

Institutionen för datavetenskap
Department of Computer and Information Science

Master's Thesis

State Estimation and Control of a Quadrotor using Monocular SLAM

Jonatan Olofsson

Reg Nr: LiTH-ISY-EX--YY/XXXX--SE
Linköping YYYY



Linköping University
INSTITUTE OF TECHNOLOGY

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

Master's Thesis


**State Estimation and Control of a
Quadrotor using Monocular SLAM**

Jonatan Olofsson

Reg Nr: LiTH-isy-ex--YY/XXXX--SE
Linköping YYYY

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

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

	Avdelning, Institution Division, Department Department of Computer and Information Science Department of Computer and Information Science Linköpings universitet SE-581 83 Linköping, Sweden	Datum Date YYYY-09-19
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN _____ ISRN LiTH-ISY-EX--YY/XXXX--SE Serietitel och serienummer ISSN Title of series, numbering _____
URL för elektronisk version http://www.ida.liu.se/divisions/aics/ http://www.ida.liu.se/divisions/aics/		
Titel Title State Estimation and Control of a Quadrotor using Monocular SLAM Författare Jonatan Olofsson Author		
Sammanfattning Abstract <p>This thesis treats the problem of modelling, estimating and autonomously controlling an Unmanned Aerial Vehicle. The <i>LinkQuad</i> quadrotor developed at Linköping University is used as a development platform, although the results are general for quadrotors and easily extended to other UAV platforms. In the thesis, the modules and core of a control system is developed to fuse state-estimation algorithms with advanced non-linear control.</p> <p>The problem of state-estimation and control of Small UAV's (SUAV's) is often approached using off-board cameras and sensors to get a precise pose estimation. This however restricts the utility of the controlled UAV's to a very limited space where sophisticated and expensive gear is available and properly configured. By using only on-board sensors, not only is the utility of the UAV greatly increased, but costs are also cut.</p> <p>To assist the positioning, the quadrotor is equipped with a camera utilized by a library for monocular video-based SLAM, Simultaneous Localization And Mapping. In the thesis, an algorithm is suggested for autonomous initialization and for transforming the localization in videoframe-coordinates to world coordinates which can be used in the state-estimation.</p>		
Nyckelord Keywords slam, ptam, control, uav, linkquad, non-linear control, sdre		

Abstract

This thesis treats the problem of modelling, estimating and autonomously controlling an Unmanned Aerial Vehicle. The *LinkQuad* quadrotor developed at Linköping University is used as a development platform, although the results are general for quadrotors and easily extended to other UAV platforms. In the thesis, the modules and core of a control system is developed to fuse state-estimation algorithms with advanced non-linear control.

The problem of state-estimation and control of Small UAV's (SUAV's) is often approached using off-board cameras and sensors to get a precise pose estimation. This however restricts the utility of the controlled UAV's to a very limited space where sophisticated and expensive gear is available and properly configured. By using only on-board sensors, not only is the utility of the UAV greatly increased, but costs are also cut.

To assist the positioning, the quadrotor is equipped with a camera utilized by a library for monocular video-based SLAM, Simultaneous Localization And Mapping. In the thesis, an algorithm is suggested for autonomous initialization and for transforming the localization in video-frame-coordinates to world coordinates which can be used in the state-estimation.

Sammanfattning

Denna rapport avhandlar modellering, estimering och autonom kontroll över en obemannad flygfarkost, UAV. Quadrotorn *LinkQuad* som utvecklas vid Linköpings Universitet har använts som utvecklingsplattform, men resultaten är generella för liknande plattformar och kan lätt utökas till andra typer av obemannade flygfarkoster. I arbetet utvecklas moduler och kärnan till ett reglersystem som binder samman algoritmer för tillståndsestimering med avancerad olinjär reglerteknik.

Problemet med tillståndsestimering och reglering av små obemannade flygfarkoster (SUAV) angrips ofta med hjälp av externa kameror och sensorer för en mycket precis estimering och reglering. Genom att enbart använda sensorer tillgängliga på farkosten kan man utöka användbarheten, samtidigt som man håller kostnader nere.

Vid positioneringen av quadrotorn används en kamera som används av positioneringsbiblioteket PTAM, som enbart behöver en kamera för att utföra SLAM, *Simultaneous Localization And Mapping*. I rapporten föreslås en algoritm för att initiera biblioteket autonomt och omvandla de erhållna video-koordinaterna till användbara världs-koordinater som kan användas i tillståndsestimeringen.

