Institutionen för datavetenskap
Department of Computer and Information Science

Master's Thesis

Towards Autonomous Landing of a Quadrotor using Monocular SLAM Techniques

Jonatan Olofsson

Linköping University

INSTITUTE OF TECHNOLOGY

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Institutionen för datavetenskap
Department of Computer and Information Science

Master's Thesis

Towards Autonomous Landing of a Quadrotor using Monocular SLAM Techniques

Jonatan Olofsson

Supervisor: **Rudol, Piotr**
IDA, Linköpings universitet
Schön, Thomas
ISY, Linköpings universitet
Wzorek, Mariusz
IDA, Linköpings universitet

Examiner: **Doherty, Patrick**
IDA, Linköpings universitet

Department of Computer and Information Science
Linköpings universitet
SE-581 83 Linköping, Sweden

Titel
Title   Towards Autonomous Landing of a Quadrotor using Monocular SLAM Techniques

Författare Jonatan Olofsson
Author

Sammanfattning
Abstract

Use of Unmanned Aerial Vehicles have seen enormous growth in recent years due to the advances in related scientific and technological fields. This fact combined with decreasing costs of using UAVs enables their use in new application areas. Many of these new tasks are suitable for miniature scale UAVs - MAVs - which have added advantage of portability and ease of deployment, as well as reducing costs further.

One of the main functionalities necessary for successful MAV deployment in real-world applications is autonomous landing. Landing puts particularly high requirements on positioning accuracy, especially in indoor confined environments where the commonly used GPS positioning technology is unavailable. For that reason using an additional sensor, such as a camera, is beneficial.

In this thesis, a set of technologies for achieving autonomous landing is developed and evaluated. In particular, state estimation based on monocular vision SLAM techniques are fused with onboard embedded sensors. This is then used as the basis for nonlinear adaptive control as well trajectory generation for a simple landing procedure. These components are connected using a new proposed framework for robotic development.

The proposed system has been fully implemented and tested in a simulated environment and validated using recorded data. Basic autonomous landing was performed in simulation and the result suggests that the proposed system is a viable solution for achieving fully autonomous landing.

Abstract

Use of Unmanned Aerial Vehicles have seen enormous growth in recent years due to the advances in related scientific and technological fields. This fact combined with decreasing costs of using UAVs enables their use in new application areas. Many of these new tasks are suitable for miniature scale UAVs - MAVs - which have added advantage of portability and ease of deployment, as well as reducing costs further.

One of the main functionalities necessary for successful MAV deployment in real-world applications is autonomous landing. Landing puts particularly high requirements on positioning accuracy, especially in indoor confined environments where the commonly used GPS positioning technology is unavailable. For that reason using an additional sensor, such as a camera, is beneficial.

In this thesis, a set of technologies for achieving autonomous landing is developed and evaluated. In particular, state estimation based on monocular vision SLAM techniques are fused with onboard embedded sensors. This is then used as the basis for nonlinear adaptive control as well trajectory generation for a simple landing procedure. These components are connected using a new proposed framework for robotic development.

The proposed system has been fully implemented and tested in a simulated environment and validated using recorded data. Basic autonomous landing was performed in simulation and the result suggests that the proposed system is a viable solution for achieving fully autonomous landing.

Acknowledgments

Contents

Chapter 1

Introduction

In this thesis, a proposed high-level control system is presented for the quadrotor developed by the AIICS[1] team at Linköping University, the LinkQuad. The system was to demonstrate autonomous landing, and while this goal could not be reached within the time-frame of this Master's thesis, the control system developed in this thesis show promising initial results on the path to achieving autonomous landing and aerial control.

Although the LinkQuad quadrotor was used as development platform for this thesis, the results are general and applicable to other quadrotors and, by extension, other UAV configurations.

The LinkQuad is considered a MAV, a Micro Air Vehicle. The size of the system poses limitations on the payload and processing power available in-flight. This means that the implementations has to be done in an efficient manner on the small computers that are available on the LinkQuad.

These limitations, however, does not necessarily limit us from utilizing advanced estimation and control techniques, and the LinkQuad design is in fact quite suitable for the camera-based pose estimation due to its dual onboard computers which allows distribution of the workload. The video-based estimation can thus effectively be detached from the core control and state estimation and modularized as an independent virtual sensor, which is exploited in this thesis.

1.1   Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAVs) have been imagined and constructed for millennia, starting in ancient Greece and China [42]. The modern concept of UAVs was introduced in the first world war, which illuminates the dominant role that the military has played in the field over the last century. A commonly cited alliteration is that UAVs are intended to replace human presence in missions that are "Dull, Dirty and Dangerous".

---

[1] http://www.ida.liu.se/divisions/aiics/

While the military continues to lead the development in the field [34], recent years have seen a great increase in domestic and civilian applications [45]. These applications range from pesticide dispersing and crop monitoring to traffic control, border watch and rescue scenarios [10].

The type of UAV that is used in the implementation of this thesis falls under the category of Micro Air Vehicles (MAVs)[2]. MAVs are designed to be man-portable in weight and size and is thus limited in payload and available processing power. This limitation, in combination with the unavailability of indoor positioning (e.g. GPS), has led to extensive use of off-board positioning and control in recent research. Systems developed for instance by Vicon[3] and Qualisys[4] yield positioning with remarkable precision, but they also limit the application to a confined environment with an expensive setup.

This thesis seeks a different approach, with an efficient self-contained on-board implementation. GPS and external cameras are replaced by inertial sensors and an on-board camera which uses visual SLAM to determine the position of the LinkQuad relative to its surroundings.

1.2 The LinkQuad Platform

The LinkQuad is a modular quadrotor developed at Linköping University. The core configuration is equipped with a standard set of sensors (accelerometers, gyroscopes, pressure sensors, GPS and magnetometers), but for the purpose of this thesis, a monocular camera was also mounted. The camera feeds data into a microcomputer specifically devoted to video processing. This devoted microcomputer is what allows the primary on-board microcomputer to focus on state estimation and control, without being overloaded by video processing.

The primary microcomputer is running a framework named C++ Robot Automation Platform (*CRAP*), which was developed by the thesis' author for this purpose. *CRAP* is a light-weight automation platform with a purpose similar to that of ROS[5], a modular robotics framework developed by Willow Garage[6]. It is, in contrast to ROS, primarily designed to run on the relatively low-end Linux systems that fits the payload and power demands of a MAV. The framework is further described in Appendix A.

The platform is also equipped with two microcontrollers, responsible for sensor sampling, motor control, flight logging etc., as depicted in Figure 1.1. The microcontrollers contain an implementation of a complementary filter to provide angle and height estimates from the available sensors.

Using the *CRAP* framework, the functionality of the thesis implementation is distributed in separated modules:

- **Observer:** Sensor fusing state estimation. Chapter 3.

---

[2]Definitions differ for the classification of UAVs, although a weight less than 5 kg has been proposed [1]. Other terms, such as *Small UAVs* (SUAVs), are used by e.g. [42].

[3]http://www.vicon.com/

[4]http://www.qualisys.com/

[5]http://www.ros.org/

[6]http://www.willowgarage.com/

- **Control:** Affine Quadratic Control. Chapter 4.

- **Logic:** State-machine for scheduling controller parameters and reference trajectory. Chapter 5.
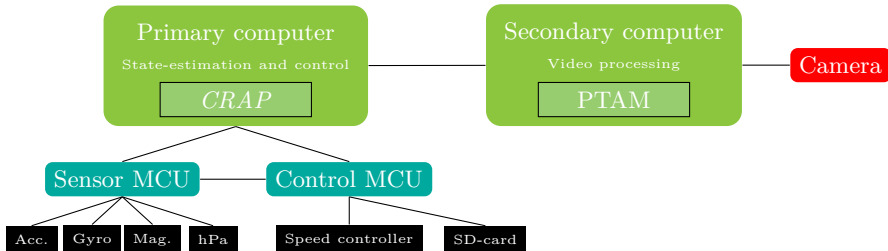
Figure 1.1. LinkQuad schematic.

Figure 1.2. The LinkQuad development platform.

1.3   Related Work

Positioning an unmanned quadrotor using visual feedback is a problem which has received extensive attention during recent years, and only recently been implemented with convincing results [3, 44]. Few have attempted to use on-board sensors only, but have relied on external setups to track the motions of a quadrotor - reportedly with high precision. Fewer still have succeeded using strictly real-time algorithms running on the limited processing power that standard MAVs generally are equipped with, although successful implementations do exist, e.g. [38].

1.3.1   Autonomous Landing and Control

The problem of landing a quadrotor can to a large extent be boiled down to achieving good pose estimates using available models and sensor measurements. The problem is studied in e.g. [28, 6], but perhaps the most interesting results are obtained in [3, 44], where the ideas from [22] of using Monocular SLAM are

implemented on a UAV platform and good results are obtained, using the feedback from the camera's pose estimate for control. Some problems remain, however, regarding the long-term stability of the controller.

A landing control scheme, inspirational to the approach in this thesis, is suggested by [6], which is summarized in Section 5.4.

Both Linear Quadratic control and PID controllers have been used for control in aforementioned projects, and ambiguous results have been attained as to which is better [4]. Continued effort of quadrotor modeling have however shown great potential of the LQ design [5].

1.3.2   Visual SLAM

Applying the idea of Simultaneous Localization And Mapping (SLAM) to the data available in a video feed is known as Visual SLAM (vSLAM). A directional paper is presented in [21], resting on the foundation of a Rao-Blackwellized particle filter with Kalman filter banks to track landmarks recognized from previous videoframes. Similar approaches, using Kalman filtering solutions, have been implemented by e.g. [9, 11], and are available as open-source software[7].

Another approach to vSLAM is suggested in [22] and implemented as the PTAM - Parallel Tracking And Mapping - library. This library splits the tracking problem - estimating the camera position - of SLAM from the mapping problem - globally optimizing the positions of registered features with regards to each other. Without the constraint of real-time mapping, this optimization can run more advanced algorithms at a slower pace than required by the tracking.

Also, by for instance not considering the full uncertainties in either camera pose or feature location, the complexity of the algorithm is reduced and the number of studied points can be increased to achieve better robustness and performance than when a filtering solution is used [40].

A modified version of the PTAM library has been implemented on an iPhone [23]. This is of special interest to this thesis, since it demonstrates a successful implementation in a resource-constrained environment similar to that available on a MAV. The library has been previously applied to the UAV state estimation field, as presented in [44], although its primary field of application is Augmented Reality (AR).

1.4   Objectives and Limitations

The main objective for this thesis was to create a control system for the LinkQuad capable of autonomous landing, using state estimation aided by visual feedback. To achieve this, methods and algorithms for control and positioning were to be implemented as nescessary to incorporate the complex measurements from the camera positioning into the state estimate. With the pose estimate and control available, landing is a matter of generating a suitable trajectory and detecting the completion of the landing.

---

[7]`http://www.doc.ic.ac.uk/~ajd/Scene/`

Although time restricted the final demonstration of landing, the necessary tools were implemented, albeit lacking the tuning necessary for real flight.

This thesis does not cover the detection of suitable landing sites, nor any advanced flight control in limited space or with collision detection. The quadrotor modeling is extensive, but is mostly limited to a study of literature on the subject.

1.5 Contributions

During the thesis work, several tools for future development have been designed and developed. The *CRAP* framework collects tools that are usable in future projects and theses, both on the LinkQuad and otherwise. The modules implemented in the *CRAP* framework include:

- General nonlinear filtering, using EKF or UKF,

- General nonlinear control, implemented as affine quadratic control,

- Extendable scheduling and reasoning through state-machines,

- Communication API for internal and external communication.

Furthermore, a general physical model of a quadrotor has been assembled. The model extends and clarifies the many sources of physical modeling available, and is presented in a scalable, general manner.

The state estimation proposed in this thesis uses a detailed physical model derived in Chapter 3. While the model still needs tuning, it does show promising results, and a physically modeled quadrotor could potentially improve in-flight performance [5].

In Chapter 3, a general method for attaining the world-to-PTAM transformation is proposed. This method is useful for extending the utility of the PTAM camera positioning library to more than its intended use in Augmented Reality applications. The utility for this has already been proven in e.g. [44], and providing the theory and implementation for this, as well as a proposed initialization routine, may be relevantly applied to future work. The PTAM library has also been extended with full autonomous operation by proposing an automated initialization procedure, as well as providing full detachment from the graphical user interface.

All tools developed during the thesis are released under the GPL license and are available online at `https://github.com/jonatanolofsson/`.

1.6 Thesis Outline

Following the introductory chapter, four chapters are devoted to presenting the theory used in the implementation. The primary sensor used in this thesis, the camera, is introduced in Chapter 2 on the topic of Monocular SLAM. The camera provide positioning measurements that are incorporated into a positioning framework in Chapter 3. The state estimate obtained from the positioning is then used

in Chapter 4, which introduces an approach to nonlinear control of the quadrotor. Chapter 5 presents the trajectory generation algorithms used to perform landing and flight, as implemented in terms of *state-machines*.

The following two chapters of the thesis, Chapters 6 and 7, present the numerical evaluation of the result and the following discussion respectively. Concluding remarks and suggestions for further work are finally presented in Chapter 8.

A technical description of the *CRAP* framework, developed for the implementation of this thesis, is appended.

Chapter 2

Monocular SLAM

In the interest of extracting positioning information from a video stream of a general, unknown, environment, a common approach is to use SLAM, *Simultaneous Localization And Mapping*, techniques. In the mathematical SLAM framework, features in the images are identified and tracked throughout time in the video stream, and a filtering solution is then traditionally applied to estimate the probability densities of the tracked landmarks' positions, as well as that of the camera position.

To determine the depth distance to a feature, stereo vision can be applied with which the correlation between two synchronized images is used, together with the known distance between the cameras, to calculate the image depth. As the distance to the tracked objects increases however, the stereo effect is lost and the algorithms' performance drops. There are several other issues to the stereo vision approach - increased cost, synchronization issues, computational load to name a few - which has led to the development of techniques to utilize the information which can be extracted from tracking movement in only a single camera to reconstruct a three-dimensional scene. The application of SLAM to this approach is referred to as *Monocular SLAM*, and two approaches are presented in this chapter.

Both approaches rely on feature detection in video frames. An extensive survey of available feature-detecting algorithms is given in [16].

2.1 Filtering Based Solutions

One novel approach for camera tracking, used by for instance [9] and [11], is to utilize a filtering framework, such as the EKF-SLAM or FastSLAM 2.0, and couple the feature and camera tracking in a time- and measurement update for each consecutive frame. The *Scenelib* library described in [9] uses a combination of the particle filter and the EKF for feature initialization and feature tracking respectively.

Common to the filter approaches is that as new features are detected, they are initialized and appended as states to the filter. As frames arrive, the change of position of each feature is measured and the filter is updated accordingly. Thus,

care must be taken to avoid misassociation of the features, as this leads to false measurements that will mislead the filter.

One advantage of the solutions with direct filtering is that it is quite trivial to extend the motion models or include other sensors to improve the precision, since the general algorithm utilize classical state-based time- and measurement updates.

2.2   Keyframe-based SLAM

A fundamentally different approach, presented and used for instance in [22], is to focus on collecting video frames of greater importance - keyframes - in which a large amount of features are detected. The camera's position is recorded at the time the keyframe is grabbed - see Figure 2.1 with caption - and the newly detected features are added to the active set. As new video frames arrive, features are reprojected and sought for in the new frame, giving a position offset from previously recorded keyframes which, by extension, gives the updated position of the camera.
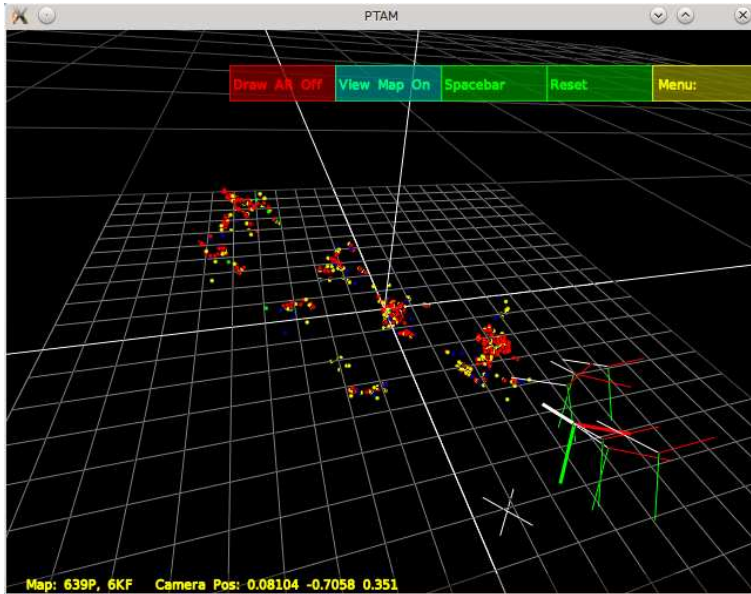
Figure 2.1. Keyframes containing features - here represented by dots - are recorded, and the camera's position at the time of the frame's capture is stored and visualized as coordinate axes. The offset from these positions is estimated from the features detected in the video stream. The thick coordinate system displays the camera's current position estimate. The colors of the features represent library-internal information on how the feature was detected.

2.3   The PTAM Library

PTAM - Parallel Tracking And Mapping is a software library for camera tracking developed for Augmented Reality (AR) applications in small workspaces [22]. It has only recently been applied for quadrotor state estimation [44], although as the library is intended for AR applications, the connection with the world's coordinate system is loose, as discussed in Chapter 3. Several libraries exist extending the functionality of the PTAM library [31].

Recently published algorithms - presented in [30] - surpass the library's performance using GPGPU[1] algorithms, although the performance of the PTAM library is nonetheless proven[2], and improves the usability over preceding libraries, such as *Scenelib*, by including a camera calibration tool, which is used to estimate the parameters describing the lens properties.
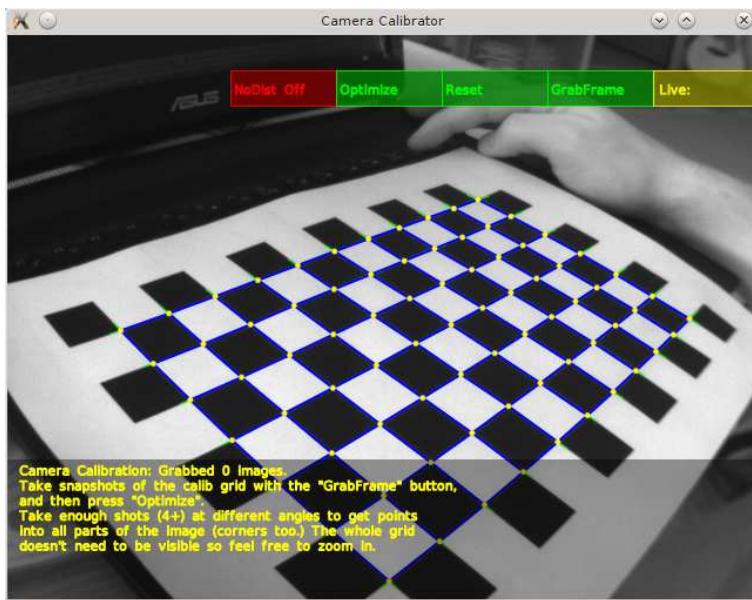
Figure 2.2. A checker-board pattern is used for calibrating the lens-specific parameters of the camera.

2.3.1   Operation

The PTAM library partitions the SLAM problem into a real-time tracking loop and a less time-critical optimization procedure on the collected keyframes.

In the tracking procedure, the PTAM library uses the keyframe technique described in Section 2.2, and randomly projects previously recorded features into the captured video frame, visualized in Figure 2.3. The selection of features to

---

[1]General-Purpose computing on Graphics Processing Units

[2]Examples of projects using PTAM are listed at http://ewokrampage.wordpress.com/projects-using-ptam/

re-project could potentially be improved, as discussed in Section 7.1, although this remains as future work.
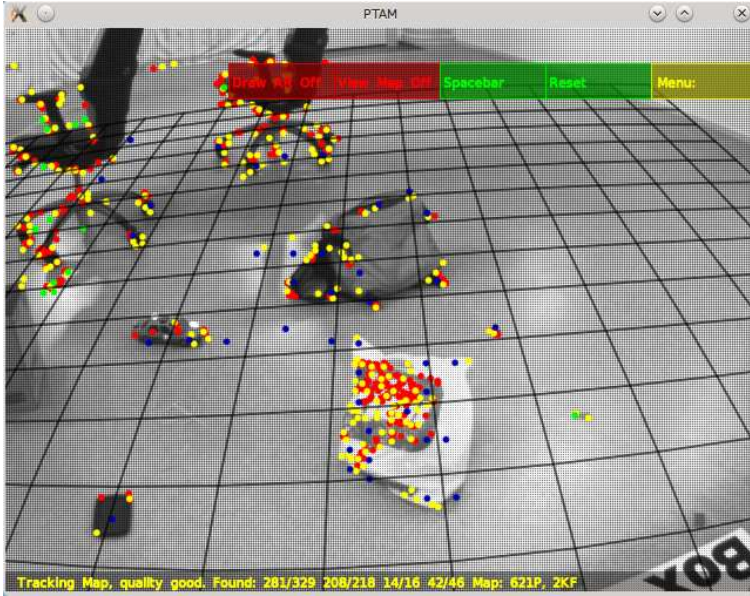
Figure 2.3. Features - visualized as dots - are projected into the image and sought for. The offset from the positions where the features were first detected then yield a position estimate.

As for mapping; Instead of continuously updating the keyframe's position estimates - as in the filtering solution - the PTAM library uses a less time-critical algorithm in a parallel processing thread[3]. The applied algorithm is known as Bundle Adjustment, and performs a nonlinear least-squares optimization over the available keyframes to calibrate their positions relative to each other, based on shared features [27]. The implementation utilizes the sparse structure of the optimization problem that is solved to make the solution computationally tractable [26].

2.3.2 Modifications to the Library

The PTAM library is originally interfaced with an included graphical user interface. Using the source-code, which is provided free for non-commercial use, the library was modified to be interfaced over a serial port and also extended with automatic initialization and full non-graphical use.

In the standard PTAM initialization procedure, an image is captured at a starting position. The camera is then translated sideways until the user deems the stereo initialization can be performed with good results. A second frame is

---

[3]Hence its name; *Parallel Tracking And Mapping*

then captured, and shared features are used to perform a stereo initialization, and extract the ground plane and a placement for the origin of the PTAM coordinate system. The tracked scene should be planar to retrieve a good estimate of the true ground plane, although the tracking will work regardless.

