monitored. The angular velocity of the rotors need to be measured manually using the manual optical RPM counter.

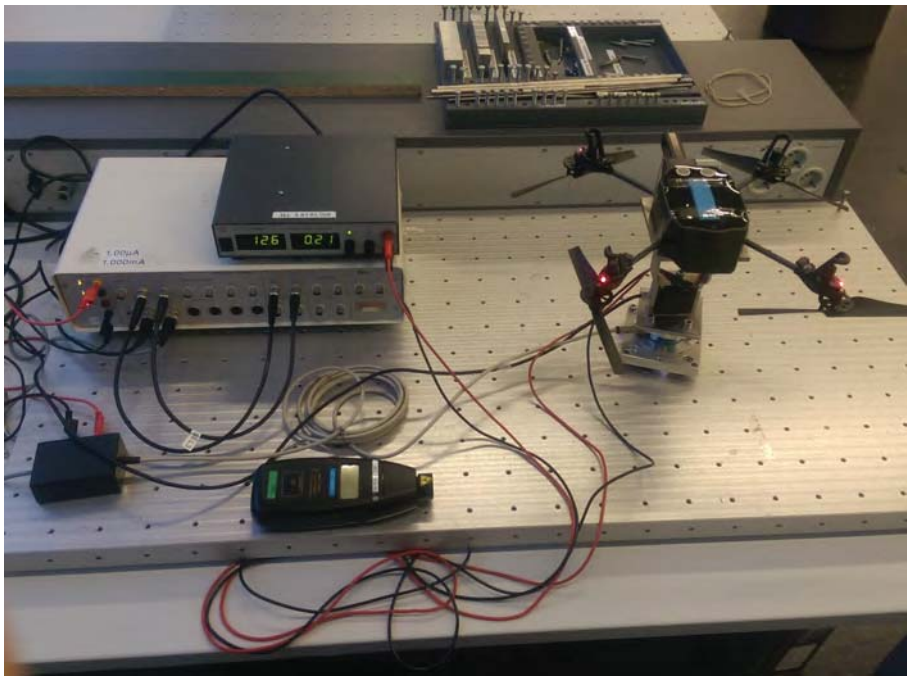

Figure C.2: Picture of test setup. On the left is the sensor interface with on top a variable power supply, on the right is the AR.Drone upside-down connected to the force transducer, mounted to the torque transducer, which is mounted to the table.

Measurement preparation

After setting up, the following steps need to be executed in order to ensure that measurements can be conducted.

1. Make sure all sensors are connected, the drone is securely mounted, and that the polarity of the power supply leads to the battery connector of the drone is correct.
2. Connect the usb cable of the sensor interface to the PC and start LabView SignalExpress. Make sure that the interface is recognised by the software and power up the interface. The sensors may need some time to reach a steady-state.
3. Set up LabView to read the correct sensor values and average the signal over a period of 5 seconds in order to determine the mean thrust and torque. The signals may vary due to signal noise, but also vibrations in the thrust and torque will be introduced due to imbalance of the rotors. Also, set up LabView such that it runs once, not continuously.
4. Dial the voltage knob of the power supply all the way down and the current knob (if applicable) fully up.
5. Switch on the power supply and increase the voltage to 12.6 volts (voltage of a fully charged 3S LiPo battery), the drone should now power on. If the drone does not power on, leave the voltage of the power supply at 12.6 volts and switch it off and on again.
6. Connect the PC to the WiFi access point of the AR.Drone 2.0 and start a Simulink model that is equipped with the Init_Actuator block and the Motor block and configured to run in external mode with the AR.Drone 2.0. Ensure that all motor commands equal 0.
7. Build the Simulink model and start the real-time connection.

Measurement procedure

The following steps need to be repeated for each rotor.

1. Read and note the current draw of the drone from the power supply while all pwm values are set to 0.
2. Read the sensor values of the thrust and torque transducers in SignalExpress while all pwm values are set to 0.
3. Increase the pwm signal of the desired motor with an increment of 5 in Simulink (while the external mode is running).
4. Manually measure the angular velocity of the rotor using the optical RPM counter.
5. Run the LabView measurement.
6. Read the current draw of the drone.
7. Read the average torque and force signals.
8. Repeat steps 3–7 until the maximum pwm value of 100 is reached.