Acknowledgments

Contents

1	Introduction	1
1.1	Unmanned Aerial Vehicles	1
1.2	The Platform	2
1.3	Previous work	3
1.3.1	Autonomous Landing and Control	3
1.3.2	Visual SLAM	4
1.4	Objectives and Limitations	4
1.5	Contributions	5
1.6	Thesis Outline	5
2	State Estimation	7
2.1	The Filtering Problem	7
2.2	The Unscented Kalman Filter	10
2.3	Motion Model	12
2.3.1	Coordinate Frames	12
2.3.2	Kinematics	15
2.3.3	Dynamics	15
2.4	Sensor Models	20
2.4.1	Accelerometers	20
2.4.2	Gyroscopes	20
2.4.3	Magnetometers	20
2.4.4	Pressure Sensor	21
2.4.5	Camera	21
3	Non-linear Control	25
3.1	The Linear Quadratic Controller	25
3.2	State-Dependent Riccati Equations and LQ Gain Scheduling	26
3.3	Control Model	27
4	Monocular SLAM	29
4.1	Filtering Based Solutions	29
4.2	Keyframe-based SLAM	30
4.3	The PTAM Library	30
4.3.1	Operation	31
4.3.2	Modifications to the Library	32

4.3.3	Practical Use	33
5	Finite State-Machines	35
5.1	Implemented Modes	35
5.1.1	Hovering	35
5.1.2	PTAM initialization	36
5.1.3	Free Flight	36
5.1.4	Landing	36
6	Results	39
6.1	Experiment Setup	39
6.2	Modelling	40
6.2.1	Accelerometers	40
6.2.2	Gyroscopes	42
6.2.3	Pressure Sensor	43
6.2.4	Camera	45
6.3	Filtering	47
6.3.1	Positioning	47
6.3.2	Velocities	48
6.3.3	Orientation, Rotational Velocity and Gyroscope Bias	49
6.3.4	Wind force	52
6.3.5	Propeller Velocity	54
6.4	Control	55
7	Discussion	57
7.1	Modelling	57
7.2	Filtering	58
7.2.1	EKF vs. UKF	58
7.2.2	Performance	58
7.3	Camera Localization	59
7.4	Controller	59
7.5	State-machine Logic	60
7.6	Real-time Performance	60
8	Concluding Remarks	61
8.1	Further work	61
8.1.1	Filtering and Control	61
8.1.2	Mission Planning	62
8.1.3	Monocular SLAM	62
8.2	Conclusions	62
A	CRAP	67
A.1	Structure	68
A.2	Communication	70
A.2.1	Internal	70
A.2.2	Serial Communication	71
A.3	Modules	72

A.3.1	Observer	72
A.3.2	Controller	72
A.3.3	Logic	73
A.3.4	Camera Reader, Sensor Reader and Baselink	74
A.4	Interface	74
B	GNU General Public License	76

Chapter 1

Introduction

The goal for this thesis was to develop and implement a high-level control system for the quadrotor developed by the AIICS¹ 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 one Master's thesis, the control system developed in this thesis shows good promise.

Although the LinkQuad quadrotor was used as development platform for this thesis, the results are general and portable to other quadrotors and, by extension, other UAV's. The LinkQuad is considered a SUAV, a Small Unmanned Aerial Vehicle - A UAV small enough to be carried by man [44].

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 it turns out that the LinkQuad design is in fact very suitable for the camera-based pose estimation due to its dual onboard computers. The video-based estimation can effectively be detached from the core control and state estimation and modularized as an independent virtual sensor, which is exploited in the thesis.

1.1 Unmanned Aerial Vehicles

Unmanned Aerial Vehicles (UAV's) have been imagined and constructed for millennia, starting in ancient Greece and China[41]. The modern concept of UAV's 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 UAV's are intended to replace human presence in missions that are "Dull, Dirty and Dangerous".

While the military continues to lead the development in the field [33], recent years have seen a great increase in domestic and civilian applications [45]. These

¹<http://www.ida.liu.se/divisions/aiics/>

applications range from pesticide dispersing and crop monitoring to traffic control, border watch and rescue scenarios [9].

The type of UAV that is used in the implementation of this thesis falls under the category of Small Unmanned Aerial Vehicles (SUAV's) [41]. SUAV's 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 Vicon² and Qualisys³ 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 position the LinkQuad relative to its surroundings.