Figure 2.4. To create the initial map, PTAM is initialized with a set of two offset images. Features are tracked during the initialization to find the ground plane. The movement of the tracked features are visualized as lines in the image.

In the graphical user interface, the tracked features are visualized during the initialization process with lines in the camera image, shown in Figure 2.4. After the initial frame has been captured, the implementation of the automated initialization uses the length of these lines as a measure of when to finalize the initialization procedure. When the length of the line associated with the first[4] feature exceeds a given threshold, the second frame is captured and the stereo initialization algorithm is started. Should the initialization procedure fail - e.g. by loosing track of most features - the procedure will be restarted until a successful initialization has been performed. The procedure is initially triggered by a command from the serial port. The initialization is further described in Chapter 5.

2.3.3 Practical Use

The PTAM library is designed to be used a wide-angle lens. It is documented that performance drops should such not be available [22]. Even though the LinkQuad-mounted camera features a changeable lens, all currently available wide-angle

---

[4]First in the list of features from the first image that are still found in the latest frame.

lenses are fish-eye lenses, which the PTAM library does not currently support. It is possible to use the PTAM library with a standard lens, although tracking is less stable, and harder to extend to unexplored areas.

The tracking and initialization quality is also - as all video tracking - dependent on the amount of trackable features in the scene. While the library can recover from lost positioning, an insufficient amount of features can cause the initialization process to fail, or the tracking to be irreparably lost. The amount of tracked features is also dependent on the processing power available, and the library may, depending on available resources, have difficulties extending the map to unexplored areas.

Chapter 3

State Estimation

A central part of automatic control is to know the state of the controlled device. The system studied in this thesis - the LinkQuad - is in constant motion, so determining the up-to-date position is of vital importance. The filter implementation currently available on the LinkQuad is based on the complementary filter. While performance is adequate for current purposes, the current implementation is difficult to extend to the quite complex measurements received from the camera positioning discussed in Chapter 2. This chapter deals with the estimation of the states relevant for positioning and controlling the LinkQuad. Filter theory and notation is established in Section 3.1.

In this thesis, an Unscented Kalman Filter (UKF) and an Extended Kalman Filter (EKF) are studied. These filters both extend the linear Kalman filter theory to the nonlinear case. The UKF circumvents the linearization used in the EKF in an appealing black-box way, albeit it is more sensitive to obscure cases in the physical model, as detailed in the discussion in Chapter 7. The theory of the EKF and UKF is treated in Sections 3.1 and 3.2, respectively.

The application of a motion model to the filtering process is known to be highly beneficial, and is a central component in the Kalman filter framework. The motion model is used to predict the behavior of the system and may do so in more or less detail by varying the complexity of the model. A more complex model may provide superior filtering performance although at increased computational cost. The motion model used in this thesis, derived in Section 3.3, models the quadrotor in detail to obtain simulational validity and precise control.

As well as being modeled, the motions of the system are measured by the onboard sensors. A measurement y is related to the motion model by the sensor model h;

$$y(t) = h(x(t), u(t), t) \tag{3.1}$$

The models for the sensors used in this work are discussed in Section 3.4.

An implementation of a state estimating filter is commonly referred to as an *observer*.

3.1 The Filtering Problem

In the general filtering problem, the estimation of the systems states - in this case its position, orientation, velocity etc. - is expressed as the problem of finding the state estimate, $\hat{x}$, that in a well defined best way (e.g. with Minimum Mean Square Error, MMSE) describes the behavior of the system.

The evolution of a system plant is traditionally described by a set of differential equations that link the change in the variables to the current state and known inputs, u. This propagation through time is described by Eq. (3.2) (f_c denoting the continuous case). The system is also assumed to be subject to noise, $v(t)$, often assumed to be Gaussian and additive, with known covariance Q. This introduces an uncertainty associated with the system's state, which accounts for imperfections in the model compared to the physical system.

$$\dot{x}(t) = f_c(x(t), u(t), t) + v_c(t) \tag{3.2}$$

With numerical integration or analytical solutions, the discrete form of (3.2) is obtained, where only the sampling times are considered. The control signal, $u(t)$, is often assumed to be constant in the time interval, thereby allowing the next predicted state to be obtained straightforwardly, as in Eq. (3.3). This yields the estimate of $\hat{x}$ at the time t *given* the information at time $t-1$[1]. This motivates the notation used in this thesis: $\hat{x}_{t|t-1}$.

$$x_{t|t-1} = f(x_{t-1|t-1}, u_t, t) + v(t) \tag{3.3}$$

In the ideal case, a simulation of a prediction $\hat{x}$ would with the prediction model in (3.3) fully describe the evolution of the system. To be able to provide a good estimate in the realistic case, however, the measurements given from sensors measuring properties of the system must be fed into the estimation.

These measurements, y_t, are fused with the prediction using the *innovation*, ν.

$$\nu_t = y_t - \hat{y}_t \tag{3.4}$$

That is, the difference between the measured value and what would be expected in the ideal (predicted) case. To account for disturbances affecting the sensors, the measurements are also associated with an additive white Gaussian noise $w(t)$, with known covariance, R.

$$\hat{y}_t = h(\hat{x}_t, u_t, t) + w(t) \tag{3.5}$$

The innovation is then fused with the prediction to yield a new estimation of x given the information available at the time t [14].

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \nu_t \tag{3.6}$$

The choice of K_t is a balancing between of trusting the model, or trusting the measurements. In the Kalman filter framework, this balancing is made by tracking and weighing the uncertainties introduced by the prediction and the measurement noise.

---

[1]Note that no measurements have yet been made that provide information about the state at time t.

Algorithm 1 (Kalman Filter) *For a linear system, given predictions and measurements with known respective covariances Q and R, the optimal solution to the filtering problem is given by the forward recursion in Eqs. (3.7)-(3.8)[2].*

 Prediction update

$$\hat{x}_{t|t-1} = A\hat{x}_{t-t|t-1} + Bu_t \tag{3.7a}$$

$$P_{t|t-1} = AP_{t-1|t-1}A^T + Q \tag{3.7b}$$

 Measurement update

$$K = P_{t|t-1}H^T \left(HP_{t|t-1}H^T + R\right)^{-1} \tag{3.8a}$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K\left(y - H\hat{x}_{t|t-1}\right) \tag{3.8b}$$

$$P_{t|t} = P_{t|t-1} + KHP_{t|t-1} \tag{3.8c}$$

Because of the previously mentioned assumptions on the noise and the linear property of the innovation feedback, the Gaussian property of the noise is preserved in the filtering process. The system states can thus ideally be considered drawn from a normal distribution, as in Eq. (3.9).

$$x \sim \mathcal{N}\left(\hat{x}, P_{xx}\right) \tag{3.9}$$

Conditioned on the state estimate $\hat{x}$ and measurements before time k ($\mathcal{Y}^k$), the covariance of the sample distribution is defined as in Eq. (3.10).

$$P_{xx}(t|k) = E\left[\left\{x(t) - \hat{x}_{t|k}\right\}\left\{x(t) - \hat{x}_{t|k}\right\}^T |\mathcal{Y}^k\right] \tag{3.10}$$

As new measurements are taken, the covariance of the state evolves with the state estimate as in Eq. (3.11) [19].

$$P_{xx}(t|t) = P_{xx}(t|t-1) - K_t P_{\nu\nu}(t|t-1)K_t^T \tag{3.11}$$

$$P_{\nu\nu}(t|t-1) = P_{yy}(t|t-1) + R(t) \tag{3.12}$$

With known covariances, K can be chosen optimally for the linear case as derived in e.g. [14] and given in Eq. (3.13).

$$K_t = P_{xy}(t|t-1)P_{\nu\nu}^{-1}(t|t-1) \tag{3.13}$$

Note that (3.13) is another, albeit equivalent, way of calculating (3.8a). This is exploited in the derivation of the Unscented Kalman Filter, presented in Section 3.2.

 Although the Kalman filter is optimal in the linear case, no guarantees are given for the case where the motion- or measurement model is nonlinear. Several methods exist to give a sub-optimal estimate for the nonlinear cases, two of which will be studied here.

 A major problem with the nonlinear case is how to propagate the uncertainty, as described by the covariance, through the prediction and measurement models.

---

[2]As first suggested by Rudolf E. Kálmán in [20].

With the assumed Gaussian prior, it is desirable to retain the Gaussian property in the posterior estimate, even though this clearly is in violation with the nonlinear propagation, which generally does not preserve this property.

$$P_{xx}(t|t-1) = AP(t|t)A^T + Q_t \tag{3.14}$$

As the linear propagation is simple however - shown in Eq. (3.14) - the novel approach is to linearize the system to yield the linear model A in every timestep. This method is called the Extended Kalman Filter, and is considered the de facto standard nonlinear filter [18].

Algorithm 2 (Extended Kalman Filter) *The Kalman filter in Algorithm 1 is in the Extended Kalman filter extended to the nonlinear case by straightforward linearization where necessary.*
  Prediction update

$$\hat{x}_{t|t-1} = f\left(\hat{x}_{t-t|t-1}, u_t\right) \tag{3.15a}$$

$$P_{t|t-1} = AP_{t-1|t-1}A^T + Q \tag{3.15b}$$

Measurement update

$$K = P_{t|t-1}H^T \left(HP_{t|t-1}H^T + R\right)^{-1} \tag{3.16a}$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K\left(y - h(\hat{x}_{t|t-1})\right) \tag{3.16b}$$

$$P_{t|t} = P_{t|t-1} + KHP_{t|t-1}, \tag{3.16c}$$

where

$$A = \left.\frac{\partial f(x,u)}{\partial x}\right|_{x=\hat{x}_{t|t}}, \quad H = \left.\frac{\mathrm{d}h(x)}{\mathrm{d}x}\right|_{x=\hat{x}_{t|t-1}} \tag{3.17}$$

This linearization, some argue [19], fails to capture the finer details of highly nonlinear systems and may furthermore be tedious to calculate, analytically or otherwise. An alternative approach, known as the Unscented Kalman Filter, is therefore discussed in Section 3.2.

3.2 The Unscented Kalman Filter

The basic version of the Unscented Kalman Filter was proposed in [19] based on the following intuition [19]:

> *With a fixed number of parameters it should be easier to approximate a Gaussian distribution than it is to approximate an arbitrary nonlinear function.*

The approach is thus to propagate the uncertainty of the system through the nonlinear system and fit the results as a Gaussian distribution. The propagation is made by simulating the system in the prediction model for carefully chosen offsets from the current state called *sigma points*, each associated with a weight

Variable	Example value	Description
α	$0 \leq \alpha \leq 1$ (e.g. 0.01)	Scales the size of the sigma point distribution. A small α can be used to avoid large non-local nonlinearities.
β	2	As discussed in [17], β affects the weighting of the center point, which will directly influence the magnitude of errors introduced by the fourth and higher order moments. In the strictly Gaussian case, $\beta = 2$ can be shown to be optimal.
κ	0	κ is the number of times that the center point is included in the set of sigma points, which will add weight to the center point and scale the distribution of sigma points.

Table 3.1. Description of the parameters used in the SUT.

of importance. The selection scheme for these points can vary (and yield other types of filters), but a common choice is the *Scaled Unscented Transform* (SUT) [43]. The SUT uses a minimal set of sigma points needed to describe the first two moments of the propagated distribution - two for each of the n dimensions of the state vector and one for the mean. The sigma points and their associated weights are chosen according to Eqs. (3.19)-(3.20).

$$\mathcal{X}_0 = \hat{x}$$
$$\mathcal{X}_i = \hat{x} + \left(\sqrt{(n+\lambda)P_{xx}} \right)_i \qquad\qquad i = 1, \cdots, n$$
$$\mathcal{X}_i = \hat{x} - \left(\sqrt{(n+\lambda)P_{xx}} \right)_i \qquad\qquad i = n+1, \cdots, 2n \qquad (3.18)$$

$$W_0^m = \frac{\lambda}{n+\lambda} \quad W_0^c = \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta)$$
$$W_i^m = W_i^c = \frac{1}{2(n+\lambda)} \qquad i = 1, \cdots, 2n \qquad (3.19)$$

$$\lambda = \alpha^2(n+\kappa) - n \qquad (3.20)$$

The three introduced parameters, α, β and κ are summarized and described in Table 3.2. The term $\left(\sqrt{(n+\lambda)P_{xx}} \right)_i$ is used to denote the i'th column of the matrix square root $\sqrt{(n+\lambda)P_{xx}}$.

When the sigma points $\mathcal{X}_i$ have been calculated, they are propagated through the nonlinear prediction function and the resulting mean and covariance can be

calculated.

$$\mathcal{X}_{\mathbf{i}}^{+} = f(\mathcal{X}_{\mathbf{i}}, u, t) \qquad i = 0, \cdots, 2n \tag{3.21}$$

$$\hat{x} = \sum_{i=0}^{2n} W_i^m \mathcal{X}_{\mathbf{i}}^{+} \tag{3.22}$$

$$P_{xx} = \sum_{i=0}^{2n} W_i^c \left\{ \mathcal{X}_{\mathbf{i}}^{+} - \hat{y} \right\} \left\{ \mathcal{X}_{\mathbf{i}}^{+} - \hat{y} \right\}^T \tag{3.23}$$

For the measurement update, similar results are obtained, and Eqs. (3.26)-(3.27) can be connected to Eqs. (3.12)-(3.13) in the general filter formulation.

$$\mathcal{Y}_{\mathbf{i}} = h(\mathcal{X}_{\mathbf{i}}, u, t) \qquad i = 0, \cdots, 2n \tag{3.24}$$

$$\hat{y} = \sum_{i=0}^{2n} W_i^m \mathcal{Y}_{\mathbf{i}} \tag{3.25}$$

$$P_{yy} = \sum_{i=0}^{2n} W_i^c \left\{ \mathcal{Y}_{\mathbf{i}} - \hat{y} \right\} \left\{ \mathcal{Y}_{\mathbf{i}} - \hat{y} \right\}^T \tag{3.26}$$

$$P_{xy} = \sum_{i=0}^{2n} W_i^c \left\{ \mathcal{X}_{\mathbf{i}} - \hat{x} \right\} \left\{ \mathcal{Y}_{\mathbf{i}} - \hat{y} \right\}^T \tag{3.27}$$

As can be seen in the equations of this section, the UKF handles the propagation of the probability densities through the model without the need for explicit calculation of the Jacobians or Hessians for the system. The filtering is based solely on function evaluations of small offsets from the expected mean state[3], be it for the measurement functions, discussed in Section 3.4, or the time update prediction function - the motion model.

3.3   Motion Model

As the system studied in the filtering problem progresses through time, the state estimate can be significantly improved if a prediction is made on what measurements can be expected. The plausibility of each measurement can then be evaluated after how well they correspond to the prediction. With assumed Gaussian white noise distributions, this evaluation can be done in the probabilistic Kalman framework as presented in Sections 3.1-3.2, where the probability estimate of the sensors' measurements are based on the motion model's prediction.

---

[3]It is not, however, merely a central difference linearization of the functions, as stressed in [19].

3.3.1 Coordinate Frames

In the model of the quadrotor, there are several frames of reference.

North-East-Down (NED): The NED-frame is fixed at the center of gravity of the quadrotor. The NED system's $\hat{z}$-axis is aligned with the gravitational axis and the $\hat{x}$-axis along the northern axis. The $\hat{y}$-axis is selected to point east to form a right-hand system.

North-East-Down Earth Fixed (NEDEF): This frame is fixed at a given origin in the earth (such as the take-off point) and is considered an inertial frame, but is in all other aspects equivalent to the NED frame. All states are expressed in this frame of reference unless stated otherwise. This frame is often referred to as the "world" coordinate system, or "w" for short in equations as disambiguation from the NED system.

Body-fixed (BF): The body-fixed coordinate system is fixed in the quadrotor with $\hat{x}$-axis in the forward direction and the $\hat{z}$-axis in the downward direction, as depicted in Figure 3.2.

Propeller fixed: Each of the propellers are associated with their own frame of reference, P_{ri}, which tracks the virtual tilting of the thrust vector due to flapping, discussed in Section 3.3.3. Starting with P_{r1} being the forward propeller, the propellers are numbered counter-clockwise.

Camera frame: This is the frame which describes the location of the camera.

PTAM frame: This is the frame of reference used by the PTAM video SLAM library. This frame is initially unknown, and its recovery is discussed in Section 3.4.5.

IMU frame: This is the body-fixed frame in which the IMU measurements are said to be done. The origin is thus fixed close to the inertial sensors.

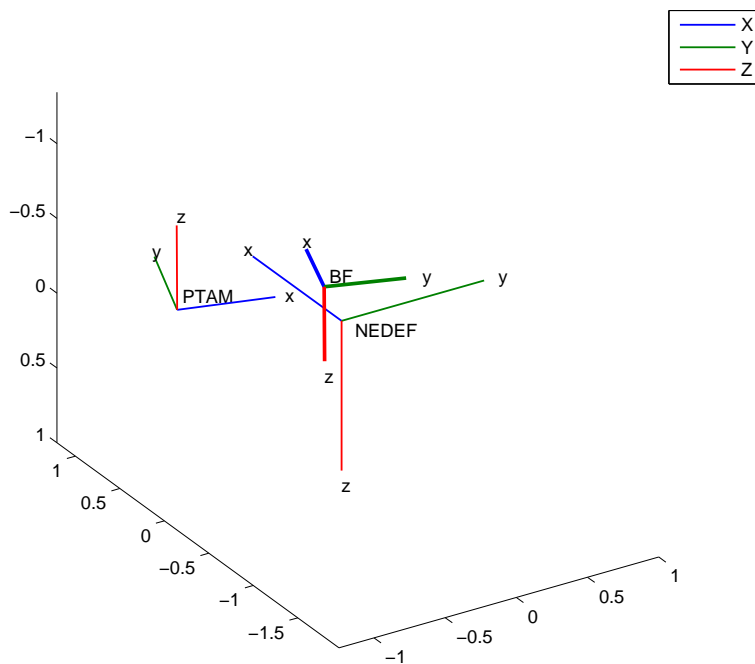The coordinate frames are visualized in Figures 3.1-3.2.

Figure 3.1. The NEDEF, body-fixed and the PTAM coordinate frames are of global importance.
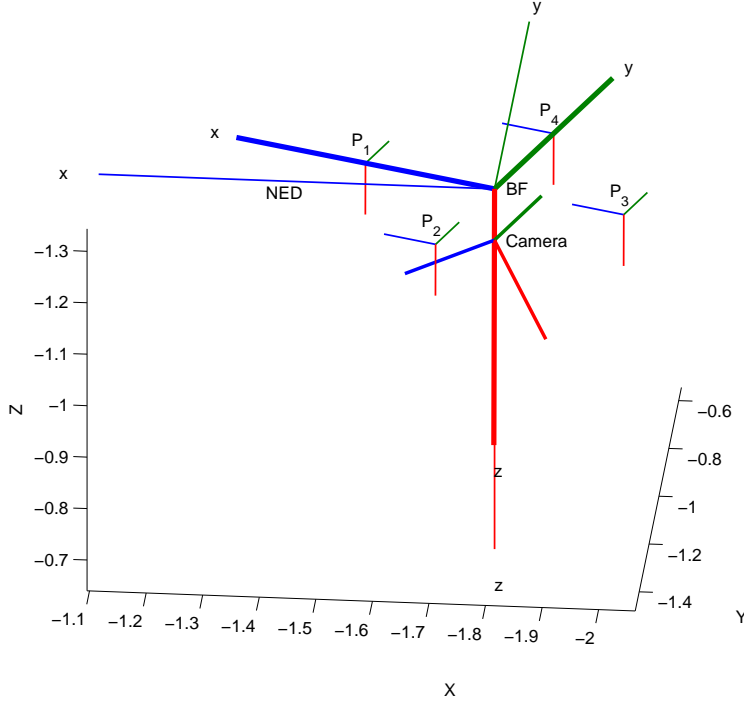
Figure 3.2. Locally, on the quadrotor, there are several coordinate frames used in the thesis. Here, the IMU frame coincides with the body-fixed frame.

The conversion between the reference frames are characterized by a transformation including translation and a three-dimensional rotation. Both the origin of the body-centered reference frames - the quadrotor's position - and the rotation of the body-fixed system are stored as system states.

The centers of each of the propeller fixed coordinate systems are parametrized on the (negative) height h and distance d from the center of gravity as follows

$$D_0 = (d, 0, h)^{BF} \tag{3.28}$$

$$D_1 = (0, -d, h)^{BF} \tag{3.29}$$

$$D_2 = (-d, 0, h)^{BF} \tag{3.30}$$

$$D_3 = (0, d, h)^{BF} \tag{3.31}$$

In the following sections, vectors and points in e.g. the NED coordinate systems are denoted x^{NED}. Rotation described by unit quaternions are denoted $R(q)$ for a quaternion q, corresponding to the matrix rotation [24][4] given by Eq. (3.32).

$$\begin{pmatrix} q_1^2 + q_i^2 - q_j^2 - q_k^2 & 2q_iq_j - 2q_1q_k & 2q_iq_k + 2q_1q_j \\ 2q_iq_j + 2q_1q_k & q_1^2 - q_i^2 + q_j^2 - q_k^2 & 2q_jq_k - 2q_1q_i \\ 2q_iq_k - 2q_1q_j & 2q_jq_k + 2q_1q_i & q_1^2 - q_i^2 - q_j^2 + q_k^2 \end{pmatrix} \tag{3.32}$$

---

[4][24] uses a negated convention of signs compared to what is used here.

   Rotation quaternions describing the rotation *to frame a from frame b* are commonly denoted q^{ab}. The characters a and b are, where unambiguous, replaced by the first character of the name of the reference frame of interest. Full transformations between coordinate systems - including rotation, translation and scaling - are similarly denoted $\mathcal{J}^{ab}$.