The motor thrust and torque can be calculated by substracting the measured values at rest from the measured values at the given PWM value. Then multiply the sensitivity to obtain the thrust and torque in Newton and Newton metres, respectively (for the above listed sensors these are 10 N/V for the force transducer and 4 Nm/V for the torque transducer). The voltage and current can be directly read from the power supply in volts and ampères. The RPM counter counts two revolutions for every actual revolution of the rotor, since the rotors have two blades each. Hence, the angular velocity of the rotors in radians per second can be deduced from the RPM readings as follows: $\omega_{r,i} = RPM_{r,i}/60 * \pi$.

Appendix D

Simulink manual

This Appendix describes the steps required to operate the AR.Drone 2.0 wirelessly from a PC with Simulink. This package only works on Windows computers and with Matlab Simulink versions 2014a and 2014b. Newer versions do not work. This software package makes use of the GitHub project resources from Daranlee and Slovak194, n.d. And whereas the compiler tool chain is provided on the DVD, it can also be downloaded directly from:

`https://sourcery.mentor.com/sgpp/lite/arm/portal/subscription?@template=lite`

A version of the Cyberduck command line interface is also included on the DVD. This is, however not necessary to install, unless there are issues uploading the generated binary to the AR.Drone 2.0 (explained later). If desired, the latest version of the Cyberduck command line interface can be downloaded directly from:

`https://dist.duck.sh/`

Note, for the use of the video library, it is required that Microsoft Visual C++ 2013 Professional (C) is the configured C compiler in Matlab. This can be checked by running "`mex -setup`" in the Matlab command window.

D.1 Installation of the toolbox

1. Extract the ARDrone_Simulink.zip file into a location of your desire, hereafter indicated by `root`.
2. Install the CodeSourcery toolchain, located in `root/Included_Software/Compiler.exe` and remember the install directory. If the installer fails with the error "Installer User Interface Mode Not Supported", try executing the installer in Windows 7 compatibility mode.
3. Execute the Matlab script `root/AR_Drone_2.0_Toolbox/install_script.m`. This installs the Simulink blocks that can be used to control the AR.Drone. It also asks the location of the compiler in an explorer window, browse to this location and select the folder where `arm-none-linux-gnuabi-gcc.exe` and similar files are located (typically: `C:/Program Files (x86)/CodeSourcery/Sourcery G++ Lite/bin`).

D.2 Implementation of the control software

In the folder `root/Simulink_Controller/` are, among others, six Simulink models located:

1. `AR_Drone_ControlPanel.slx`, this model contains an interface for connecting to the drone when the controller is compiled in "standalone" mode. It receives several signals, such as position and orientation, and sends the desired trajectory identifier, control gains, and a stop command to the drone.

2. `ARDrone_External`, this model implements the controller from `Controller_NonLin_Sebastiaan.slx` and the observer from `Observer_DLKF_2mag.slx` in the actual drone. It runs in, either external, or standalone mode and requires building and compilation. This file can be edited to change the sensor pre processing (i.e. filtering and signal selection).

3. `ARDrone_Simulation_Measurements.slx`, this model uses sensor measurements (saved on the drone when the controller runs in standalone mode) to simulate the observer and controller behaviour. It, therefore, is only necessary to save the sensor signals as the rest can be reconstructed.

4. `ARDrone_Simulation_Model.slx`, this model implements the controller from `Controller_NonLin_Sebastiaan.slx` and the observer from `Observer_DLKF_2mag.slx` in a simulation. This can be used to test the controller on the model of Chapter 2. This file can be edited to change/adapt the model of the AR.Drone 2.0.

5. `Controller_NonLin_Sebastiaan.slx`, this model contains the actual control laws of Chapter 3 and the position and velocity observer of Chapter 4.

6. `Observer_DLKF_2mag.slx`, this model contains the orientation observer of Chapter 4.