1.2 The Platform

The LinkQuad is a modular quadrotor developed at Linköping University. The core configuration is equipped with standard MEMS sensors (accelerometers, gyroscopes and a magnetometer), but for our purposes, a monocular camera has also been mounted, which feeds data into a microcomputer specifically devoted to the processing of the camera feed. 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 ROS⁴. It is, in contrast to ROS, primarily designed to run on the kind relatively low-end Linux systems that fits the payload and power demands of a SUAV. The framework is further described in Appendix A.

Using the framework, the functionality of the implementation is distributed in separate modules:

- **Observer** Sensor fusing state estimation. Chapter 2.
- **Control** Affine Quadratic Control. Chapter 3.
- **Logic** State-machine for scheduling controller parameters and reference trajectory. Chapter 5.

The platform is also equipped with two microcontrollers, responsible for sensor sampling, motor control, flight logging etc, as depicted in Figure 1.1.

²<http://www.vicon.com/>

³<http://www.qualisys.com/>

⁴<http://www.ros.org/>

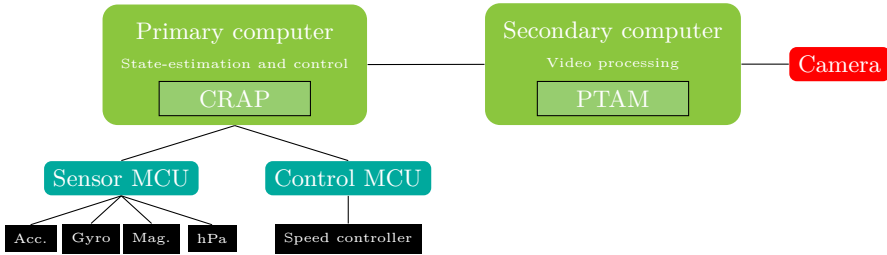


Figure 1.1. LinkQuad schematic.

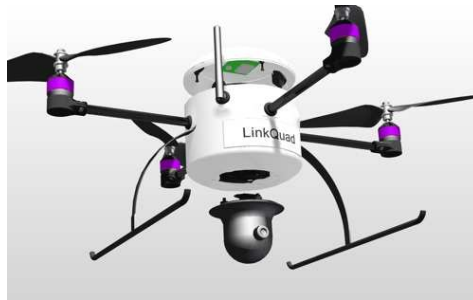


Figure 1.2. The LinkQuad development platform.

1.3 Previous work

The problem of 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 [2, 43]. 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 SUAV's generally are equipped with, although successful implementations do exist, e.g. [37]. The LinkQuad is in that regard, with its distributed processing power, well suited as a development platform when studying this problem.

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 information. This problem is studied in e.g. [27, 5], but perhaps the most interesting results are obtained in [2, 43], where the ideas from [21] of using monocular SLAM are implemented on a UAV platform and excellent results are obtained in terms of achieving real flight, although some problems remain regarding the long-term stability of the controller.

A landing control scheme, inspirational to the approach in this thesis, is suggested by [5], which is summarized in Section 5.1.4.

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 [3]. Continued effort of quadrotor modelling have however shown great potential[4].

1.3.2 Visual SLAM

A first take on a Visual SLAM (VSLAM) algorithm is presented in [20], resting on the foundation of a Rao-Blackwellized particle filter with Kalman filter banks. The algorithm presented in [20] also extends to the case of multiple cameras, which could be interesting in a longer perspective. Similar approaches, using common Kalman Filtering solutions have been implemented by e.g. [8, 10], and are available open-source⁵.

An alternative approach to VSLAM is suggested by [21] in the PTAM - Parallel Tracking And Mapping - library it presents. 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. A modified version of the said library has also demonstrationaly been implemented on an iPhone [22]. 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 SUAV.

The algorithm proposed in [21] uses selected keyframes from which offsets are calculated continuously in the tracking thread, while the map optimization problem is addressed separately as fast as possible using information only from these keyframes. This opposed to the traditional VSLAM filtering solution where each frame has to be used for continuous filter updates. Several existing implementations exist with this technique, e.g. as described by [8].

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[39]. Again, this is the method on which [43] bases its implementation.

1.4 Objectives and Limitations

The main objective for this thesis was to create a high-level control and state-estimation system and to demonstrate this by performing autonomous landing with the LinkQuad. To achieve this, the main work was put into achieving theoretically solid positioning and control. Having control over the vehicle, landing is

⁵<http://www.doc.ic.ac.uk/~ajd/Scene/>

then a matter of generating a suitable trajectory and detecting the completion of the landing.