   The equations in this chapter is given in their time-continuous form to simplify the description of the forces. The equations are discretized in the implementation using numerical Euler integration (Eq. (3.33)) or Runge-Kutta integration (Eq. (3.34)), using an integration step of T.

$$X_{t+1} = X_t + T \cdot f(X, t) \tag{3.33}$$

$$X_{t+1} = X_t + \tfrac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{3.34a}$$
$$k_1 = Tf(X_t, t) \tag{3.34b}$$
$$k_2 = Tf(X_t + \tfrac{1}{2}k_1, t + \tfrac{1}{2}T) \tag{3.34c}$$
$$k_3 = Tf(X_t + \tfrac{1}{2}k_2, t + \tfrac{1}{2}T) \tag{3.34d}$$
$$k_4 = Tf(X_t + k_3, t + T) \tag{3.34e}$$

3.3.2   Kinematics

The motions of the quadrotor are described in terms of the change in position, ξ, and orientation, q^{wb}, by the relations in Eq. (3.35) [35].

$$\dot{\xi} = V \tag{3.35a}$$

$$\begin{pmatrix} \dot{q}_0^{wb} \\ \dot{q}_i^{wb} \\ \dot{q}_j^{wb} \\ \dot{q}_k^{wb} \end{pmatrix} = -\frac{1}{2}\begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & -\omega_z & \omega_y \\ \omega_y & \omega_z & 0 & -\omega_x \\ \omega_z & -\omega_y & \omega_x & 0 \end{pmatrix}\begin{pmatrix} q_0^{wb} \\ q_i^{wb} \\ q_j^{wb} \\ q_k^{wb} \end{pmatrix} \tag{3.35b}$$

   As a rotation described by quaternions require unit quaternion length, a normalization step also has to be applied in practice after the orientation has been updated.

3.3.3   Dynamics

The motions of the quadrotor can be fully explained by the forces and moments acting on the vehicle. Using the rigid-body assumption, Euler's extension of Newton's laws of motion, applied to the quadrotor's center of gravity ($\mathcal{G}$), yield Eqs. (3.36). These equations describe the acceleration and angular acceleration of the vehicle as related to the forces and moments acting on the vehicle.

$$\dot{V} = a_{\mathcal{G}}^w = R(q^{wb})\frac{1}{m}\sum F \tag{3.36a}$$

$$\dot{\omega} = R(q^{wb})I_{\mathcal{G}}^{-1}\sum M_{\mathcal{G}} \tag{3.36b}$$

The vector V contains the NEDEF velocities of the vehicle, while ω contains the body-fixed roll (ϕ), pitch (θ) and yaw (ψ) rates.

The main forces acting upon the quadrotor are the effects of three different components

- $\sum_{i=1}^{4} F_{ri}$ - Rotor thrust,

- F_g - Gravity,

- F_{wind} - Wind.

Of these, the gravity is trivially described with the gravitational acceleration and the total mass of the quadrotor as

$$F_g = mg \cdot \hat{z}^{NED} \tag{3.37}$$

The following sections will describe the rotor thrust and wind forces respectively. Additionally, other minor forces and moments are discussed in Section 3.3.3.

Rotor thrust

Each of the four propellers on the quadrotor induce a torque and a thrust vector on the system, proportional to the square of the propeller velocity - with sign depending on the direction of rotation and the propeller angle of attack. The rotational velocity of the propeller is directly influenced by the controller. It may as such be modeled as a first order system using the time constant τ_{rotor} with the reference velocity as input, as in Eq. (3.38). For testing purposes, or where the control signal, r_i is not available, Eq. (3.39) may be used instead, assuming constant propeller velocity.

$$\dot{\omega}_{ri} = \frac{1}{\tau_{rotor}} \left(\omega_{ri} - r_i \right) \tag{3.38}$$
$$\dot{\omega}_{ri} = 0 \tag{3.39}$$

Due to the differences in relative airspeed around the rotor blade tip as the blades move either along or against the wind-relative velocity, the lifting force on the blade will vary around a rotation lap. This unbalance in lifting force will cause the blades to lean and the direction of the thrust vector to vary with regards to the motions of the quadrotor.

This phenomenon is called *flapping*, and is discussed in e.g. [35]. The flapping of the rotors and the centrifugal force acting upon the rotating blades will result in that the tilted blade trajectories will form a cone with the plane to which the rotor axis is normal. These motions of the propellers add dynamics to the description of the quadrotor's motion which must be considered in a deeper analysis.

It is desirable, for the purpose of this thesis and considering computational load, to find a closed-form solution to the flapping equations. The resulting flapping angles and their impact on the thrust vectors can be described as in Eqs. (3.40a)-(3.41) [35, 36, 25].

The momenta induced by the propeller rotation and thrust are described in equations (3.40b)-(3.40c). It should be noted that the differing momenta from the varying speed of the propellers is the most important factor for the actuation of yaw rate control. This entails that the control - with such weak source of actuation - is expected to be slow. In the equations below, rotor velocities are given with positive orientation around the downwards pointing $\hat{z}$-axis. With the current LinkQuad setup, this results in positive rotational velocities for rotor 2 and 4. With all motors being single-directional, all thrust vectors are pointing upwards.

All equations in this section are given in the body-fixed coordinate system, for each rotor i, ri.

$$F_{ri} = C_T \rho A_r R^2 \omega_{ri}^2 \begin{pmatrix} -\sin\left(a_{1_s i}\right) \\ -\cos\left(a_{1_s i}\right)\sin\left(b_{1_s i}\right) \\ -\cos\left(a_{1_s i}\right)\cos\left(b_{1_s i}\right) \end{pmatrix} \tag{3.40a}$$

$$M_{Qi} = -C_Q \rho A R^3 \omega_{ri}|\omega_{ri}|\hat{z}^{\text{NED}} \tag{3.40b}$$

$$M_{ri} = F_{ri} \times D_i \tag{3.40c}$$

The equations for the flapping angles $(a_{1_s i}, b_{1_s i})$ are derived in [35, 36, 25], but are in (3.41) extended to include the velocity relative to the wind. $V_{ri(n)}$ denotes the nth element of the vector V_{ri}.

$$V_{\text{rel}} = V - V_{\text{wind}} \tag{3.41a}$$

$$V_{ri} = V_{\text{rel}} + \omega \times D_i \tag{3.41b}$$

$$\mu_{ri} = \frac{||V_{ri(1,2)}||}{\omega_{ri}R} \tag{3.41c}$$

$$\psi_{ri} = \arctan\left(\frac{V_{ri(2)}}{V_{ri(1)}}\right) \tag{3.41d}$$

$$\alpha_{si} = \frac{\pi}{2} - \arccos\left(-\frac{V_{ri}\cdot\hat{z}}{||V_{\text{ri}}||}\right) \tag{3.41e}$$

$$v_{1i} = \sqrt{-\frac{V_{rel}^2}{2} + \sqrt{\left(\frac{V_{rel}^2}{2}\right)^2 + \left(\frac{mg}{2\rho A_r}\right)^2}} \tag{3.41f}$$

$$\lambda_{ri} = \mu_{ri}\alpha_{si} + \frac{v_{1i}}{\omega_{ri}R} \tag{3.41g}$$

$$\begin{pmatrix} a_{1_s}i \\ b_{1_s}i \end{pmatrix} = \begin{pmatrix} \cos\left(\psi_{ri}\right) & -\sin\left(\psi_{ri}\right) \\ \sin\left(\psi_{ri}\right) & \cos\left(\psi_{ri}\right) \end{pmatrix} \begin{pmatrix} \frac{1}{1-\frac{\mu_{ri}^2}{2}}\mu_{ri}\left(4\theta_{twist} - 2\lambda_{ri}\right) \\ \frac{1}{1+\frac{\mu_{ri}^2}{2}}\frac{4}{3}\left(\frac{C_T}{\sigma}\frac{2}{3}\frac{\mu_{ri}\gamma}{a} + \mu_{ri}\right) \end{pmatrix}$$
$$+ \begin{pmatrix} \frac{-\frac{16}{\gamma}\left(\frac{\omega_\theta}{\omega_{ri}}\right) + \left(\frac{\omega_\psi}{\omega_{ri}}\right)}{1-\frac{\mu_{ri}^2}{2}} \\ \frac{-\frac{16}{\gamma}\left(\frac{\omega_\psi}{\omega_{ri}}\right) + \left(\frac{\omega_\theta}{\omega_{ri}}\right)}{1+\frac{\mu_{ri}^2}{2}} \end{pmatrix} \tag{3.41h}$$

$$C_T = \frac{\sigma a}{4} \left\{ \left(\frac{2}{3} + \mu_{ri}^2 \right) \theta_0 - \left(\frac{1}{2} + \frac{\mu_{ri}^2}{2} \right) \theta_{\text{twist}} + \lambda_{ri} \right\} \tag{3.41i}$$

$$C_Q = \sigma a \left[\frac{1}{8a} \left(1 + \mu_{ri}^2 \right) C_{D,r} + \lambda_{ri} \left(\frac{1}{6} \theta_0 - \frac{1}{8} \theta_{\text{twist}} + \frac{1}{4} \lambda_{ri} \right) \right] \tag{3.41j}$$

Wind

To describe the wind's impact on the quadrotor motion, a simple wind model is applied where the wind is modeled that imposes forces and moments on the quadrotor. The wind velocities in the filter are given in the NEDEF reference frame.

The wind drag force is calculated using equation (3.42), whereas the moments are given by equations (3.43). In this thesis, the moments acting on the quadrotor body (as opposed to the rotors) are neglected or described by moments imposed by the wind acting on the rotors.

$$F_{\text{wind}} = F_{wind,body} + \sum_{i=0}^{3} F_{wind,ri} \tag{3.42a}$$

$$F_{\text{wind,body}} = -\frac{1}{2} C_D \rho A V_{\text{rel}} ||V_{\text{rel}}|| \tag{3.42b}$$

$$F_{\text{wind},ri}^{BF} = -\frac{1}{2} \rho C_{D,r} \sigma A_r (V_{ri} \cdot e_{P_{ri}3}^{BF}) ||V_{ri} \cdot e_{P_{ri}3}^{BF}|| e_{P_{ri}3}^{BF} \tag{3.42c}$$

$$M_{\text{wind}} = M_{\text{wind,body}} + \sum_{i=0}^{3} M_{\text{wind},ri} \tag{3.43a}$$

$$M_{\text{wind,body}} \approx 0 \tag{3.43b}$$

$$M_{\text{wind},ri} = D_i^{BF} \times F_{\text{wind},ri}^{BF} \tag{3.43c}$$

The wind model applied in this thesis is a decaying model that tends toward zero if no measurements tell otherwise. The decaying model is given in Eq. (3.44) (ϵ being a small number).

$$\dot{V}_{\text{wind}} = -\epsilon \cdot V_{\text{wind}} \tag{3.44}$$

Additional Forces and Moments

Several additional forces act on the quadrotor to give its dynamics in flight. Some of these are summarized briefly in this section, and are discussed further in [5]. Unless where explicitly noted, annotation is similar to Section 3.3.3.

Symbol	Expression	Description	Unit
a	$\frac{\mathrm{d}C_L}{\mathrm{d}\alpha} \approx 2\pi$	Slope of the lift curve.	$\frac{1}{\mathrm{rad}}$
α_{si}	-	Propeller disk angle of attack.	rad
A_r	-	Rotor disk area.	m^2
c	-	Blade chord - the (mean) length between the trailing and leading edge of the propeller.	m
$C_{D,r}$	-	Propeller coefficient of drag.	1
C_L	-	Coefficient of lift.	1
C_T	Eq. (3.41i)	Coefficient of thrust. This is primarily the scaling factor for how the thrust is related to the square of ω_{ri}, as in Eq. (3.40a).	1
C_Q	Eq. (3.41j)	Torque coefficient. This constant primarily is the scaling factor relating the square of ω_{ri} to the torque from each rotor.	1
γ	$\frac{\rho a c R^4}{I_b}$	γ is the Lock Number [25], described as the ratio between the aerodynamic forces and the inertial forces of the blade.	1
I_b	-	Rotational inertia of the blade.	kgm^2
λ_{ri}	Eq. (3.41g)	λ_{ri} denotes the air inflow to the propeller.	1
R	-	Rotor radius.	m
ρ	-	Air density.	$\frac{\mathrm{kg}}{\mathrm{m}^3}$
σ	$\frac{\text{blade area}}{\text{disk area}}$	Disk solidity.	1
θ_0	-	The angle of the propeller at its base, relative to the horizontal disk plane.	rad
θ_{twist}	-	The angle with which the propeller is twisted.	rad
v_{1i}	Eq. (3.41f)	Induced velocity of propeller i.	$\frac{\mathrm{m}}{\mathrm{s}}$
$\omega_\phi, \omega_\theta, \omega_\psi$	-	The rotational, body-fixed, velocity of the quadrotor.	$\frac{\mathrm{rad}}{\mathrm{s}}$
ω_{ri}	-	The rotational velocity of propeller i.	$\frac{\mathrm{rad}}{\mathrm{s}}$
μ_{ri}	-	The normalized, air-relative, blade tip velocity.	1

Table 3.2. Table of symbols used in the flapping equations

Symbol	Expression	Description
A	-	3x3 matrix describing the area of the quadrotor, excluding the rotors.
C_D	-	3x3 matrix describing the drag coefficients of the quadrotor.
$C_{D,r}$	-	Propeller's coefficient of drag.

Table 3.3. Table of symbols used in the wind equations

Hub Force

$$C_H = \sigma a \left[\frac{1}{4a} \mu_{ri} C_{D,r} + \frac{1}{4} \lambda_{ri} \mu_{ri} \left(\theta_0 + \frac{\theta_{twist}}{2} \right) \right] \tag{3.45}$$

$$F_{\text{hub},i} = -C_H \rho A R^2 \omega_{ri}^2 \hat{x} \tag{3.46}$$

$$M_{\text{hub},i} = D_i \times F_{hub,i} \tag{3.47}$$

Rolling Moment

$$C_{\text{RM}} = -\sigma a \mu_{ri} \left[\frac{1}{6} \theta_0 + \frac{1}{8} \theta_{twist} - \frac{1}{8} \lambda_{ri} \right] \tag{3.48}$$

$$M_{\text{RM},i} = C_{\text{RM}} \rho A R^3 \omega_{ri}^2 \tag{3.49}$$

Ground Effect

As the vehicle gets close to ground, the wind foils of the propellers provide a cushion of air under the vehicle, giving extra lift.

$$F_{ri,\text{IGE}} = \frac{1}{1 - \frac{R^2}{16z^2}} F_{ri} \tag{3.50}$$

Gyro Effects and Counter-Torque

I_{rotor} is the propeller inertia.

$$\begin{pmatrix} \dot{\omega}_\theta \dot{\omega}_\psi (I_{yy} - I_{zz}) + I_{\text{rotor}} \dot{\omega}_\theta \sum_{i=0}^{4} \omega_{ri} \\ \dot{\omega}_\theta \dot{\omega}_\psi (I_{zz} - I_{xx}) + I_{\text{rotor}} \dot{\omega}_\theta \sum_{i=0}^{4} \omega_{ri} \\ \dot{\omega}_\theta \dot{\omega}_\phi (I_{xx} - I_{yy}) + I_{\text{rotor}} \sum_{i=0}^{4} \dot{\omega}_{ri} \end{pmatrix} \tag{3.51}$$

3.4   Sensor Models

This section relates the estimated state of the quadrotor to the expected sensor measurements, $\hat{y}$.

In UAV state estimation, it is common to include a GPS sensor, providing world-fixed measurements, to prevent drift in the filtering process. In this thesis, the GPS is replaced with a camera, which is discussed in detail in Section 3.4.5, following a description of the accelerometers, gyroscopes, magnetometers and the pressure sensor.

In the measurement equations in this section, a zero-mean Gaussian term e with known covariance is added to account for measurement noise. The Gaussian assumption may in some cases be severely inappropriate, but the Kalman filter framework requires its use.

3.4.1 Accelerometers

The accelerometers, as the name suggests, provides measurements of the accelerations of the sensor. In general, this does not directly correspond to the accelerations of the mathematical center of gravity used as center of the measured vehicle. This motivates a correction for angular acceleration and velocity with regards to the sensor's relative position from the center of gravity, $r_{\text{acc}/\mathcal{G}}$.

$$\hat{y}_{\text{acc}} = a_{\mathcal{G}} + \dot{\omega} \times r_{\text{acc}/\mathcal{G}} + \omega \times \left(\omega \times r_{\text{acc}/\mathcal{G}} \right) + e_{\text{acc}} \tag{3.52}$$

3.4.2 Gyroscopes

Gyroscopes, or rate gyroscopes specifically, measure the angular velocity of the sensor. Unlike acceleration, the angular rate is theoretically invariant of the relative position of the sensor and the center of gravity. However, gyroscope measurements are associated with a bias which may change over time. This bias term may be introduced as a state variable in the observer, modeled constant as in Eq. (3.54), leaving its adjustment to the observer's measurement update.

$$y_{\text{gyro}} = \omega + b + e_{\text{gyro}} \tag{3.53}$$

$$\dot{b} = 0 + e_{\text{bias}} \tag{3.54}$$

3.4.3 Magnetometers

Capable of sensing magnetic fields, the magnetometers can be used to sense the direction of the Earth's magnetic field and, from knowing the field at the current location, estimate the orientation of a vehicle.

$$y = R(q)m^e + e_m \tag{3.55}$$

The Earth's magnetic field can initialized at startup, or approximated using the World Magnetic Model[8], which for Linköping, Sweden, is given in Eq. (3.56).

$$m^e = \left[\begin{pmatrix} 15.7 & 1.11 & 48.4 \end{pmatrix}^T \right]^{NEDEF} \mu T \tag{3.56}$$

Magnetometer measurements are, however, very sensitive to disturbances, and in indoor flight, measurements are often useless due to electrical wiring, lighting etc. Thus, the magnetometers were not used for the state estimation in this thesis.

Parameter	Description	Value	Unit
L	Temperature lapse rate.	0.0065	$\frac{\text{K}}{\text{m}}$
M	Molar mass of dry air.	0.0289644	$\frac{\text{kg}}{\text{mol}}$
p_0	Atmospheric pressure at sea level.	101325	Pa
R	Universal gas constant.	8.31447	$\frac{\text{J}}{\text{mol·K}}$
T_0	Standard temperature at sea level.	288.15	K

Table 3.4. Table of Symbols used in the pressure equation, Eq. (3.57)

.

3.4.4 Pressure Sensor

The barometric pressure, p, can be related to altitude using Eq. (3.57) [32]. The pressure sensor, as shown in the validation, is inherently noisy and especially so in an indoor environment where air conditioning causes disturbances in the relatively small - and thus pressure sensitive - environments that are available indoors. The pressure sensor is also affected by propeller turbulence.

$$p = p_0 \left(1 - \frac{L \cdot h}{T_0} \right)^{\frac{g \cdot M}{R \cdot L}} + e_p \tag{3.57}$$

3.4.5 Camera

To estimate the position of the camera using the captured images, the PTAM camera positioning library presented in Chapter 2 is used. The main application of the PTAM library is reprojection of augmented reality into a video stream. Consistency between a metric world-fixed coordinate frame (such as the NEDEF-system used on the LinkQuad), and the, quite arbitrarily placed [22], internally used coordinate system is not vital for its intended operation, although to position the camera in the real world, it is. The transformation between the NEDEF coordinate system and the PTAM coordinate system thus has to be determined to attain useful measurements.

The measurements from the camera consists of the transform from PTAM's coordinates to the camera lens, in terms of

- Translation[5], X^{PTAM},

- Orientation, $q^{PTAM,c}$.

A rough estimate of the quality of the tracking is also provided as an enumerated representation of either *Good*, *Poor* or *Bad*. Since the coordinate system of PTAM is neither of the same scale nor aligned with the quadrotor's coordinate system, the affine transformation between the two must be estimated.

Since both the NEDEF and the PTAM coordinate frame is world-fixed, the transformation is ideally static and characterized by

---

[5]Notably, the translation is of arbitrary, initially unknown, scale.

- a translation T to the origin, $\varnothing_{\text{PTAM}}$,

- a rotation R by the quaternion q^{Pw},

- a scaling S by a factor s.

These are collected to a single transformation in Eq. (3.58), forming the full transformation from the global NEDEF system to the PTAM coordinate frame.

$$x^{\text{PTAM}} = \underbrace{S(s)R(q^{Pw})T(-\varnothing_{\text{PTAM}})}_{\triangleq \mathcal{J}^{Pw},\text{transformation from camera to PTAM}} x^{\text{NEDEF}} \qquad (3.58)$$

The offline case of this problem is partially studied in [15], whereas the method used in this thesis can be extended to the on-line case where no ground truth is available by introducing continuously improved states to the observer filter, using the first measurement to construct an initial guess.

Initialization

When the first camera measurement arrives, there is a need to construct the world-to-PTAM transformation. Since the PTAM initialization places the origin at what it considers the ground level, the most informed guess, without any further information about the environment, is to assume that this is a horizontal plane at zero height.

The orientation of the PTAM coordinate system is calculated, using quaternion multiplication, as in Eq. (3.59) from the estimated quadrotor orientation and the measurement in the PTAM coordinate frame, $q^{PTAM,c}$.

$$q^{Pw} = q^{PTAM,c}q^{cb}q^{bw} \qquad (3.59)$$

The quaternion q^{bc}, the inverse of q^{cb} of Eq. (3.59), describes the rotation from camera coordinates to body-fixed coordinates. With known camera pitch and yaw, Θ_c and Ψ_c respectively, this corresponds to four consecutive rotations, given in Eq. (3.60) as rotations around gives axes, the last two rotations accounting for the differing definitions between the PTAM library camera coordinate system ($\hat{z}$ upwards) and that used in this thesis ($\hat{z}$ downwards).

$$q^{bc} = \text{R}(\Theta_c, \hat{y}) \cdot \text{R}(\Psi_c, \hat{z}) \cdot \text{R}(\tfrac{\pi}{2}, \hat{z}) \cdot \text{R}(\tfrac{\pi}{2}, \hat{x}) \qquad (3.60)$$

To determine the distance to this plane according to the current estimation, Eq. (3.61) is solved for λ in accordance with Eq. (3.62).

$$\begin{cases} \varnothing_{\text{PTAM}} & = \xi + R(q^{wb})r_{\text{camera}/\mathcal{G}} + \lambda R(q^{wP})\frac{X^{PTAM}}{|X^{PTAM}|} \\ \varnothing_{\text{PTAM}} \cdot \hat{z} & = 0 \end{cases} \qquad (3.61)$$

$$\lambda = -\frac{\left(\xi + R(q^{wb})r_{\text{camera}/\mathcal{G}}\right) \cdot \hat{z}}{\left(R(q^{wP})\frac{X^{PTAM}}{|X^{PTAM}|}\right) \cdot \hat{z}} \qquad (3.62)$$
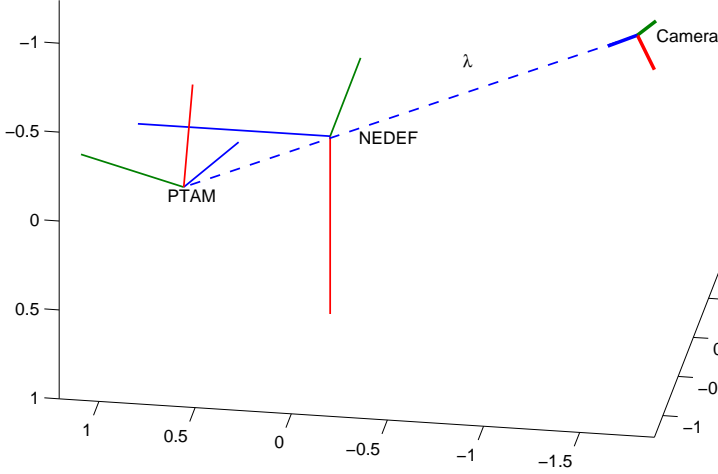
Figure 3.3. The PTAM coordinate system is assumed to be positioned at zero height. λ is calculated as the estimated real-world distance to the PTAM coordinate system.