Before starting either one of these models, the .m file `AR_Drone_Parameters.m` needs to be executed. It loads the parameter values for the model and the controller from `Model_Parameters.mat` and the configuration parameters for the IMU data bus and the Simulink configuration file. In this file the user can select whether to use the standalone or external mode in simulink. This can be done by setting the variable `STANDALONE = 1`, when the standalone mode is desired, and `STANDALONE = 0` when the external mode is desired.

If a controller has been designed in Simulink, it can be tested by running the model `AR_Drone_Simulation_Models.slx`. When the results are satisfactory and the designed controller is ready for testing on the actual drone, the Simulink model `ARDrone_External.slx` needs to be opened and built (`ctrl+b`). During the build process, the compiler creates a binary that can be run on the AR.Drone 2.0 and automatically uploads it via FTP. A Windows console window opens and gives some information, do not close this window until the Simulink model is stopped.

When the external mode of Simulink is selected, after building and uploading, click on connect in Simulink and then on run. This starts the control software on the drone.

When the standalone mode of Simulink is selected, after building and uploading, the control software is started automatically. A connection to the drone can be made by running the Simulink model `AR_Drone_ControlPanel.slx`. When connection is lost, or a stop command is sent to the drone, the control software terminates cleanly. In Matlab, the file `AR_Drone_PostDownload.m` can be used to download all `.mat` files in `/update/` back to the PC, using FTP. Note that files will be overwritten if a file already exists.

Note that the standard Simulink blocks "to file" and "to workspace" do not seem to work in external mode. The Simulink scopes, however, can be used to export data to the workspace. Additionally, the scopes can be opened during the execution of the external mode of Simulink, real-time data then is displayed. In standalone mode, the to file block does work and saves the selected signals in a `.mat` file, located at `/update/` on the drone.

If the upload of the generated binary (`.elf`) file fails, try installing Cyberduck (can be used to transfer files via ftp) from `root/software/Cyberduck.exe` and change the lines

```
echo open %AR_DRONE_IP_ADDRESS% [5551] >> ftpcmd.dat
echo user >> ftpcmd.dat
echo put "%EXE_PATH%%EXE_NAME%">> ftpcmd.dat
echo disconnect >>ftpcmd.dat
echo quit >> ftpcmd.dat
echo Connecting to FTP and uploading binary
ftp -s:ftpcmd.dat
echo Done Uploading
del ftpcmd.dat
```

in the file `root/AR_Drone_Target/ssh_download.bat` to:

```
SET DUCK="C:\Program Files (x86)\Cyberduck CLI\duck.exe"
ECHO FTP uploading binary to %AR_DRONE_IP_ADDRESS%:5551 using Cyberduck
ECHO PATH: %EXE_PATH%%EXE_NAME%
ECHO IP:   ftp://%AR_DRONE_IP_ADDRESS%:5551/
%DUCK% -upload ftp://%AR_DRONE_IP_ADDRESS%:5551/ "%EXE_PATH%%EXE_NAME%"
```

Herein is `"C:\Program Files (x86)\Cyberduck CLI\duck.exe"` the install directory of the command line interface of Cyberduck.

D.3 Segmentation fault

During testing using the drone, one recurring issue that we encountered was crashing of the controller software due to segmentation faults. Such fault is caused by a program that tries to acces a block of memory that is outside of wat it has been assigned. This issue occured mainly while calculating n^{th} root functions of the form `a^(1/n)` in matlab. An alternative manner of calculating such root is using the function `nthroot(a,n)`. This, however, does not solve the issue of segmentation faults. Hence, we recommened using a numerical approximation of the n^{th} root for implementation, such as the following algorithm that solves $x^n = a$ for x: Using Algorithm D.1, the

Algorithm D.1 Numerical approximation algorithm of the n^{th} root (Atkinson, 1989).

$x_0 = a$;
$k = 0$;
while $\Delta x_k \geq \text{Tol}_x$ & $k \leq \text{maxIter}$ **do**
  $\Delta x_k = \frac{1}{n}\left(\frac{a}{x_k^{n-1}} - x_k\right)$;
  $x_{k+1} = x_k + \Delta x_k$;
  $k = k + 1$;
end while

occurrence of segmentation fault is significantly reduced. This fault, however, still occasionally occurs. Hence, additional investigation is required in order to completely eliminate crashing of the control software, caused by segmentation faults.