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 modelling 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 developed for the *CRAP* framework include, but is far from limited to;

- General non-linear filtering, using EKF or UKF,
- General non-linear control, implemented as affine quadratic control,
- Extendable scheduling and reasoning through state-machines,
- Real-time plotting,
- 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 modelling available, and is presented in a scalable, general manner.

The state-estimation proposed in this thesis uses the full physical model derived in Chapter 2. While the model still needs tuning, it does show promising results, and a physically modelled quadrotor could potentially improve in-flight performance.

In Chapter 2, a general method for retaining the world-to-PTAM transform is proposed. This method could prove useful for extending the utility of the PTAM camera positioning library to more than its intended use in Augmented Reality. The utility for this has already been proven in e.g. [43], and providing the theory and implementation for this, as well as a proposed initialization routine, could be of great use in future work. The PTAM library has also been extended to full autonomy with the proposition of an automated initialization procedure as well as providing full detachment from the graphical interface.

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

1.6 Thesis Outline

Following the introductory Chapter 1, four chapters are devoted to presenting the theory and the equations used in the implementation, in order;

- **State Estimation** State estimation theory and physical modelling of a quadrotor,
- **Non-linear Control** Non-linear control theory and its implementation,
- **Monocular SLAM** Video-based SLAM and its applications.

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 then presented in Chapter 8.

A detailed description of the *CRAP* framework is appended to the thesis.

Chapter 2

State Estimation

A central part of automatic control is to know the state of the device you are controlling. The system studied in this thesis - the LinkQuad - is in constant motion, so determining the up-to-date position is of vital importance. This chapter deals with the estimation of the states relevant for positioning and controlling the LinkQuad. Filter theory and notation is established in Section 2.1.

In this thesis, an Unscented Kalman Filter (UKF) and an Extended Kalman Filter (EKF) both are evaluated. These filters both extends the linear Kalman filter theory to the non-linear 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 2.1 and 2.2 respectively.

The motion model of the system is derived and discussed in Section 2.3.

As well as beeing modelled, the motions of the system are captured by the on-board sensors. A measurement y is related to the motion model by the sensor model h ;

$$y(t) = h(x(t), u(t), t) \quad (2.1)$$

The models for the sensors used in this work are discussed in Section 2.4.

2.1 The Filtering Problem

The problem of estimating the state of a system - in this case its position, orientation, velocity etc. - is in general filtering 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 behaviour 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. (2.2) (f_c denoting the continous case). The system is also assumed to be subject to an additive white Gaussian noise $v(t)$ 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 real-world-system.

$$\dot{x}(t) = f_c(x(t), u(t), t) + v_c(t) \quad (2.2)$$

With numeric or analytical solutions, we can obtain the discrete form of (2.2), where only the sampling times are considered. The control signal, $u(t)$, is for instance assumed to be constant in the time interval, and we obtain the next predicted state directly, yielding the prediction of \hat{x} at the time t given the information at time $t - 1$ ¹. This motivates the notation used in this thesis - $\hat{x}_{t|t-1}$.

$$x_{t|t-1} = f(x_{t-1|t-1}, u_t, t) + v(t) \quad (2.3)$$

In the ideal case, a simulation of a prediction \hat{x} would with the prediction model in (2.3) fully describe the evolution of the system. To be able to provide a good estimate in the realistic case, however, we must also feed back measurements given from sensors measuring properties of the system.

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

$$\nu_t = y_t - \hat{y}_t \quad (2.4)$$

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 .

$$\hat{y}_t = h(\hat{x}_t, u_t, t) + w(t) \quad (2.5)$$

The innovation is then fused with the prediction to yield a new estimation of x given the information available as of the time t [13].

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t \nu_t \quad (2.6)$$

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

Algorithm 1 (Kalman Filter) *For a linear system, given predictions and measurements with known covariances Q and R , the optimal solution to the filtering problem is given by the forward recursion²*

Prediction update

$$\hat{x}_{t|t-1} = A\hat{x}_{t-1|t-1} + Bu_t \quad (2.7a)$$

$$P_{t|t-1} = AP_{t-1|t-1}A^T + Q \quad (2.7b)$$

¹Note that we have not yet performed any measurements that provide information about the state at time t .

²As first suggested by Rudolf E. Kálmán in [19].