With this definition, λ corresponds to the approximated distance from the camera to the PTAM origin, in the metric world-fixed coordinate system. By comparing this approximation with the distance measured in the PTAM coordinate system, an estimate for the scaling factor, s, is obtained through Eq. (3.63).

$$s = \frac{|X^{PTAM}|}{|\lambda|} \tag{3.63}$$

Together, the parameters derived in this section describe the transformation connecting the PTAM reference frame to the metric world. For further refinement, these parameters could be inserted into the global observer filter - introducing eight new states containing q^{Pw}, s and $\varnothing_{\text{cam}}$. Because the PTAM coordinate system is defined fixed in the global NEDEF coordinate system, the transform parameters would be static through the observer's time update.

Camera Measurements

The camera measurement update is made separate from the update of the other sensors, using the measurement equations in (3.64), expanding the equation derived in Eqs. (3.58) and (3.59).

$$\hat{X}^{\mathrm{PTAM}} = \mathcal{J}^{Pw}(\xi + R(q^{wb})r_{\mathrm{camera}/\mathcal{G}}) + e_{\mathrm{PTAM,X}} \tag{3.64a}$$

$$\hat{q}^{\mathrm{PTAM,c}} = q^{Pw}q^{wb}q^{bc} + e_{\mathrm{PTAM,q}} \tag{3.64b}$$

Teleportation

The PTAM tracking may sometimes exhibit a "teleporting" behavior. That is, although tracking is overall stable, the origin may sometimes be misassociated and placed at a new position as the tracking gets lost. To detect this, the measurements may be monitored for sudden changes in position. If a teleportation is detected, a reinitialization would be needed, either performing a new initial estimation, or utilizing the previous state to recognize the new pose of the origin. The teleporting behavior is currently detected in the thesis implementation using simple thresholding, as in Eq. (3.65), although no action is currently implemented to recover.

$$\left|X_t^{\mathrm{PTAM}} - X_{t-1}^{\mathrm{PTAM}}\right| \cdot s > \epsilon \tag{3.65}$$

The thresholded condition in Eq. (3.65) is scaled by the factor s from the transformation, to achieve a metric comparison with the configurable parameter, ϵ.

Chapter 4

Nonlinear Control

The state estimate from the algorithms discussed in Chapter 3 can be used to control the quadrotor's movements according to a desired reference.

In this thesis, a nonlinear controller is proposed to be applied to control the movements of the physical system, using a model of the system to calculate the signals of control to each of the motors driving the propellers.

The controller approach proposed in this thesis is extended from the Linear Quadratic (LQ) controller, the theory of which is presented in Section 4.1. The extension, related to the technique of *gain scheduling*, is discussed in Section 4.2.

The physical model of the system was derived in Section 3.3. In Section 4.3, this is further developed and adapted for compatibility as a model for the controller. The controller is interfaced by providing references for the NED-frame velocities and the, body-fixed, yaw rate. The controller outputs reference angular rates for each propeller.

4.1   The Linear Quadratic Controller

In this section, the theory of Linear Quadratic control is presented, considering the continuous linear plant in (4.1), with control signal u and reference r.

$$\dot{x} = Ax + Bu \tag{4.1a}$$

$$z = Mx \tag{4.1b}$$

$$e = z - r \tag{4.1c}$$

The basic LQ controller, described in e.g. [13], uses a linear state-space system model and costs on the states (Q) and control signals (R) respectively to calculate the control signals that would - given a starting state, a motion model and a constant reference - minimize the integral in Eq. (4.2). In the minimization, the cost for deviating from the reference scale quadratically, as the name suggests.

$$\mathcal{J} = \int\limits_{0}^{\infty} e^T(t)Qe(t) + u^T(t)Ru(t)dt \tag{4.2}$$

By varying the elements of the cost matrices Q and R respectively, the solution to the optimization will yield control signals that will control the system in a fashion that the amplitude of the control signals and the errors are balanced. By e.g. increasing the costs of the control signals, the system LQ controller will issue smaller control signals, protecting the engines but slowing the system down.

In the linear case, Eq. (4.2) can be solved analytically, resulting in a linear feedback gain, given in Eqs. (4.3)-(4.4).

$$u_t = -L\hat{x}_t \tag{4.3}$$

$$L = R^{-1}B^T S \tag{4.4}$$

S is the Positively Semi-Definite (PSD) solution to the Continuous Algebraic Riccati Equation (CARE) [13], stated in Eq. (4.5).

$$A^T S + SA + M^T QM - SBR^{-1}B^T S = 0 \tag{4.5}$$

To improve the reference following abilities of the controller, the reference may be brought into the control signal by a scaling matrix L_r, describing the inverse system kinematics. L_r is chosen so that the static gain of the system equals the identity matrix [13]. In the case of equal number of control signals as controlled states, Eqs. (4.6)-(4.7) are obtained.

$$u_t = -L\hat{x}_t + L_r r_t \tag{4.6}$$

$$L_r = \left[M(BL - A)^{-1}B \right]^{-1} \tag{4.7}$$

4.2 The State-Dependent Riccati Equation

Even though any system could be described at any point by its linearization, the linear nature of the LQ control poses a limitation in that a general system such as the one studied in this thesis - a quadrotor - will sooner or later leave the vicinity of the linearization point and no longer adhere to the physical circumstances valid there.

This will lead to sub-optimal control and possibly even to system failure. A common approach in nonlinear control is gain scheduling - to switch between pre-calculated control gains which has been calculated for selected linearization points.

The approach used in this thesis is closely related to gain scheduling, but instead of using pre-calculated gains, the linearization is done in-flight.

The problem of solving of the Riccati equation online is treated under the subject of *State-Dependent Riccati Equations*, SDRE's. While being computationally more expensive than the standard LQ formulation due to repeated solving of the Riccati equation, the need for finding valid linearization points is removed and more general problems can be treated. The basic formulation of the problem is covered in e.g. [37], and an extensive survey is presented in [7]. Implementation details are detailed and evaluated in [12, 2, 39].

The LQ theory can be extended to the nonlinear case by using the Taylor expansion of a general motion model, as shown in Eq. (4.8). This approximation is

exploited to form the general result of Eq. (4.9). Similar to several other modifications to the LQ methodology - such as Model Predictive Control (MPC) - the cost associated with the control signal may be applied relative to its current level, to avoid forcing all control signals to zero in the optimization problem that is solved. In each time-step, the local linearization of the motion model, as presented in Eq. (4.8), is used to solve the LQ-equations for Δu, the change in the control signal.

$$\dot{x} = f(x, u) \approx f(x_0, u_0) + \left.\frac{\partial f}{\partial x}\right|_{\substack{x = x_0 \\ u = u_0}} \underbrace{(x - x_0)}_{\Delta x} + \left.\frac{\partial f}{\partial u}\right|_{\substack{x = x_0 \\ u = u_0}} \underbrace{(u - u_0)}_{\Delta u} \quad (4.8)$$
$$\underbrace{\phantom{\left.\frac{\partial f}{\partial x}\right|_{x=x_0}}}_{A} \qquad \underbrace{\phantom{\left.\frac{\partial f}{\partial u}\right|_{u=u_0}}}_{B}$$

In the standard formulation of LQ control, the linearization has to be made at a stationary point (x_0, u_0), where $f(x_0, u_0) = 0$, to attain the nescessary property of linearity. In a more general formulation, it is possible to lift this constraint by using a homogeneous state [37], as in Eq. (4.9) to attain this property in the general case.

$$\dot{X} = \begin{bmatrix} \dot{x} \\ 0 \end{bmatrix} = \begin{bmatrix} A & f(x_0, u_0) - Ax_0 \\ 0 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} x \\ 1 \end{bmatrix}}_{X} + \begin{bmatrix} B \\ 0 \end{bmatrix} \Delta u \quad (4.9)$$

Eq. (4.9) is a linear system for which the ordinary LQ problem can be solved, using Eqs. (4.10)-(4.7).

The linearized output signal, Δu, is then added to u_0 to form the controller output, as in Eq. (4.10).

$$u = u_0 + \Delta u = u_0 - L\bar{X} + L_r r \quad (4.10)$$

A problem with the linearizing extension of the affine controller in Eq. (4.9), is that it introduces a non-controllable constant state, with an associated eigenvalue inherently located in the origin. This poses a problem to the traditional solvers of the Riccati equation, which expects negative eigenvalues to solve the problem numerically, even though the weights of (4.2) theoretically could be chosen to attain a well defined bounded integral.

The problem is circumvented by adding slow dynamics to the theoretically constant state, effectively nudging the eigenvalue to the left of the imaginary axis to regain full stability of the system. This guarantees that Eq. (4.2) tends to zero.

Eq. (4.9) is thus implemented as in Eq. (4.11).

$$\dot{X} = \begin{bmatrix} \dot{x} \\ 0 \end{bmatrix} = \begin{bmatrix} A & f(x_0, u_0) - Ax_0 \\ 0 & -\mathbf{10^{-9}} \end{bmatrix} \underbrace{\begin{bmatrix} x \\ 1 \end{bmatrix}}_{X} + \begin{bmatrix} B \\ 0 \end{bmatrix} \Delta u \quad (4.11)$$

4.3 Control Model

In Chapter 3, a physical model of the system was derived. To incorporate the information from the physical model into the governing control law, the model needs to be fitted into Eq. (4.9) by providing the Jacobi matrices with regards to x and u, while also removing states which will not be used in the controller.

The Jacobians of the system are acquired numerically by using central difference, as in Eq. (4.12).

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{4.12}$$

While all states are of importance to the dynamics of the quadrotor, only a subset of the states in x is used for control. New matrices thus need to be formed, containing the relevant states.

Notation

$\bar{x}$ denotes the rows in x that are used for control.

$\bar{x}^\dagger$ is used to denote the rows in x that are *not* used for control.

$\bar{A} = [\bar{A}^\square \bar{A}^\dagger]$ defines the separation of the square part ($\square$) of $\bar{A}$ with columns associated with the controller states, from the other columns ($\dagger$).

The new matrices should contain only the rows that are to be used for control. Extracting those rows from Eq. (4.8) yields Eq. (4.13).

$$\begin{aligned}
\dot{\bar{x}} = \bar{f}(x, u) \approx \quad & \bar{f}(x_0, u_0) + \bar{A}(x - x_0) + \bar{B}\Delta u \\
= \quad & \bar{f}(x_0, u_0) - \bar{A}^\square \bar{x}_0 - \bar{A}^\dagger \bar{x}_0^\dagger + \bar{A}^\square \bar{x} + \bar{A}^\dagger \bar{x}^\dagger + \bar{B}\Delta u \\
= \quad & \bar{f}(x_0, u_0) - \bar{A}^\square \bar{x}_0 + \bar{A}^\square \bar{x} + \bar{B}\Delta u
\end{aligned} \tag{4.13}$$

In the last equality of Eq. (4.13), it is assumed that the states not described in the controller are invariant of control and time, giving $\bar{x}_0^\dagger = \bar{x}^\dagger$. The results of Eq. (4.13) can be directly fitted into Eq. (4.9) to form the new matrices for the controller. The derivation of (4.13) is completely analogous in the discrete-time case.

The following states, from Chapter 3, are used in the control model:

- Velocities,

- Angular rates,

- Propeller velocities,

- Roll angle,

- Pitch angle.

Chapter 5

Finite State-Machines

State-machines are a modeling tool to describe the behavior of a system in terms of *states* and the transitions between them. Several paradigms of state-machines exist - Mealy and Moore being the notable forefathers in the field.

In projects related to the LinkQuad quadrotor, generic state-machine frameworks have been developed and researched in previous projects [29, 46]. On the LinkQuad, for the purpose of this thesis, a simple state-machine engine was implemented.

This state-machine is responsible for transitioning between the states predefined in an action sequence (a *mode*), ultimately providing reference signals for the controller. Each state represents an action, which is performed repeatedly until a transition condition is met. Figure 5.1 exemplifies the notation used in this chapter.
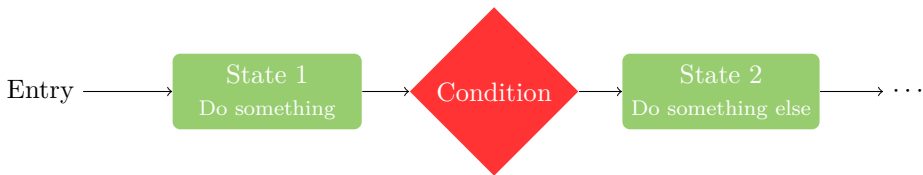
Figure 5.1. State-machine example mode. An active state is repeatedly invoked until a transition condition is met. The next state is then activated.

In this chapter, four basic modes are presented which were implemented in the time-frame of this thesis.

5.1 Hovering Mode

In the hover mode, the position of the quadrotor is recorded at the time of the activation, and three independent PID-controllers are initialized to generate reference velocities to the main controller, to keep the quadrotor at the place where the mode was first initialized.
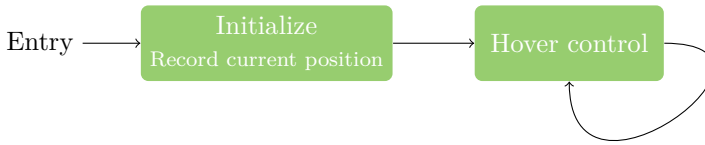
Figure 5.2. Hovering scheme. The position of the quadrotor is recorded when the mode is activated. The hover control state is then immediately activated to keep the quadrotor in this position.

5.2 PTAM Initialization Mode

By entering this mode, an automated initialization process is started to initialize the PTAM library and set up the transformation discussed in Chapter 3. Commands are sent to the PTAM module to initialize tracking, after which the quadrotor should be moved sideways for the stereo initialization to be performed by the PTAM library.

As part of the proposed modifications to the PTAM library, the initialization process is started remotely on command, and the video-captured motion is then monitored to determine either when the initialization has failed - in which case it is simply restarted - or when the camera has moved enough for a stable stereo initialization to be performed.
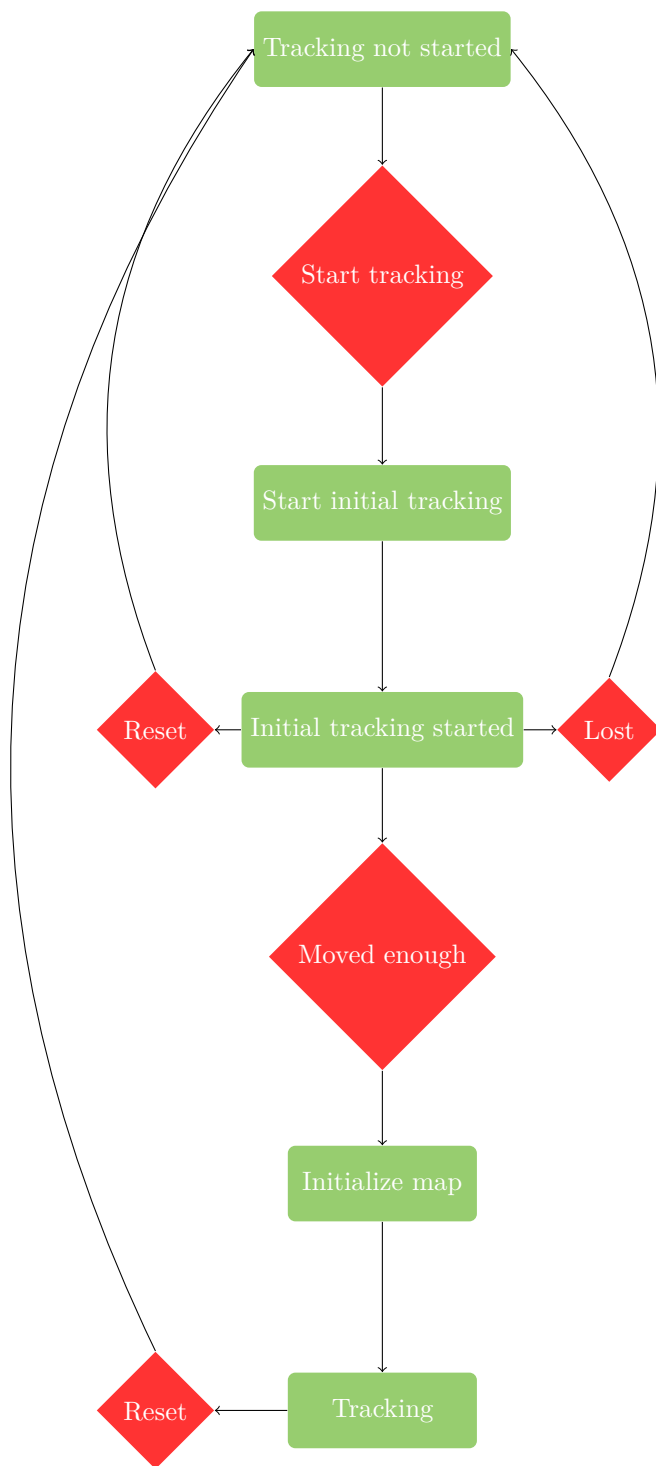
Figure 5.3. After the commands have been sent to start the PTAM initialization routine, the quality of the tracking is monitored until a stable map initialization is deemed to be possible.

5.3 Free Flight Mode

In the free flight mode, control reference signals for velocties and yaw rate are forwarded from the joystick reference provided over the serial interface from the user interface.

Reference velocities

Entry ⟶ Control
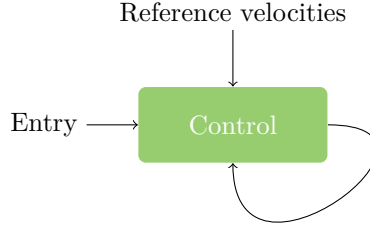
Figure 5.4. Free flight scheme. The free flight mode merely forwards controller reference signals received from the user interface.

5.4 Landing Mode

There have been several studies of autonomous quadrotor landing, e.g. [28, 6]. In [6] a landing scheme is proposed which is closely related to that which is proposed in this thesis. The algorithm used can be summarized in the following steps:

- Detection of landing site,

- Refinement of landing site position estimate,

- Descent on landing site,

- Landing detection.

In the landing site detection phase, the environment is searched for a suitable landing place. In [6], landing is then performed on an elevated surface which is detected using video processing. After the landing area has been located, the position of the landing site - relative to the quadrotor - is filtered to increase the confidence of the position estimate.
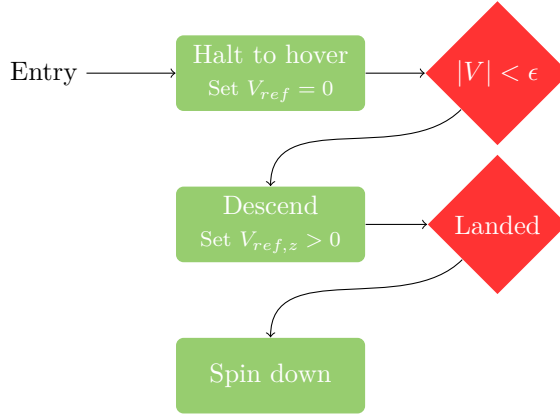
Figure 5.5. Landing scheme. The landing mode consists of three stages to perform and detect landing.

While the landing site position estimate converges, the quadrotor is moved to a position above the landing site as preparation for the Descend phase, where the quadrotor lowers until landing has been detected, using the camera feedback and other sensors to stabilize the descent.

The landing site detection of the algorithm is not covered by this thesis, but the other steps of the described landing scheme is implemented as described by Figure 5.5.

5.4.1   Landing Detection

In the landing detection, it must be recognized that the estimated position - initialized at the starting point - is not necessarily consistent with the true Height Over Ground (HOG) at the landing site. It is thus necessary to implement a more robust method to detect the completion of the landing procedure than simply halting at zero height. The proposed approach is to use the observer's estimate to determine when the movement has stopped; that is, when the quadrotor has reached ground. Detection theory, as discussed in e.g. [41, 33], provides several tools for detecting the nonlinear event that the quadrotor can descend no further.

In the physical model presented in Chapter 3, two terms is of specific interest for the detection. The first - and the obvious - is the altitudinal velocity. When sensor measurements pull this term towards zero, this is a first indication that the quadrotor has stopped. When the sensor measurements indicate a halt, the observer - whose motion model is oblivious to the forces imposed by the ground contact - will explain the lack of movement by a drastic increase in the estimated vertical wind velocity. This estimated state - the second of interest - is easily monitored and could be further filtered to increase detection confidence, or simply thresholded to detect landing.

Chapter 6

Experimental Results

This chapter contains results evaluating the algorithms presented in the previous chapters. The model is verified against data collected on the LinkQuad, as well as ground truth from the Vicon motion tracking system available. Ground truth was used only for initializing the model, and all test were run off-line on recorded sensor data and video feed.

6.1 Experimental Setup

The data used for the evaluation of the model and the filter in this chapter was recorded on the LinkQuad quadrotor in the UASTech Vicon Lab at Linköping University, Sweden. Ground truth data was recorded using the Vicon tracking system at a rate of 10 Hz, while sensors were sampled at 500 Hz and logged on-board the LinkQuad. For the dataset used in this Chapter, camera data was collected at a rate of 30 Hz during 20-second bursts after which the data had to be written to memory. The camera was tilted approximately 30 degrees downwards from the horizontal body-fixed plane of the quadrotor, giving an overview of the cluttered floor as seen in Figure 6.1. The camera settings were tuned to minimize the disturbance from light-sources and the infrared light used by the Vicon system.

The LinkQuad was then manually moved by hand to resemble flight conditions while recording sensor data, synchronized with video frames and reference Vicon data.

Figure 6.1. To provide visual features for the PTAM library to detect, the testscene was cluttered with objects.

After the validation of the model, it was then used for simulated control and landing, presented in Section 6.4, to validate the proposed control scheme. For the validation, simulated wind, random Gaussian system noise and random Gaussian measurement noise was injected into the system.

6.2   Model Verification

The verification of a complex model is best done in small parts. It is however, with the model given in Chapter 3, difficult to evaluate each equation individually due to the couplings of the model. Instead, the verification is performed by evaluating a full test-flight with recorded data, using one dataset for calibrating sensor covariance and a second for validating the motion model.

For each sensor the predicted and the measured values are compared, and the residuals - the difference between the two - are studied and fitted to a normal Probability Density Function, *PDF*.

6.2.1   Accelerometers

As most of the modeling in Chapter 3 concerns the forces acting upon the quadrotor, the accelerometers provide an interesting measure of the quality of the model. It should be noted that parameters were set to reasonable values, but no parameter tuning was performed on the motion model in the scope of this thesis. As depicted in Figure 6.2, the model leaves clearly trended residuals, not least in the horizontal directions where the model does very little. From the Figures 6.3 and 6.4, it

can be seen that the model does have beneficial effects for the estimation, yielding predictions with lesser residuals than for instance a constant velocity model would. The residuals of Figure 6.3 are centered around zero due to the compensation for sensor bias, which is relevant to reduce drift in the system, and exhibit a fairly good match to the normal PDF in Figure 6.4.
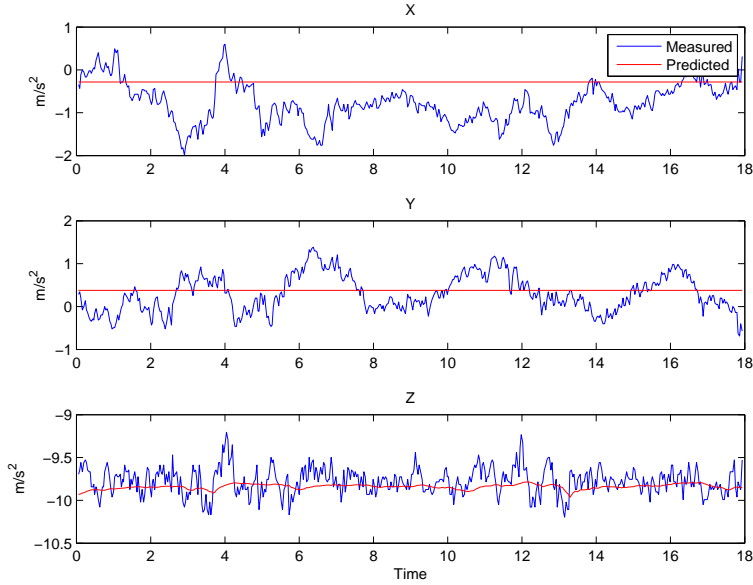

Figure 6.2. Measured and predicted accelerations in the NEDEF system. The motion model does not describe the horizontal motion although does - even in its roughly tuned state - explain some of the altitudinal acceleration.
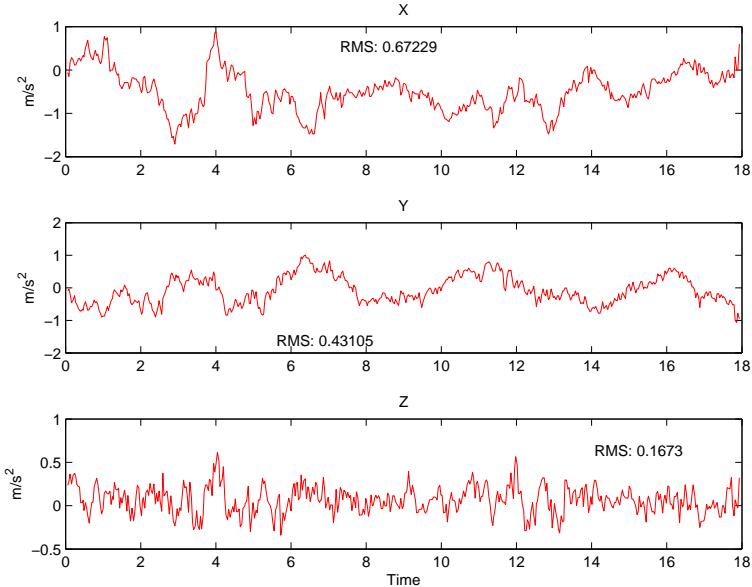
Figure 6.3. Residuals between measured and predicted accelerations. A zero mean is relevant to reduce drift in the system.
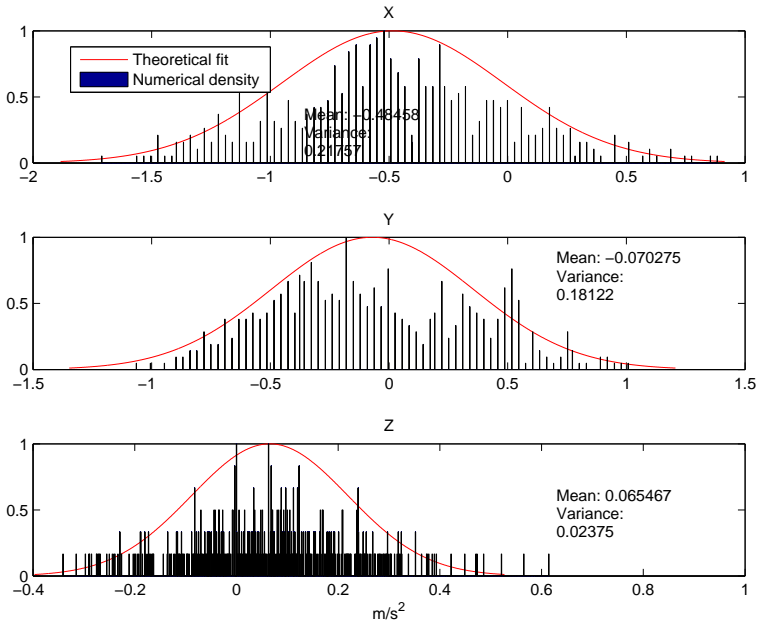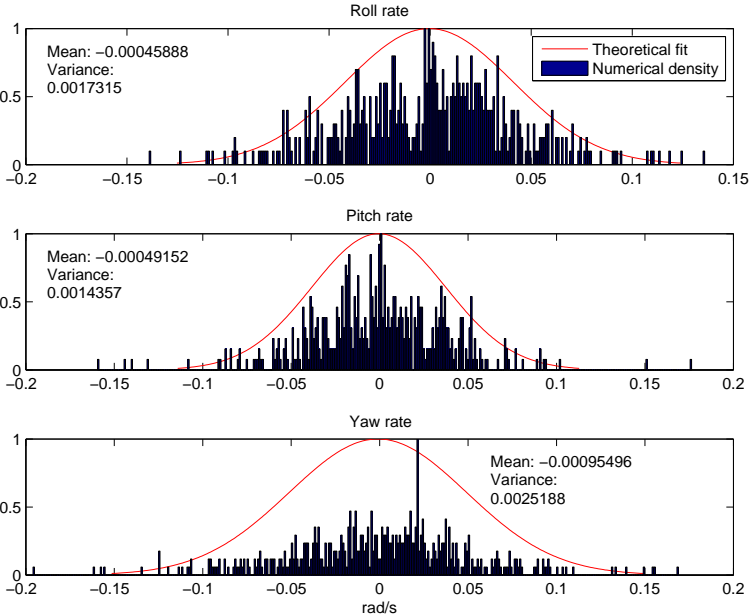

Figure 6.4. Accelerometer residuals fitted to a normal PDF. Both theoretical and numerical values have been normalized to a maximum height of one.

6.2.2 Gyroscopes

It is clear, from Figure 6.5, that the model describes the angular velocity of the quadrotor well. The residuals, presented by Figures 6.6 and 6.7 display a behavior which is adequately well described by a random, normally distributed, variable, which is expected from the standard Kalman filter framework.
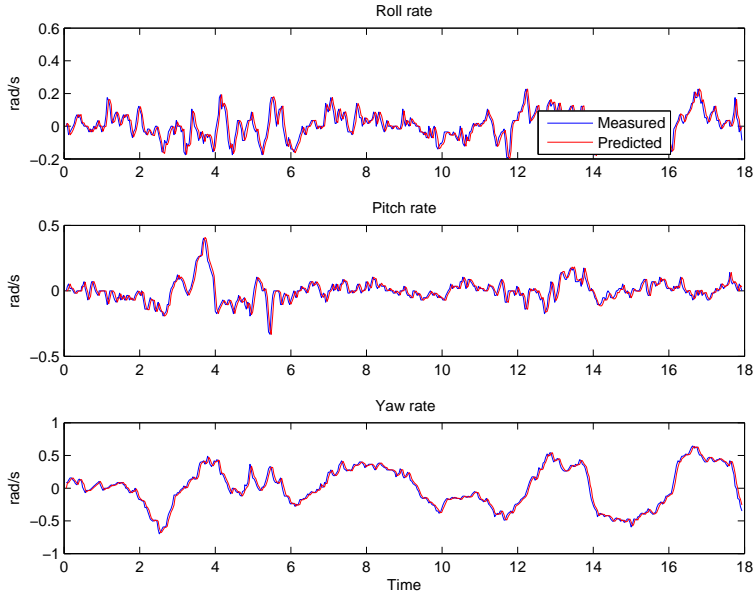
Figure 6.5. Measured and predicted angular rates, in the body-fixed coordinate frame.
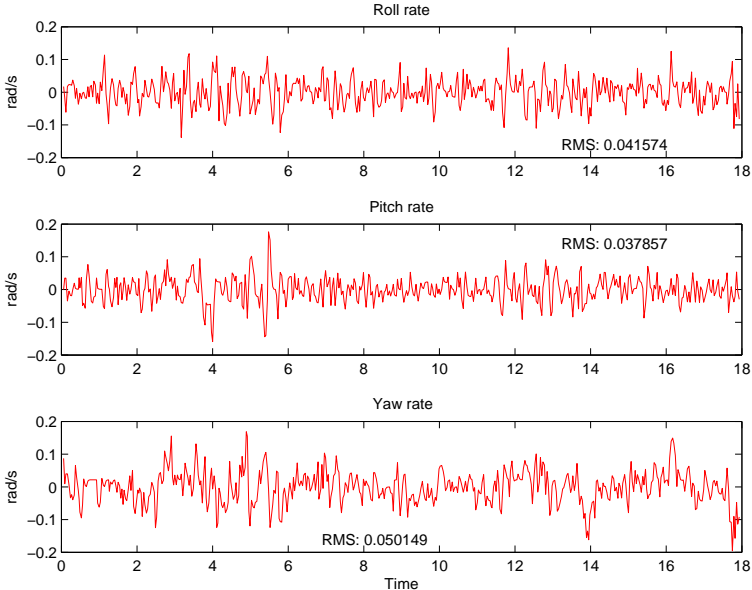
Figure 6.6. Residuals between measured and predicted angular rates.

Figure 6.7. Gyroscope residuals fitted to a normal PDF. Both theoretical and numerical values have been normalized to a maximum height of one.

6.2.3   Pressure Sensor

Pressure sensors are associated with a great amount of noise, as clearly seen in Figure 6.8. While the residuals, Figures 6.9 and 6.10, does not exhibit any obvious trends, noise does spill into the positioning with the current tuning, currently adding little contribution to the state estimation. This may be improved by further filter tuning, although indoor use of pressure sensors is, like magnetometers, known to be problematic due to air conditioning and other sources of pressure changes. To remove the noise induced into the filter by the pressure sensor, it was removed from the final filtering evaluation.
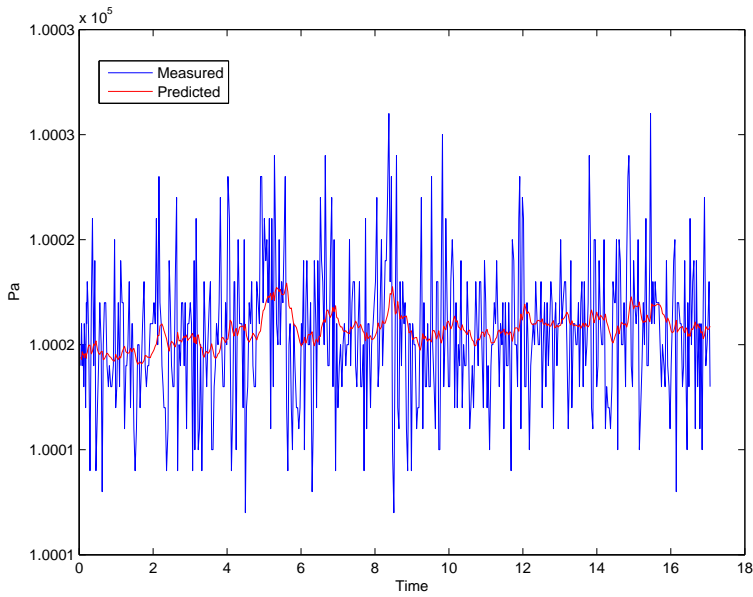
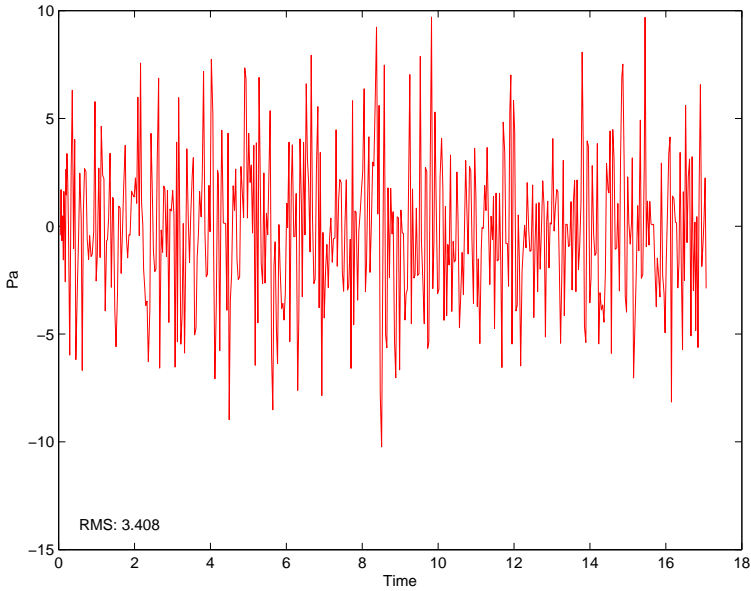**Figure 6.8.** Measured and predicted pressure.

Figure 6.9. Residuals between measured and predicted pressure.

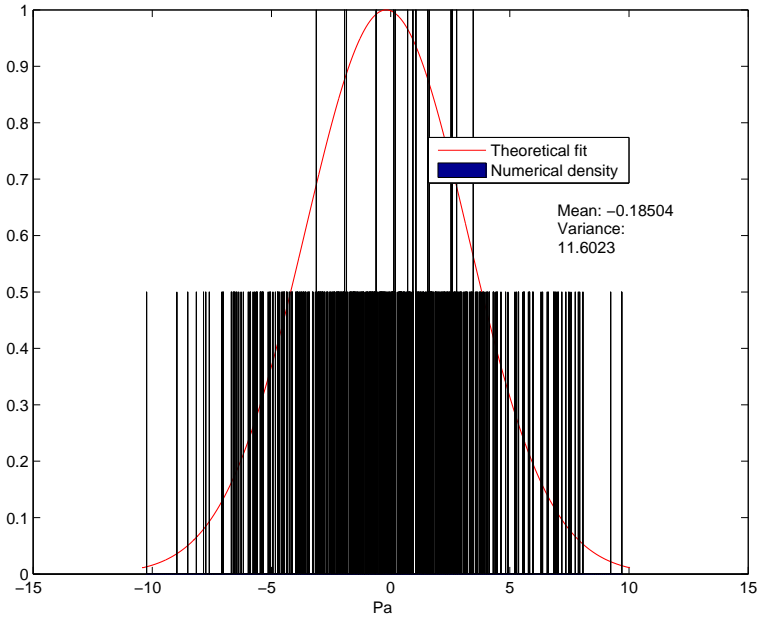**Figure 6.10.** Pressure residuals fitted to a normal PDF. Both theoretical and numerical values have been normalized to a maximum height of one.

6.2.4 Camera Positioning

The camera tracking, shown in Figures 6.11-6.13, exhibit very good stability and performance, and significantly add to the filter performance. The absolute positioning provided by the camera counters the drift in derived observer states, and provides accurate measurements of orientation angles and position, as exhibited in Figure 6.11. Because of potential errors in the world-to-PTAM transformation, the orientation - which is independent of the scale - is often the most reliable of the two measured quantities. It should be noted that the position results, especially in Figures 6.12-6.13 should be scaled by a factor - for this data approximately 3 - to correspond to metric quantities.

When the PTAM library is initialized, it tries to determine the ground plane. It is thus possible to verify, in Figure 6.14, the initialization process and world-to-PTAM transformation by confirming that the $\hat{z}$-axes ($\hat{z}^{NEDEF}$ and $\hat{z}^{PTAM}$) are approximately parallel. The slight tilting observed in Figure 6.14 is caused by a misplacement of the ground plane in the initialization process, which is also visible in Figure 2.3. Since the pose at the time of initialization is known, this tilting is compensated for in the calculated transformation, making the positioning less sensitive for PTAM initialization errors. Exact positioning on the moment of initialization is still of importance, however.

In Figure 6.11, a slight bias can be noted in the altitudinal positioning. This is due to the discrepancies between the PTAM library's ground plane position and the true ground plane.
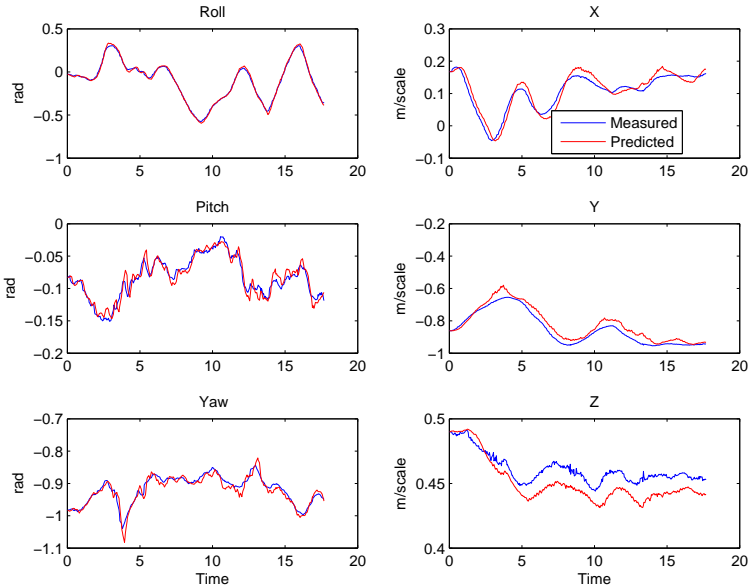

Figure 6.11. Measured and predicted angles and positions, in the PTAM coordinate frame. Positional measures should be multiplied by a scale of approximately 3 for metric comparison.
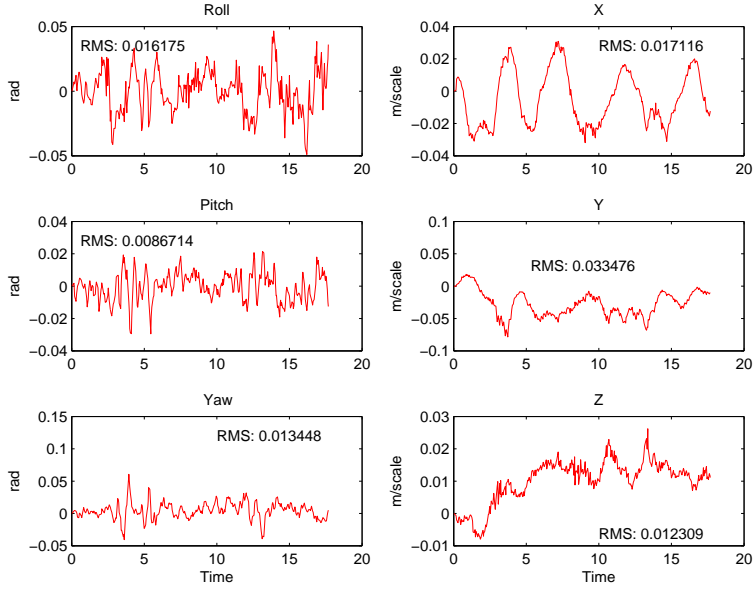
Figure 6.12. Residuals between measured and predicted angles and positions, in the PTAM coordinate frame. Positional measures should be multiplied by a scale of approximately 3 for metric comparison.
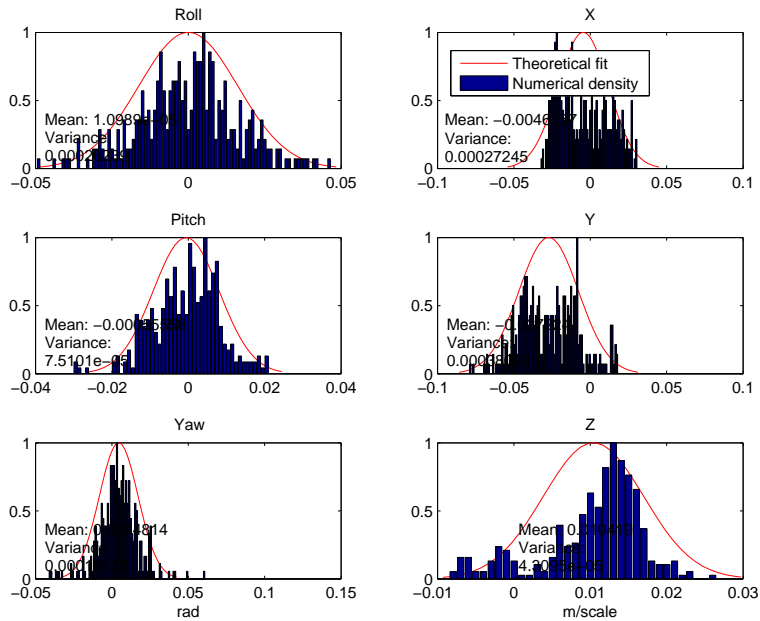
Figure 6.13. Residuals fitted to a normal PDF. Both theoretical and numerical values have been normalized to a maximum height of one. Positional measures should be multiplied by a scale of approximately 3 for metric comparison.
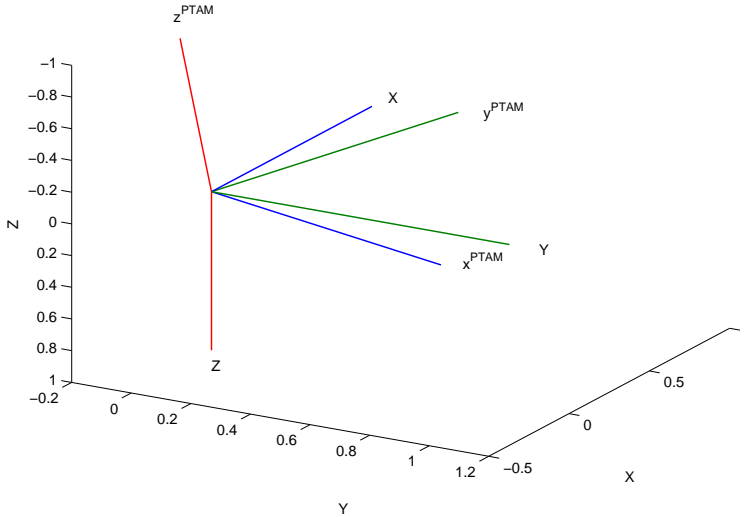
Figure 6.14. The PTAM coordinate system is somewhat askew from its theoretical orientation, with the $\hat{z}$-axis orthogonal to the ground. This can also be seen in Figure 2.3, which is captured from the same dataset.

6.3   Filtering

The filter implementation was evaluated with recorded data as described in Section 6.1. Due to model stability issues, as discussed in Chapter 7, the EKF-filter was selected for evaluation.

6.3.1   Positioning

The position estimate generally exhibit very good performance relative to the ground truth. This reflects the fact that the position states are observed, in the state estimation, more or less directly by the camera, which exhibit stable positioning in the validation.

In the plot of the altitudinal positioning in Figure 6.15, the estimates from the pre-existing complementary filter - moved to share the initial conditions of the Kalman filter - was added for comparison. The complementary filter exhibits problems associated with the pressure sensor, while the camera-based positioning is stable throughout the validation.
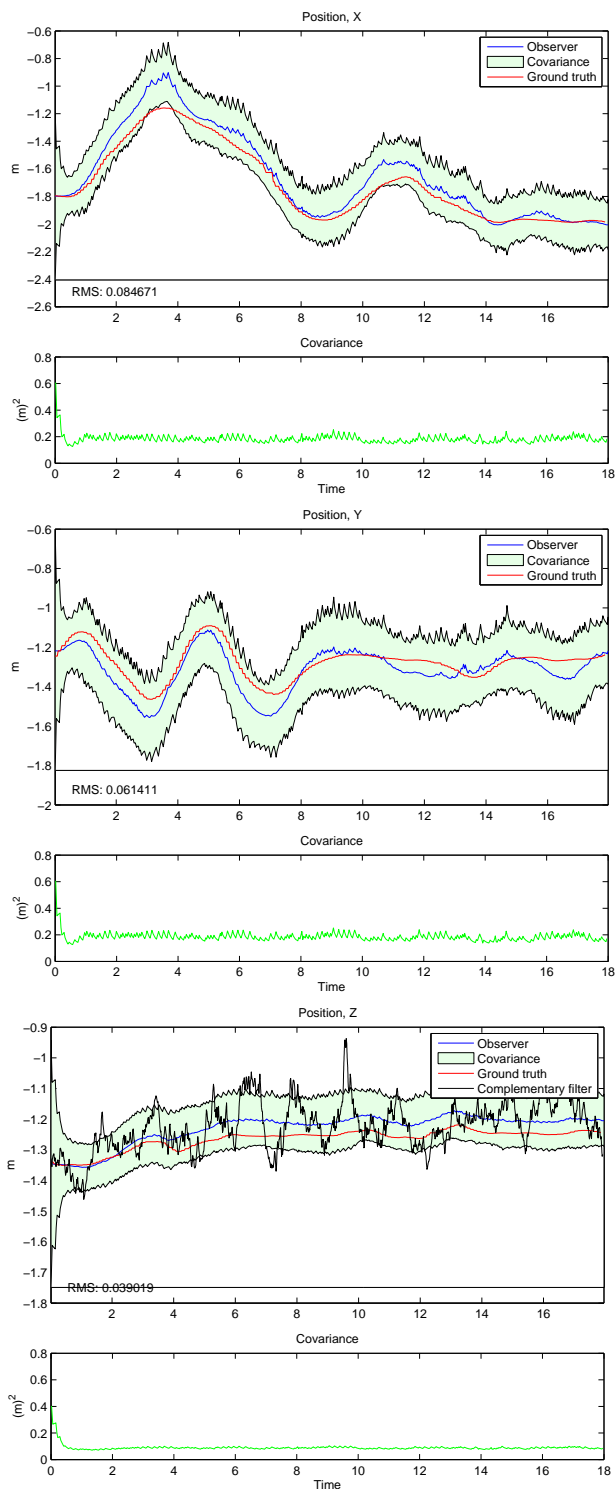
Figure 6.15. Positioning in the horizontal directions well corresponds to the ground truth, thanks to the camera positioning. For the altitude, the estimation of the pre-existing complementary filter is added for comparison, its starting point adjusted to produce comparable plots.

6.3.2   Velocities

The velocities, being closely coupled with the camera observed position, also exhibit good performance (Figure 6.16). There are shortcomings to the estimation's horizontal precision, although this could probably be significantly improved with further filter tuning and the availability of control signals to the motion model.
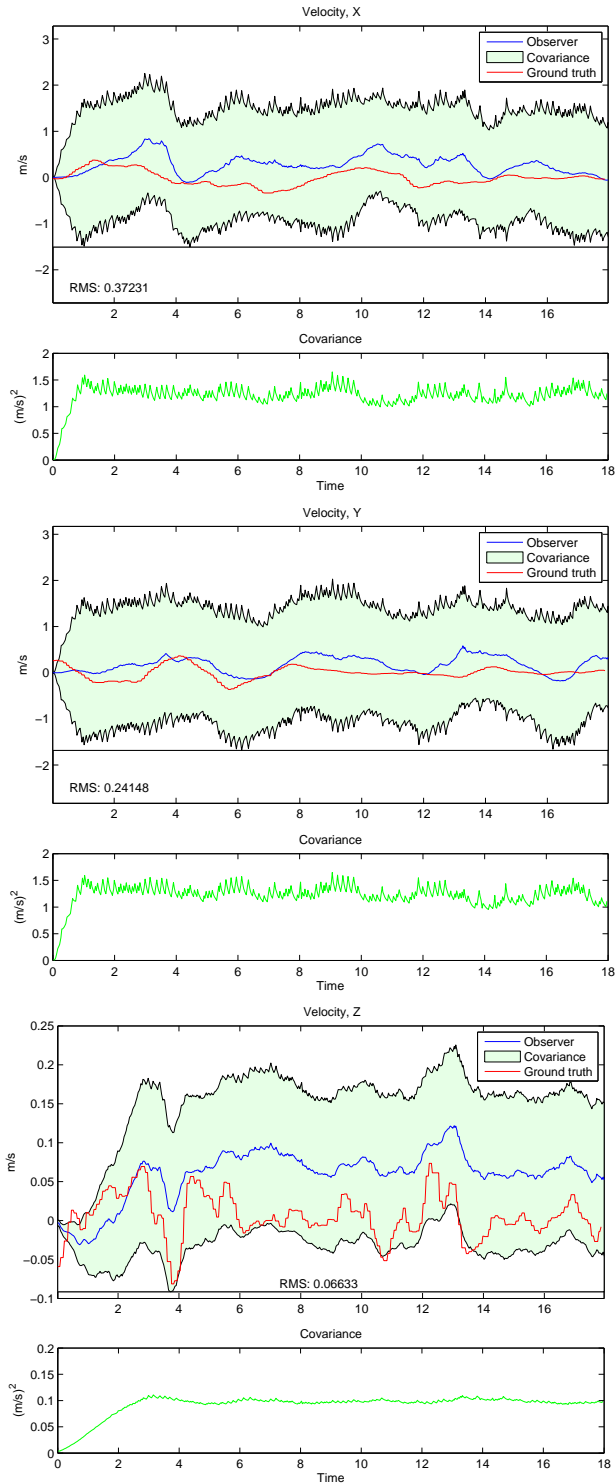
Figure 6.16. Velocity estimates are generally adequate, but with more tuning, the results are likely to improve.

6.3.3 Orientation, Rotational Velocity and Gyroscope Bias

Along with the position, the orientation is estimated from the camera, yielding notable precision, as seen in Figure 6.17.

The bias of the gyroscopes is removed during the initialization process. Since the time-frame of the tests were far less than the time expected to detect a change in the bias, these should thus be estimated to zero, as verified in Figure 6.19.

As noted in Section 6.2.2, the filtering of the rotational velocities of the quadrotor body, exhibited with their associated covariance in Figure 6.18, correlates very well to the measurements. In Figure 6.20, the Kalman filter performance is compared to the previously existing complementary filter. It can be seen that the prediction update of the Kalman filter improves the phase of the filter, although towards the end of the dataset, they are of comparable performance.
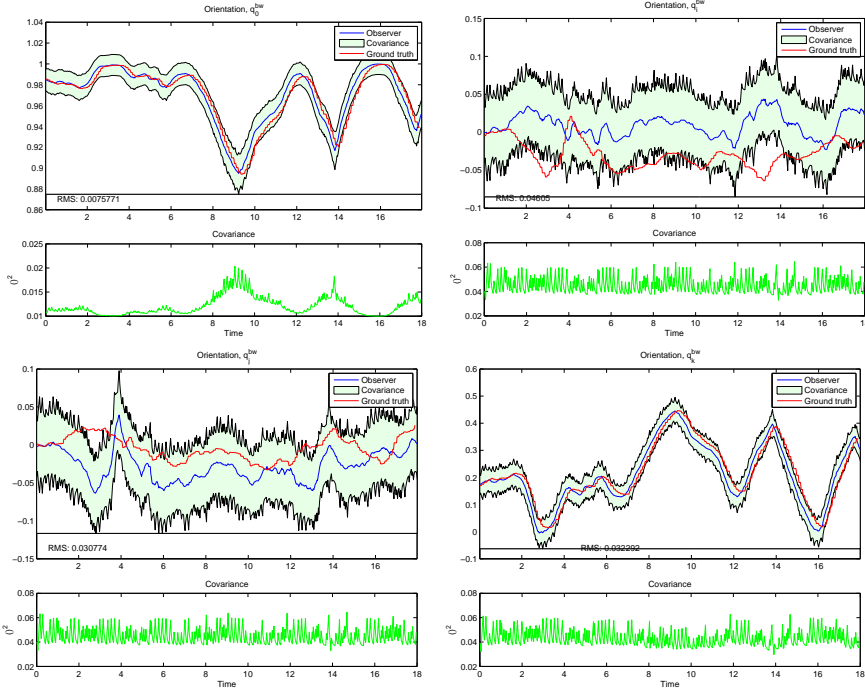

Figure 6.17. The orientation of the quadrotor was estimated with good accuracy.
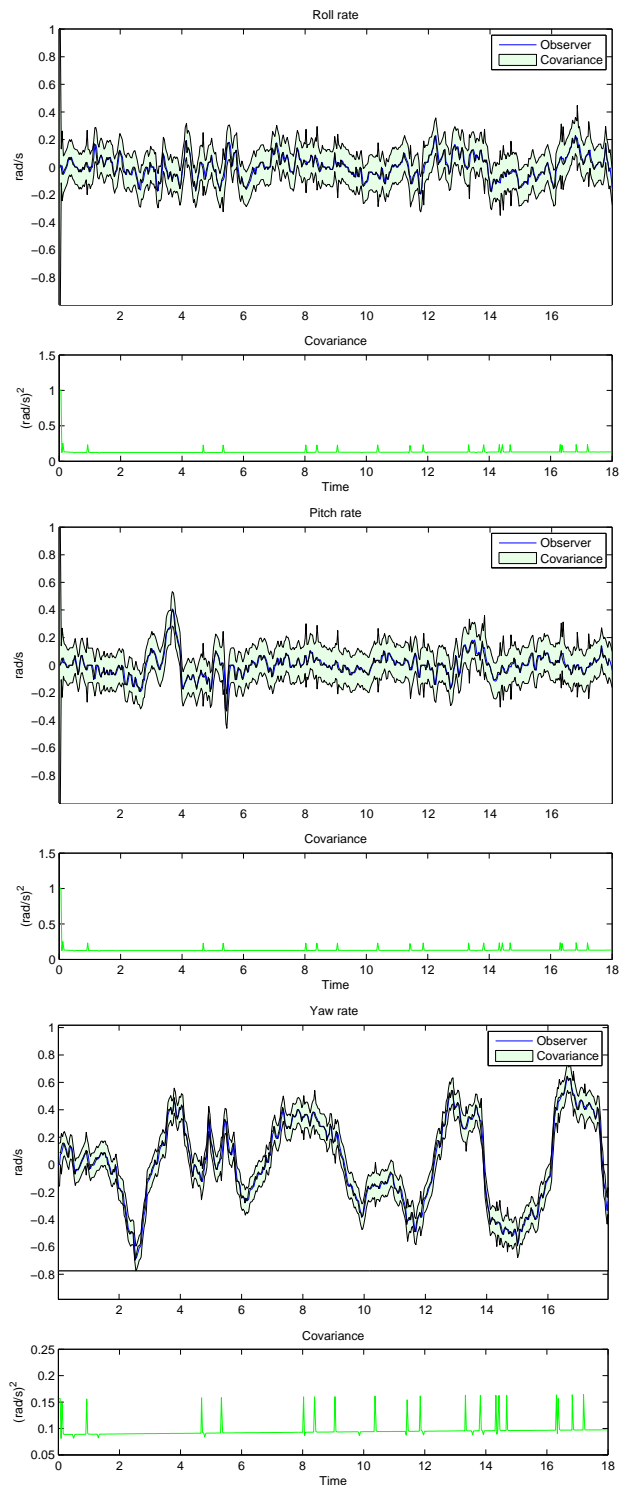
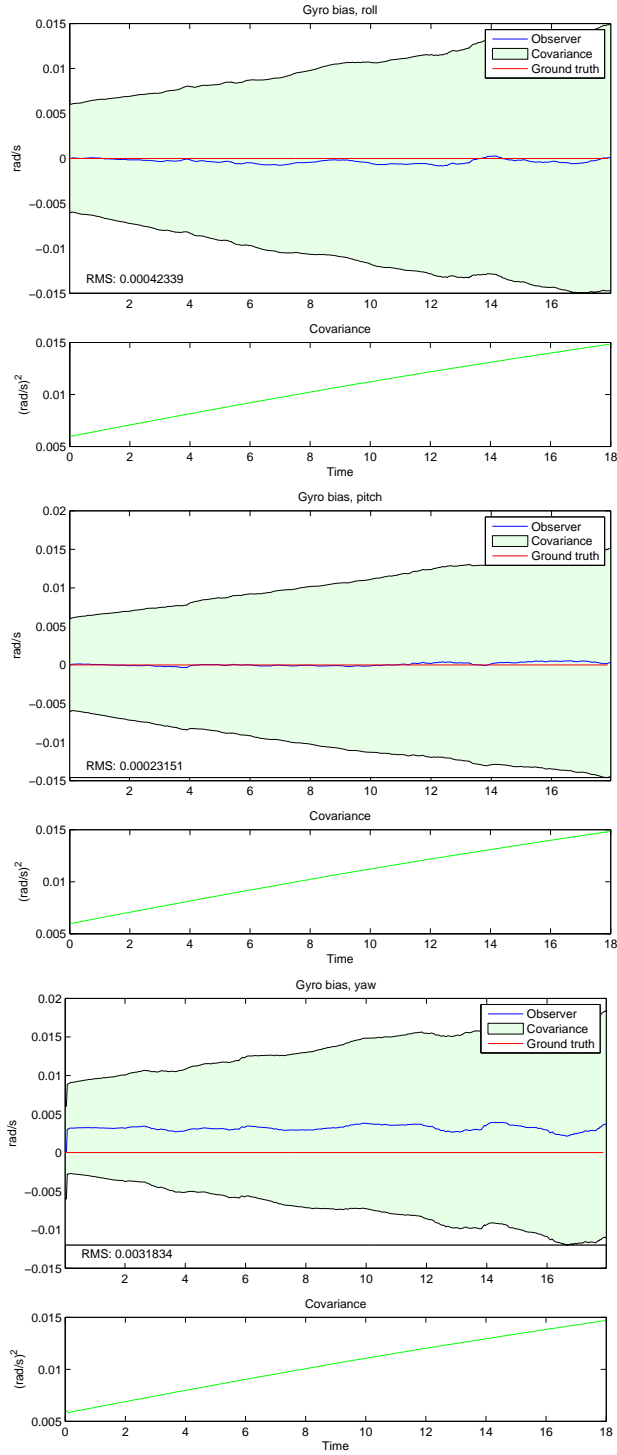**Figure 6.18.** The predicted angular velocities corresponds very well to the gyro measurements.

Figure 6.19. The gyroscopes' drift was removed prior to entering the filter, and does not change during the short recording of data.
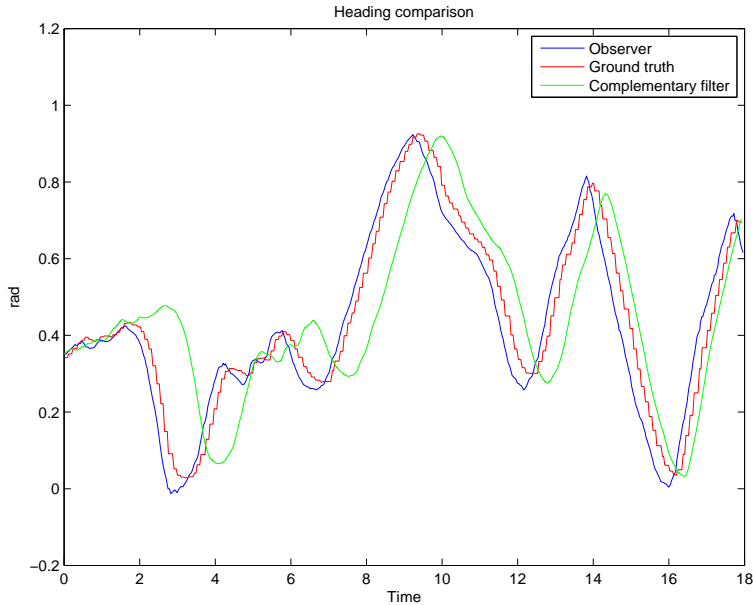
Figure 6.20. A comparison with the tuned complementary filter show that both filters accurately describe the heading, and although the phase of the Kalman filter is better in the beginning, they are of comparable performance towards the end of the dataset.

6.3.4 Wind Force

As the tests were performed inside, the filter was tuned to keep the wind at a zero velocity estimate. Figure 6.21 shows them to be correctly estimated to zero in the collected dataset, although in the case with simulated data with wind, shown in Figure 6.22, results are quite poor, most likely due to lack of filter tuning.

In simulations to verify the wind estimate's behaviour at landing, it is evident that the wind estimate is affected by ground contact, although as the simulation model does not yet fully cover the the non-continuous event of touching the ground, as simulations provide unexpected results, as shown in Figure 6.23.
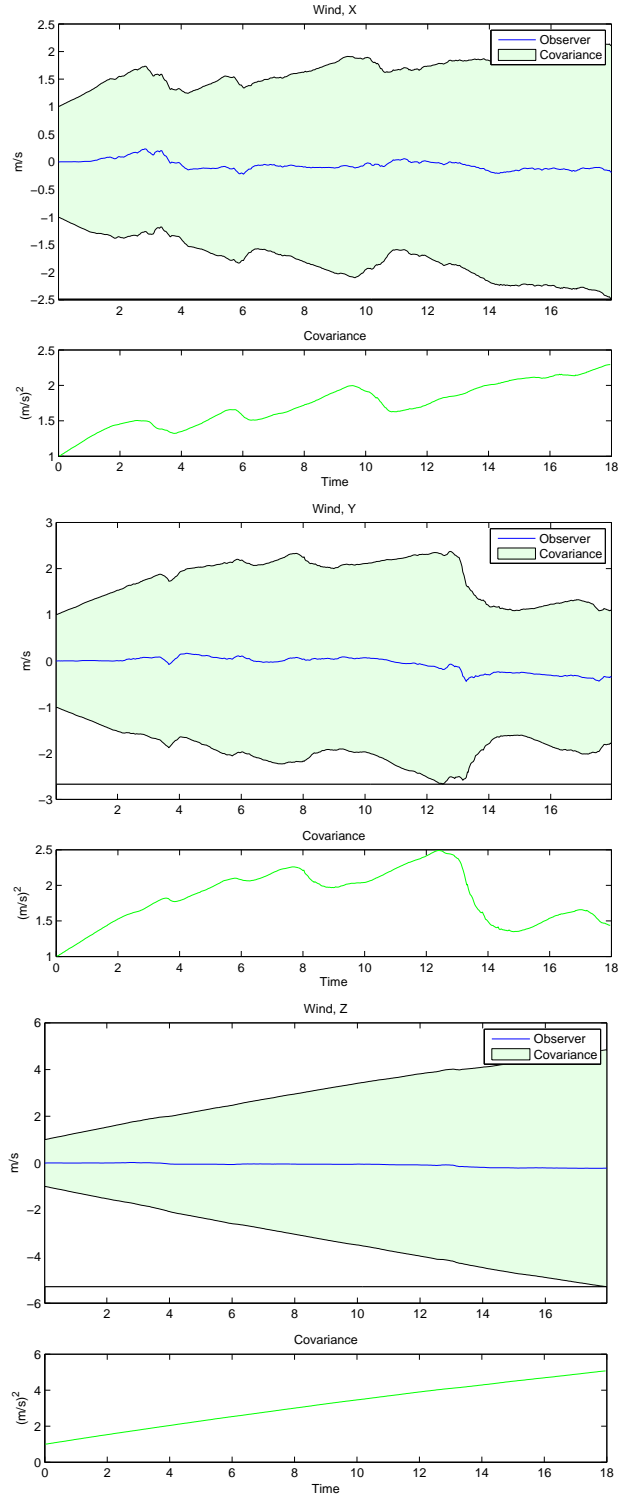
Figure 6.21. Wind estimates from recorded test-data.
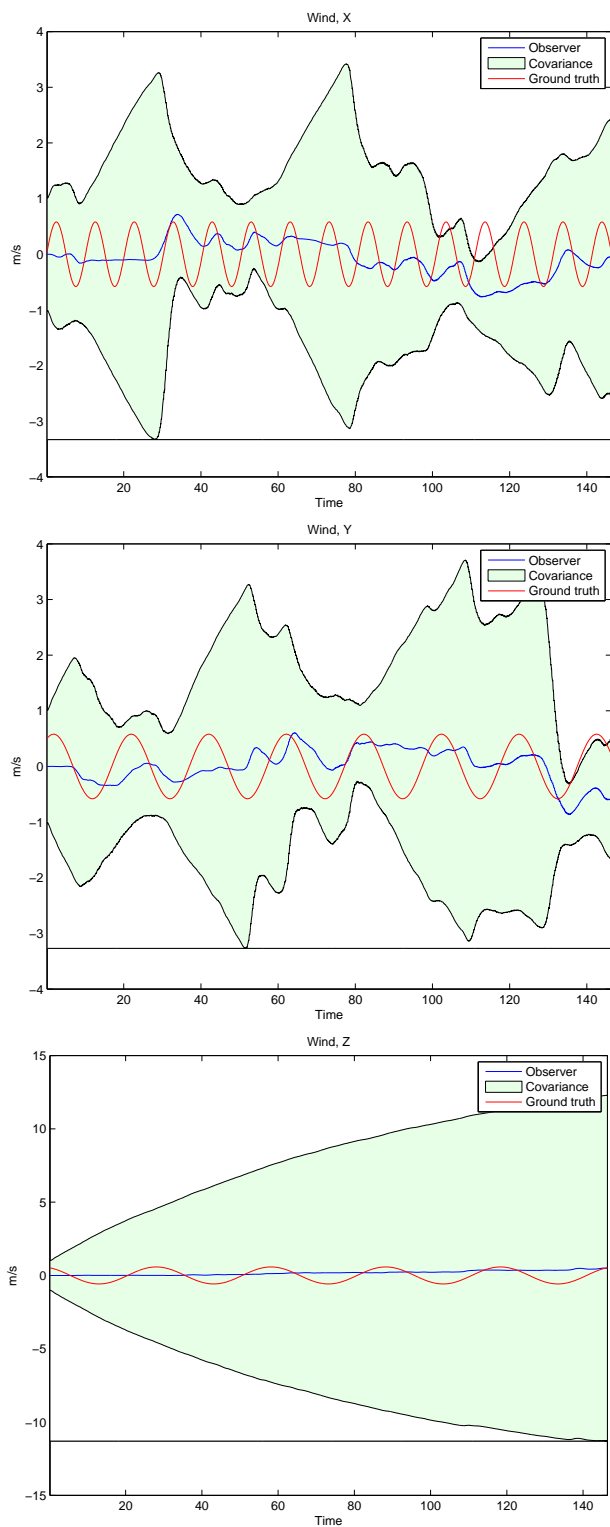
Figure 6.22. Wind from simulated test-flight.
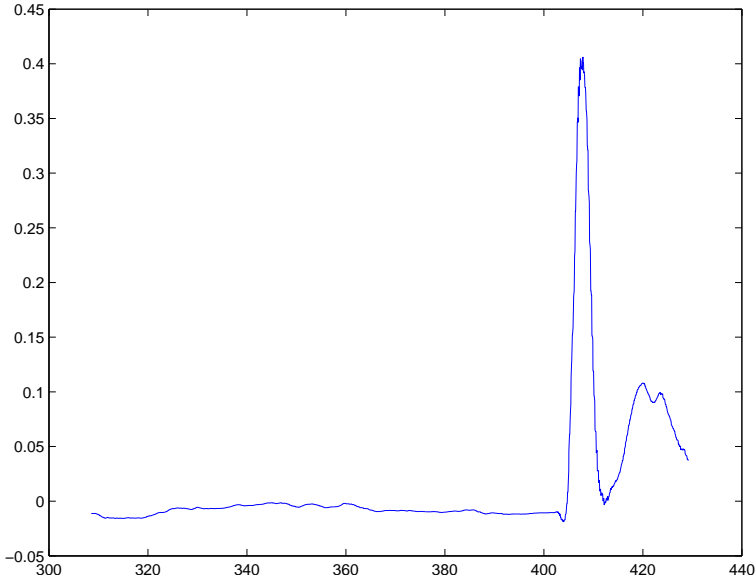
Figure 6.23. Even in simulation, the landing was detectable in the vertical wind estimate, although more work is required to properly model and detect the event.

6.3.5   Propeller Velocity

As the filter evaluation was performed without the use of the controller, the control signal is unavailable. Thus, Eq. (3.39) was used as prediction model for the propeller velocity in the filter validation, effectively leaving the estimation of the propeller velocities to the measurement update. It is evident, in Figure 6.24 that the estimation is active, however it is impossible to validate properly with the available data. Ideally, the velocities of the propellers should be measured in flight. However, that data is currently unavailable in the development system used for evaluation. The estimated velocities are, notably, in a reasonable range, increasing the plausibility for correctness of Eq. (3.40a), which is otherwise hard to verify using the available data.
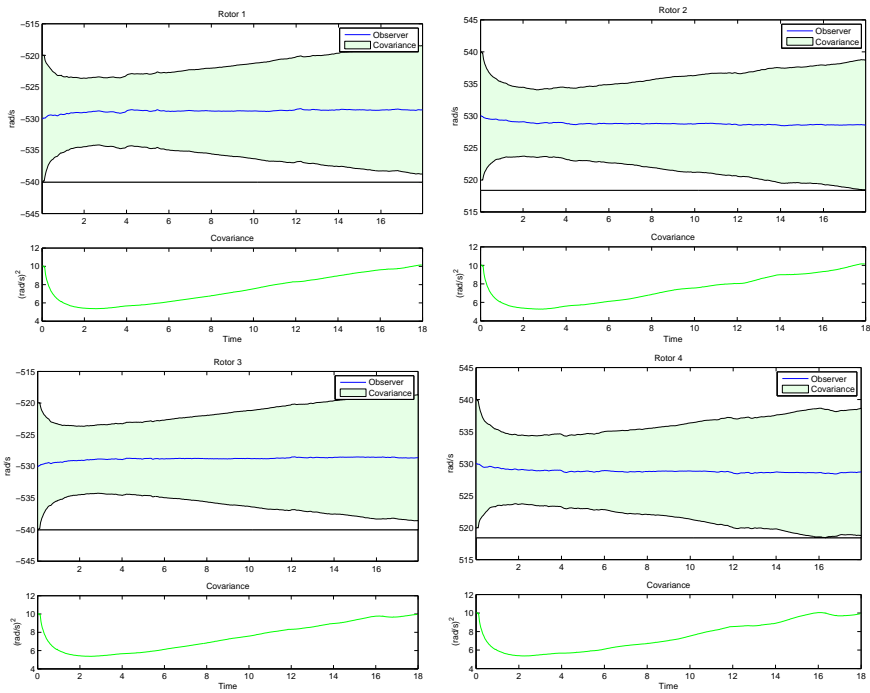
Figure 6.24. Propeller angular rate estimates could not be properly verified with the data available, although do exhibit a reasonable value range given the model parameter settings.

6.4   Control

The control algorithm was tested in simulation on the model presented of Chapter 3 and verified in Section 6.2. The control, although roughly tuned, did in simulation exhibit stable and responsive properties. The reference track included velocity control in all directions, control of yaw rate and finally landing. This corresponds to a wide range of the operations to which a quadrotor may be used, including the landing procedure, which is central to this thesis.

In Figure 6.25, the reference flight is plotted, and as can be seen it starts with a sinusoid velocity. After 150 seconds - simulation time - the landing procedure is initialized and the velocity is reduced until after about 180 seconds, when the velocity is below the threshold to begin descent. Landing is detected by the shortfall of velocity after just over 400 seconds.

Figure 6.25. Control of the simulated flight was stable, although - being untuned - somewhat slow. The actuation of the yaw rate is very weak, which is evident in slow reaction of the yaw rate control.

Chapter 7

Discussion

In this chapter, the results of the thesis are discussed, analyzing the contents of Chapter 6 and providing a basis a for later discussion on future work.
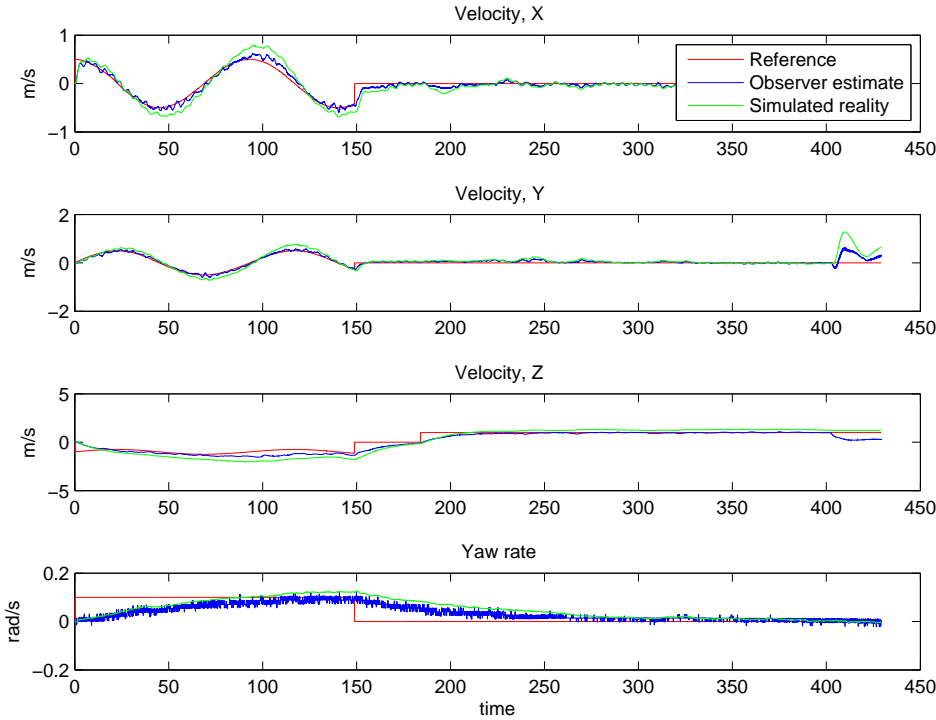
7.1 Camera Localization

Two libraries were evaluated for the camera localization part of this thesis, namely Scenelib and PTAM. Scenelib is the older library of the two, and is also the library to which PTAM was compared in its original paper [22]. The Scenelib library, using the filter approach described in Section 2.1, has the advantage to use metric measures. This was believed to potentially simplify the integration with the observer's measurement update, although in the end, difficulties with camera calibration rendered the Scenelib library subpar.

The PTAM library is designed to be used with a wide-angle camera lens, which unfortunately was impossible with the available resources. While good performance could be achieved in the restricted testing environment, a wide-angle lens is likely to improve the tracking performance when larger scenes are explored.

One can note that tracking was quite stable in recorded test cases with the low capture rate of 15 Hz. A faster rate is nonetheless advisable.

7.2 Filtering

The results for the state estimation algorithms of this thesis were presented in Sections 6.2-6.3 of Chapter 6. In the evaluation, the parameters of the motion model were set to reasonably guessed values but otherwise untuned. Similarly, filter covariances were roughly tuned to reasonable values. Video was recorded for 20 seconds (30 Hz mode) or 40 seconds (15 Hz mode), and the Vicon ground Truth was used for initialization but then only for plot reference.

7.2.1   EKF vs. UKF

Two nonlinear filter algorithms were studied in this thesis. After an initial evaluation period of the Unscented Kalman Filter, one weakness of the algorithm was exposed that rendered the algorithm very difficult to use further.

Since the distribution of sigma points in the UKF is scaled with the covariance, they move farther away as the positioning gets less confident. With a fragile model such as that of a quadrotor, evaluating the model in too large offsets from reasonable ranges will cause effects large enough to destabilize the filtering solution. Excessive covariance in angle estimates may for instance cause the filter to evaluate and compare two cases of the quadrotor being upside down, both irrelevant to the mean case of stable, vertical, flight. Hence, the implementation was changed to use the more stable EKF algorithm.

With further tuning, the UKF may be usable in a later stage of development, but with the inherent instability of the model, the UKF is still problematic in the development stage, as measurements are not always available to reduce covariance.

7.2.2   Performance

The quality of the tracking, despite lack of tuning and previously mentioned problems, is in some cases remarkable given the preconditions, the angle and angular rate estimates in particular. The most important factor for this is the high precision achieved by the camera positioning. By experimenting with filter tuning, it is apparent that noise from the pressure sensor in Figure 6.8 affects the altitude positioning somewhat negatively. However, as the trends in measured and predicted pressure of Figure 6.8 seem to correlate with the ground truth altitude in Figure 6.15, further tuning and modeling may increase the utility of the pressure sensor. Due to its problematic behavior, the pressure sensor was disabled in the filter verification.

With the filter structure in place, the single most rewarding factor to increase the quality of the filtering would be simply to further tune it. Since this can be quite time-consuming, it could not be included in the thesis, but tools for simulation and evaluation are provided in the implementation.

Currently, the integration of the continuous model from Chapter 3 is performed using numerical Euler integration. This is a simple and fast integration method, but lacks in accuracy. Applying a more advanced solution, e.g. a standard Runge-Kutta solver, could be expensive for a complex model, but a study to determine a reasonable trade-off using different integration-techniques could be performed. The cost of integration may very well outweigh the performance gained by applying the advanced motion model, and the precision loss caused by the Euler integration might exceed the gain from the finer details of the said model. A simpler model could remove the need for numerical integration entirely and provide a significant speed-up. The complex model would however still be highly relevant for validation and development purposes.

7.3 Modeling

It is arguable whether the detail of the physical modeling presented in Chapter 3 is tenable, as performance similar or better than presented in Section 6.3 is very likely to have been achieved using a well tuned, far simpler, motion model. These results are not surprising for the untuned state of the model, however. Aside from requiring accurate parameter settings, the full potential of the proposed model is dependent on the availability of control reference signals and more accurate control over the true rotational velocity of the propellers. Feedback of the measured RPM is likely to appear in a later stage of the development of the LinkQuad, and may thus be available in the future.

The model, as presented in this thesis, needs further in-detail verification and tuning. The flapping equations presented in Section 3.3.3 were in fact even disabled during the verification in Chapter 6 due to the difficulty of verification. In its current state, the model adequately describes the accelerations of the system - and with tuning it might become great - but the performance of the model must also be weighed against the computational complexity.

The nature and extensiveness of the model does however open for new applications, including future outdoor applications, due to its handling of wind and other disturbances. The state-space type of model also allows for simple addition of further modeling and added - or removed - states.

The proposed model also bear the advantage that the linearization performed in the Extended Kalman Filter may be cleverly re-used in the control algorithm, removing duplication of computations. In fact, by clever use of the model, only a single model needed to be developed in the thesis, then re-used for state estimation, control, simulation of reality and sensor measurements. The model can, as future work, be simplified, yet still benefit from the added detail where suitable.

One model simplification in particular that is believed to be rewarding, is to replace the physical thrust equations using model identification. Another approach would be to replace the control interface for a more subsidiarized control approach, using existing structure for controlling rotational velocities of the quadrotor. This would however remove the benefits of using the theory of optimal control to control the physical model directly.

7.4 Controller

The advantages and disadvantages of different control techniques and its applications to quadrotor control have been previously discussed and evaluated in [4]. The conclusions are ambiguous, but the state-space approach is believed to excel with continued modeling.

One disadvantage with the state-space approach is its - in comparison to PID control - lack of easily implemented integrating states. This does add demands on a physically correct model, which may not always be available in all flight-conditions.

The SDRE approach used in this thesis is notably rare - if at all existing - in the application of quadrotor control and with its ease of use, it is an interesting topic

for further research and applications. The control implementation is very versatile although the computational burden could be somewhat lifted by comparing other techniques for solving the Riccati equation of Eq. (4.5) and its discrete equivalent. Several numerical approaches exist to solving the Riccati equation, although an iterative approach could prove beneficial since the equation is solved repeatedly and the previous solution would make a good starting guess.

While the control would need to be further verified and tuned before applied in real flight, the tools and structure have been verified in simulation with good results. It should be noted that the control model is fundamentally the same as the observer model and, more importantly, the underlying simulation model. Even though noise was added to both the system and the measurements, the control evaluation suffers from the fact that the control is "perfect" for the simulated model, and not necessarily for real flight.

One clear advantage with the LQ control methodology is its simple tuning, and even though very little of the work was focused on tuning the control, the performance of the simulated control was acceptable. The control handled moderate simulated winds and disturbances without trouble, and followed velocity references adequately. Since the actuating of the yaw rate control is very weak in configurations such as the LinkQuad, slow control is to be expected here, especially without more tuning of the control parameters.

7.5   State-machine Logic

The implementation of the state-machine engine uses a simple, yet programmatically interesting solution. The implemented modes are quite trivial, yet the structure is in place to implement far more advanced modes. In the future, the implementation could be extended to support concurrent state-machines as has been extensively studied in projects closely related to the LinkQuad.

One text-book solution to detecting landing would be to introduce an estimated state of the ground force, which could be thresholded in the detection of landing. One finds, however, that such a force would be indistinguishable from the force acting on the quadrotor by a ever-increasing wind. In simulation, the landing detection failed to detect landing as proposed in Section 5.4.1, although when a landing is simulated, Figure 6.23 exhibits an interesting property where a notable bump occurs at the time of the landing. While this is not the expected behavior - the estimate should be negative and constant - it does show that the wind may be useful as a detector for landing. This discrepancy of results is believed to be due to the simulation's lack of a model for sensor measurements at the event of ground contact, and the observer's lack of control signals in its state estimation.

The results achieved in simulation implies that landing detection using wind estimation might be feasible, although more work is required to achieve the quality of estimation needed to securely detect landing.

7.6   Real-time Performance

Although real-time performance is easily achieved on a modern laptop, simulations indicate that real-time performance could not be achieved on the LinkQuad's gumstix without modifications.

The performance of the proposed algorithms when evaluated on the gumstix is penalized by the lack of a floating point processor (FPU) to carry out the calculations. However, the gumstix has a Digital Signal Processing unit (DSP) which would be possible to use as a FPU to increase execution speed. Attempts were in fact also made to convert libraries and algorithms from double precision calculations to single precision, although these attempts were deferred. This is still, however, believed to be of significance to achieve real-time performance.

Furthermore, the system was evaluated using the reference implementation of BLAS[1], which could possibly be replaced by a tuned implementation such as ATLAS[2]. Model simplifications may also be necessary to achieve real-time performance on the existing hardware, although the nonlinearities introduced by the vehicle's rotations are believed to be relevant. Landing in the trivial case can no doubt be performed using a simple, even linear, model, although generalisations extending the utility of the controller, for instance when using the controller for other flight cases, would require the use of a more advanced model, such as proposed in this thesis.

---

[1] *Basic Linear Algebra Subprograms*, a standard set of mathematical routines
[2] *Automatically Tuned Linear Algebra Software*

Chapter 8

Concluding Remarks

This thesis covers the theory and implementation of the most important aspects of autonomous flight. The implementation is still in need of tuning, both for state estimation accuracy and control, but the foundations for high-level control and estimation using modern and effective algorithms have been laid.

While the implementation has yet to see real flight, it is believed that relatively little work remain before test-flights can be performed, the most important work being tuning of state estimation and control.

8.1 Conclusions

The system proposed in this thesis has been implemented and verified using the framework presented in Appendix A. The verification suggests that with further work, the proposed approach is a feasible solution for landing a quadrotor autonomously. The advanced algorithms used in the thesis allow for simple scaling of the solution, which also makes the method applicable for other types of platforms, or platforms with other sensor configurations.

Positioning using Monocular SLAM has been shown to improve the estimation of the states of the quadrotor significantly. By extending the PTAM library, it has been possible to automate the initialization and utilization while also providing the coupling to a real-world metric system needed for positioning applications.

Even in an untuned state, the positioning provides a state estimate that is adequate for the nonlinear control methodology presented in Chapter 4 to perform well in simulation. The controller extends the well-studied Linear Quadratic controller to a far more general case, while retaining the appreciated simplicity of tuning. The controller successfully lands a quadrotor in a simulated environment, and performs well in symbiosis with the suggested filtering algorithms.

8.2 Further work

While the implementation still needs tuning, it shows promise and one can see many fields which would prove interesting for future in-detail study - filter, model and control tuning included.

8.2.1 Filtering and Control

As a future study, it would be of interest to make a comparison between a fully tuned advanced model, compared to simpler motion models that can be applied in the filtering framework. Also, after the proposed model has been properly verified and tuned, further modeling may be of interest.

As more processing power becomes available one can, in the filtering process, consider using even more state-of the art filtering techniques, such as GPU-implemented Particle filtering. Similarly, simpler control models could be compared with the advanced SDRE-solution proposed in this thesis.

Future work could also investigate the applications of other types of optimal control, such as Model Predictive Control, MPC.

8.2.2 Monocular SLAM

One of the main results of this thesis is the relating of the PTAM coordinate system to the world coordinate system. This result could be relevant for any future work including real-world positioning using video feedback.

There is also a potential to improve the PTAM library. The PTAM library was specifically developed for hand-held cameras without any additional sensors, yet the algorithm would benefit from such. The state estimate and motion model of the observer could be utilized to improve the quality and stability of the camera tracking.

The PTAM library also exhibit performance issues when the internal map of features grows. This could be improved by using informed search, for instance using R-trees, to index features according to their position and limit the search for relevant features to re-project into the image.

Bibliography

[1] Maziar Arjomandi. Classification of Unmanned Aerial Vechicles. Technical report, Mechanical Engineering, 2006.

[2] Peter Benner. Accelerating Newton's Method for Discrete-Time Algebraic Riccati Equations. In *In Proc. MTNS 98*, pages 569–572, 1998.

[3] Michael Blösch, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Vision based MAV navigation in unknown and unstructured environments. In *ICRA*, pages 21–28, 2010.

[4] S Bouabdallah, A Noth, and R Siegwart. PID vs LQ Control Techniques Applied to an Indoor Micro Quadrotor. In *Proc. of The IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[5] Samir Bouabdallah and Roland Siegwart. Full control of a quadrotor. In *IROS'07*, pages 153–158, 2007.

[6] Roland Brockers, Patrick Bouffard, Jeremy Ma, Larry Matthies, and Claire Tomlin. Autonomous landing and ingress of micro-air-vehicles in urban environments based on monocular vision. In Thomas George, M. Saif Islam, and Achyut K. Dutta, editors, *Micro- and Nanotechnology Sensors, Systems, and Applications III*, volume 8031, page 803111. SPIE, 2011.

[7] Tayfun Çimen. State-Dependent Riccati Equation (SDRE) Control: A Survey. In *Proceedings of the 17th IFAC World Congress*, pages 3761–3775, 2008.

[8] National Geophysical Data Center. The World Magnetic Model. `http://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml`, March 2012.

[9] Andrew J. Davison. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *ICCV*, pages 1403–1410. IEEE Computer Society, 2003.

[10] Patrick Doherty and Piotr Rudol. A UAV Search and Rescue Scenario with Human Body Detection and Geolocalization, 2007.

[11] Ethan Eade and Tom Drummond. Scalable Monocular SLAM. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and*

Pattern Recognition - Volume 1, CVPR '06, pages 469–476, Washington, DC, USA, 2006. IEEE Computer Society.

[12] Evrin Bilge Erdem. Analysis and Real-time Implementation of State-Dependent Riccati Equation Controlled Systems, 2001.

[13] T. Glad and L. Ljung. *Reglerteori: flervariabla och olinjära metoder.* Studentlitteratur, 2003.

[14] F. Gustafsson. *Statistical Sensor Fusion.* Utbildningshuset/Studentlitteratur, 2010.

[15] Masayuki Hayashi et al. A Study of Camera Tracking Evaluation on TrakMark Data-Set. Technical report, University of Tsukuba, 2011.

[16] M.Y.I. Idris, H. Arof, E.M. Tamil, N.M. Noor, and Z. Razak. Review of Feature Detection Techniques for Simultaneous Localization and Mapping and System on Chip Approach. *Information Technology Journal*, 8:250–262, 2009.

[17] Simon J. Julier and Idak Industries. The scaled unscented transformation. In *in Proc. IEEE Amer. Control Conf*, pages 4555–4559, 2002.

[18] Simon J. Julier and Jeffrey K. Uhlmann. Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, pages 401–422, 2004.

[19] Simon J. Julier, Jeffrey K. Uhlmann, and Hugh F. Durrant-Whyte. A new approach for filtering nonlinear systems. In *1995 American Control Conference, 14th, Seattle, WA; UNITED STATES; 21-23 June 1995*, pages 1628–1632, 1995.

[20] Kalman, Rudolph, and Emil. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[21] Niklas Karlsson, Enrico Di Bernardo, Jim Ostrowski, Luis Goncalves, Paolo Pirjanian, and Mario E. Munich. The vSLAM algorithm for robust localization and mapping. In *In Proc. of Int. Conf. on Robotics and Automation (ICRA*, pages 24–29, 2005.

[22] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[23] Georg Klein and David Murray. Parallel Tracking and Mapping on a Camera Phone. In *Proc. Eigth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'09)*, Orlando, October 2009.

[24] J.B. Kuipers. *Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality.* Princeton paperbacks. Princeton University Press, 2002.

[25] J.G. Leishman. *Principles of helicopter aerodynamics*. Cambridge aerospace series. Cambridge University Press, 2002.

[26] Manolis I.A. Lourakis, editor. *Bundle adjustment gone public*, 10 2011.

[27] M.I. A. Lourakis and A.A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *ACM Trans. Math. Software*, 36(1):1–30, 2009.

[28] D. Mellinger, M. Shomin, and V. Kumar. Control of Quadrotors for Robust Perching and Landing. In *Proceedings of the International Powered Lift Conference*, Oct 2010.

[29] Torsten Merz, Piotr Rudol, and Mariusz Wzorek. Control System Framework for Autonomous Robots Based on Extended State Machines. In *ICAS 2006 - International Conference on Autonomic and Autonomous Systems,2006*, 2006.

[30] Richard Newcombe, Steven Lovegrove, and Andrew Davison. DTAM: Dense Tracking and Mapping in Real-Time. In *13th International Conference on Computer Vision (ICCV2011)*, November 2011.

[31] Thanh Nguyen, Christian Sandor, and Jun Park. PTAMM-Plus: Refactoring and Extending PTAMM. *Camera*, pages 84–88, 2010.

[32] Carl Nordling and Jonny Österman. *Physics Handbook*. Studentlitteratur, 2006.

[33] Mattias Nyberg and Erik Frisk. *Model Based Diagnosis of Technical Processes*. LiU-tryck, 2011.

[34] United States. Dept. of Defense. Office of the Secretary of Defense. *U.S. Army Roadmap for unmanned aircraft systems, 2010-2035*. U. S. Army UAS Center of Excellence, 2010.

[35] Paul Pounds, Robert Mahony, and Peter Corke. Modelling and Control of a Quad-Rotor Robot.

[36] R.W. Prouty. *Helicopter performance, stability, and control*. Krieger Pub., 1995.

[37] A. Rantzer and M. Johansson. Piecewise Linear Quadratic Optimal Control, 1999.

[38] Piotr Rudol, Mariusz Wzorek, and Patrick Doherty. Vision-based Pose Estimation for Autonomous Indoor Navigation of Micro-scale Unmanned Aircraft Systems. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA)*, number 2010 in Proceedings - IEEE International Conference on Robotics and Automation, pages 1913–1920. IEEE conference proceedings, 2010.

[39] V. Sima and P. Benner. A SLICOT Implementation of a Modified Newton's Method for Algebraic Riccati Equations. *Mediterranean Conference on Control and Automation*, 0:1–6, 2006.

[40] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Real-time monocular SLAM: Why filter? In *ICRA*, pages 2657–2664, 2010.

[41] David Törnqvist. *Estimation and Detection with Applications to Navigation*. PhD thesis, Linköping University, 2008.

[42] K. Valavanis. *Advances in unmanned aerial vehicles: state of the art and the road to autonomy*. International series on intelligent systems, control, and automation. Springer, 2007.

[43] Rudolph van der Merwe, Arnaud Doucet, Nando de Freitas, and Eric A. Wan. The Unscented Particle Filter. In *NIPS'00*, pages 584–590, 2000.

[44] S Weiss, D Scaramuzza, and R Siegwart. Monocular-SLAM based navigation for autonomous micro helicopters in GPS-denied environments. *Journal of Field Robotics*, 28(6):854–874, 2011.

[45] K C Wong and C Bil. UAVs Over Australia - Market And Capabilities. *Flight International*, pages 1–16, 2006.

[46] Mariusz Wzorek. Selected Aspects of Navigation and Path Planning in Unmanned Aircraft Systems, 2011.

Appendix A

CRAP

For the implementation of the theory presented in this report, a framework was developed for connecting the different separable modules and provide a core library of useful functions. The result is called *C++ Robot Automation Platform*, or *CRAP* for short. To emphasize the central idea of modularity, one can note that the actual compiled core - headers not included - are less that two-hundred lines of code.
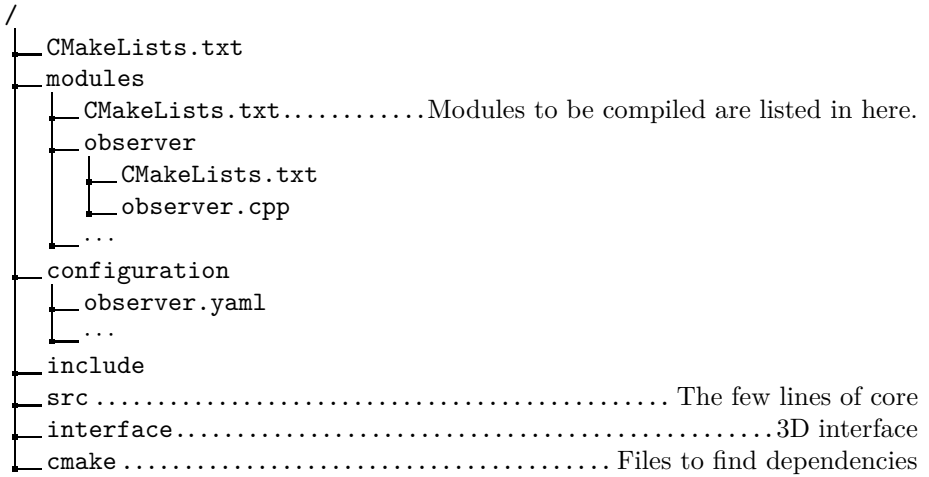
The main ideas of *CRAP* are borrowed from the *Robot Operating System*, ROS[1], while implementing these in a more efficient and uniform manner. By constraining the intermodular messaging to C++[2] and compartmentalizing the modules in separate threads as opposed to processes, *CRAP* significantly reduces the overhead associated with the flexibility of such modular software.

This document contains a brief technical description of the inner workings of the framework, focused on the implementation for quadrotor control described in this thesis. Further documentation is available in the code, which can be found at `https://github.com/jonatanolofsson/crap`.

The relevant directory structure is as follows, exemplified for the observer module:

---

[1]`http://ros.org/`
[2]ROS allows messaging between Python and C++ modules

```
/
├── CMakeLists.txt
├── modules
│   ├── CMakeLists.txt............Modules to be compiled are listed in here.
│   ├── observer
│   │   ├── CMakeLists.txt
│   │   └── observer.cpp
│   └── ...
├── configuration
│   ├── observer.yaml
│   └── ...
├── include
├── src ........................................... The few lines of core
├── interface...............................................3D interface
└── cmake ..................................... Files to find dependencies
```

A.1 Structure

Modules, as shown in Figure A.1, connect to each other using *messages* sent in *message topics*. Topics are identified by a string name, and can be *listened to* or *published to*.

Each module is initialized in its own processing thread, but may chose to end this initial thread and react only to incoming messages, which are received in separate messaging threads, which is instantiated when a module starts to *listen* to a topic. Further details of this procedure is given in Section A.2

CRAP makes use of the *yaml-cpp* C++ library, which allows configuration to be simply parsed run-time. The core configuration file is given as the first input argument to the *CRAP* executable and contains a list of the modules that should be loaded, each with a unique name, executable shared object and an optional configuration file, as exemplified in Listing A.1.

Each module is compiled individually to a *shared object*, which can be dynamically linked to the *CRAP* platform at run-time, in accordance with the selected core configuration. The modules are all located in the modules directory, and all compilation details needed for run-time linking are defined in the CMake configuration of that directory.

```
 1   −−−
 2   module_root: modules/
 3   config_root: /home/shared/projects/crap/configuration/
 4   modules:
 5   −   name: sender
 6       file: comm_sender.so
 7
 8   −   name: baselink
 9       file: baselink.so
10       configuration:
11           receive_flight_commands: false
12           send_flight_data: true
13           port: /dev/pts/5
14           reference_timeout: 10.0
15
16   −   name: observer
17       file: observer.so
18       configuration: observer.yaml
19
20   −   name: reality
21       file: reality.so
22       configuration: reality_reader.yaml
23
24   −   name: sensor_reader
25       file: sensor_reader.so
26       configuration: sensor_reader.yaml
27
28   −   name: camera_reader
29       file: camera_reader.so
30       configuration: camera_reader.yaml
```

Listing A.1. Core configuration example, listing what modules should be loaded on execution.
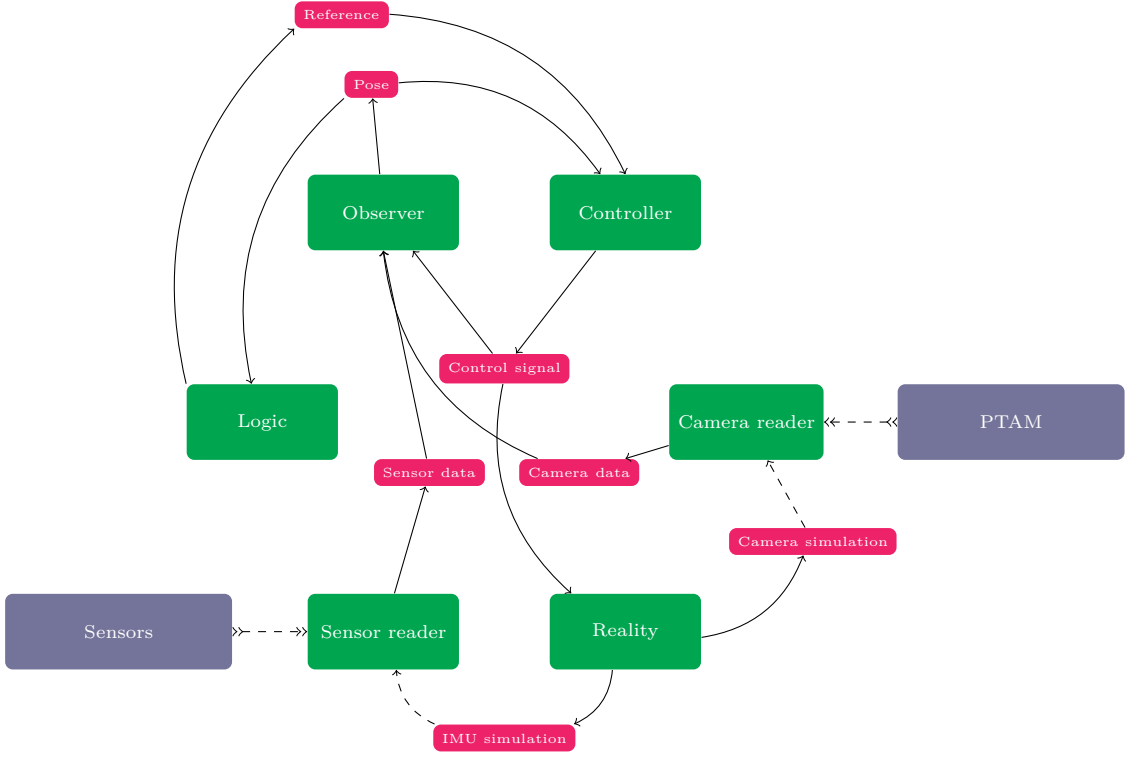
Figure A.1. CRAP schematic. The Figure describes the relationship of modules (green) and their internal communication through messages (red). External modules, connected through a serial interface, are shown in blue. Dashed lines are optionally enabled or disabled in different run-configurations of the thesis implementation.

A.2 Communication

As presented in Section A.1, the modules interact using *messages* in *topics*. When a message is sent, it is pushed to a messenger thread which will invoke the callback specified by each listening module.

Using the existing message definitions for serial communication on the LinkQuad, the framework also provides an implementation for serial communication which reuses this interface, allowing callbacks to be used interchangeably for both serial and internal communication.

The following subsections provide code samples for both use cases.

A.2.1 Internal

The example in Listing A.2 displays the simple interface for internal communication. As long as both sender and receiver agree on the type, any data can be transferred, as the data is passed directly through function pointers without need

for serialization. Each topic can have multiple listeners and multiple senders, but only one data type can be sent on each topic. This restriction is not programmatically enforced, but must be asserted in the development process.

```
1  void switch_mode(const std::string& mode) {...}
2  ...
3  comm::listen("/logic/mode", switch_mode);
4  ...
5  comm::send("/logic/mode", std::string("hover"));
```

Listing A.2. Excerpts from the mode-switching code in the state-machine, demonstrating the use of internal messaging.

A.2.2 Serial Communication

The serial communication is slightly more restricted than the internal communication, to comply with the existing definition of the LinkQuad serial communication. As exemplified in Listings A.3-A.5, the interface is however as far as possible the same as for the internal communication. It should be noted that the `listen` method used in Listing A.5 automatically requests the data accepted by the callback by sending a data request of the specified serial port. This request is disabled using the alternative, but analog, method `passive_listen`.

The serial communication has been branched into a separate library to allow use outside of the *CRAP* framework.

```
1  using namespace LinkQuad::comm::serial;
2  using namespace LinkQuad::comm::serial::data;
3  using namespace LinkQuad::comm::serial::data::SUser;
4  typedef LinkQuad::comm::serial::data::serial_data<
5      params_32f_0,
6      params_32f_1,...> serial_data;
```

Listing A.3. Message definition for the serial communication. Both communicating parts must agree on what data is sent.

```
1   using namespace LinkQuad::comm::serial::data;
2   using namespace LinkQuad::comm::serial::data::SSMCU;
3   void camera_receive(const serial_data& d) {
4           measurement.z <<
5               (scalar) d.params_32f_0,
6               (scalar) d.params_32f_1,
7   ...
8   }
9   ...
10  LinkQuad::comm::serial::listen<SSMCU::Part>("/dev/pts/1", camera_receive);
```

Listing A.4. Excerpts from code receiving data over serial communication. The `listen` automatically requests the correct data using LinkQuad data-request messages.

```
1  msg.params_32f_0 = q.x();
2  msg.params_32f_1 = q.y();
3  ...
4  LinkQuad::comm::serial::send(serial_port, msg);
```

Listing A.5. Excerpts from code for sending data over the serial interface. The data is then sent in a separate thread, leaving the caller to continue.

A.3 Modules

This section describes the most important modules, and their implementations, individually.

A.3.1 Observer

The observer is implemented according to the theory presented in Chapter 3. As implemented, the observer responds to three different events:

- The filter's time update is performed at a fixed rate.

- The IMU measurement update is run as soon as IMU data is received.

- The camera measurement update is run as soon as camera data is received.

The two latter events are triggered by incoming messages, and are run in separate threads, as depicted in the schematic in Figure A.2.
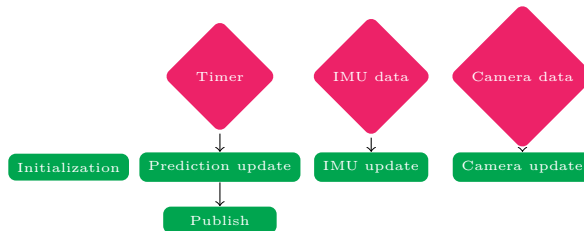

Figure A.2. The observer reacts to three events; A timer executes the prediction update, and incoming messages invoke the measurement updates. The state is published to other modules at a fixed rate immediately after the prediction update.

A.3.2 Controller

The controller implements the nonlinear control described in Chapter 4. The control output is updated as soon as a new pose estimate is received from the observer. The reference signal is updated asynchronously as such a message arrives, but does not cause an update of the output signal. The observer is thus reactive only, and the original thread will terminate as soon as the message topic callbacks have been registered.
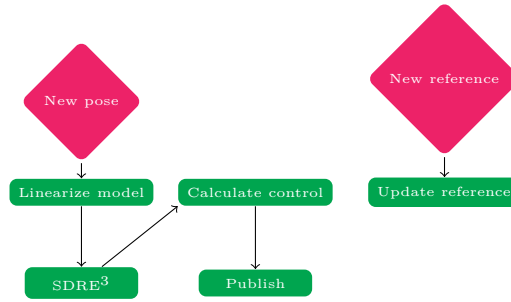
Figure A.3. The controller output is changed only when the a new pose estimate arrives from the observer.

A.3.3 Logic

The logic module uses a single-threaded state-machine to implement the state-machine modes proposed in Chapter 5. Just as the modules, each mode is compiled independently to a shared object which is then loaded and linked when the mode is requested. The relevant directory stucture is as follows:

```
/
├── configuration
│   └── modes
│       └── hover.yaml
└── modules
    └── logic
        └── modes
            ├── CMakeLists.txt ....... Modes to be compiled are listed in here.
            └── landing
                ├── CMakeLists.txt
                └── hover.cpp
```

The file `hover.cpp` is what contains the chain that is executed in the state-machine mode. Each mode must contain a C-method with the same name as the mode. This is the entry point to the mode. The return-value of this method, and all other parts of the chain, is a void pointer to the next function to be executed. The active function is invoked at a fixed rate until a NULL value is returned. Listing A.6 contains excerpts from the `hover` mode, which exemplifies the function chaining as well as demonstrating practical use of the *CRAP* framework. The configure method in said listing will - when the mode is first requested and thus loaded - receive the contents of the configuration file in the `modes` directory.

---

[3]Solve the Damn Riccati Equation

```
1   extern "C" {
2       YAML::Node config;
3       void configure(YAML::Node& c) {config = c;}
4
5       typedef state_vector(*state_fn)();
6       state_fn get_state = comm::bind<state_fn>("observer", "get_state");
7
8       void* hover_control() {
9           ...
10          comm::send("/reference", ref);
11          return (void*) hover_control;
12      }
13
14      void* hover() {
15          x = get_state();
16          position = x.segment<3>(state::position);
17          return (void*) hover_control;
18      }
19  }
```

Listing A.6. Code sample demonstrating the central functionality of each state-machine mode.

A.3.4 Camera Reader, Sensor Reader and Baselink

The camera reader provides the interface to the serially transferred measurements from the PTAM library, running on the secondary computer. Likewise, the sensor reader module interfaces the sensor MCU on the LinkQuad to retrieve IMU measurements, which are forwarded to the observer.

The baselink module interfaces the graphical front-end, presented in Section A.4, through a serial interface, allowing the front-end to e.g. run on a connected laptop when *CRAP* is run on a host without support for graphics.

A.4 Interface

The baselink module that connects to the graphical interface provides relevant parts of the observer's current state estimate. This is used to visualize the simulation or flight in real-time, as depicted in Figure A.4.

The baselink module also accepts commands over the serial interface, which can be sent to *CRAP* and forwarded by the `freeflight` state-machine mode to the controller as control reference. The interface implements 3D control through the open-source spacenav[4] drivers for 3DConnexion's 3D-mice.
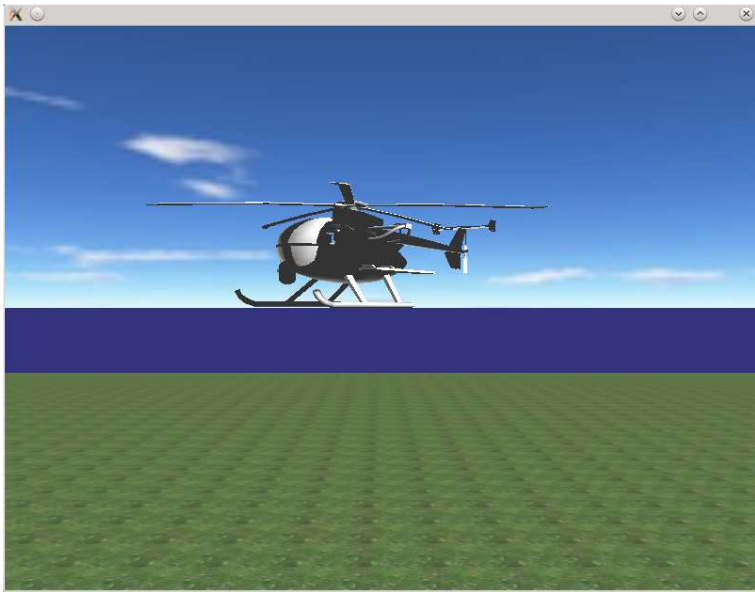
---

[4]http://spacenav.sourceforge.net/

Figure A.4. The *CRAP* framework features a simple graphical viewer to visualize the observer's pose estimates.