Measurement update

$$K = P_{t|t-1} H^T (H P_{t|t-1} H^T + R)^{-1} \quad (2.8a)$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K (y - H \hat{x}_{t|t-1}) \quad (2.8b)$$

$$P_{t|t} = P_{t|t-1} + K H P_{t|t-1} \quad (2.8c)$$

Because of the assumptions on the noise and of 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.

$$x \sim \mathcal{N}(\hat{x}, P_{xx}) \quad (2.9)$$

Conditioned on the state and measurements before time k (kept in the set \mathcal{Y}^k), the covariance of the sample distribution is defined as

$$P_{xx}(t|k) = E \left[\{x(t) - \hat{x}_{t|k}\} \{x(t) - \hat{x}_{t|k}\}^T | \mathcal{Y}^k \right]. \quad (2.10)$$

As new measurements are taken, the covariance of the state evolves with the state estimate as in Eq. (2.11) [18].

$$P_{xx}(t|t) = P_{xx}(t|t-1) - K_t P_{\nu\nu}(t|t-1) K_t^T \quad (2.11)$$

$$P_{\nu\nu}(t|t-1) = P_{yy}(t|t-1) + R(t). \quad (2.12)$$

With known covariances, K can be chosen optimally for the linear case as derived in e.g. [13];

$$K_t = P_{xy}(t|t-1) P_{\nu\nu}^{-1}(t|t-1). \quad (2.13)$$

Note that (2.13) is another, albeit equivalent, way of calculating (2.8a), which will be exploited in Section 2.2.

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

One problem with the non-linear case is how to propagate the uncertainty, as described by the covariance, through the prediction and measurement models. 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 non-linear propagation, which generally does not preserve this property.

As the linear case is simple however;

$$P_{xx}(t|t-1) = A P(t|t) A^T + Q_t; \quad (2.14)$$

the novel approach is to linearize the system to yield A in every timestep. This method is called the Extended Kalman Filter, and is considered the de facto standard non-linear filter[17].

Algorithm 2 (Extended Kalman Filter) *The Kalman filter in Algorithm 2 is in the Extended Kalman filter extended to the non-linear case by straight-forward linearization where necessary.*

Prediction update

$$\hat{x}_{t|t-1} = f(\hat{x}_{t-1|t-1}, u_t) \quad (2.15a)$$

$$P_{t|t-1} = AP_{t-1|t-1}A^T + Q \quad (2.15b)$$

Measurement update

$$K = P_{t|t-1}H^T (HP_{t|t-1}H^T + R)^{-1} \quad (2.16a)$$

$$\hat{x}_{t|t} = \hat{x}_{t|t-1} + K(y - h(\hat{x}_{t|t-1})) \quad (2.16b)$$

$$P_{t|t} = P_{t|t-1} + KHP_{t|t-1}, \quad (2.16c)$$

where

$$A = \left. \frac{\partial f(x, u)}{\partial x} \right|_{x=\hat{x}_{t|t}}, \quad H = \left. \frac{dh(x)}{dx} \right|_{x=\hat{x}_{t|t-1}}. \quad (2.17)$$

This linearization, some argue[18], fails to capture the finer details of highly non-linear 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 2.2.

2.2 The Unscented Kalman Filter

The basic version of the Unscented Kalman Filter was proposed in [18] based on the following intuition [18]

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 non-linear 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 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) [42]. The SUT uses a minimal set of sigma points needed to describe the first two moments of the propagated distribution - two for each dimension (n) of the state vector and one for the mean.

$$\begin{aligned} \mathcal{X}_0 &= \hat{x} \\ \mathcal{X}_1 &= \hat{x} + \left(\sqrt{(n+\lambda)P_{xx}} \right)_i & i = 1, \dots, n \\ \mathcal{X}_i &= \hat{x} - \left(\sqrt{(n+\lambda)P_{xx}} \right)_i & i = n+1, \dots, 2n \end{aligned} \quad (2.18)$$

Variable	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 non-linearities.
β	2	As discussed in [16], 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 centerpoint and scale the distribution of sigma points.

Table 2.1. Description of the parameters used in the SUT.

$$\begin{aligned}
W_0^m &= \frac{\lambda}{n+\lambda} & W_0^c &= \frac{\lambda}{n+\lambda} + (1 - \alpha^2 + \beta) \\
W_i^m &= W_i^c = \frac{1}{2(n+\lambda)} & i &= 1, \dots, 2n
\end{aligned} \tag{2.19}$$

$$\lambda = \alpha^2(n + \kappa) - n \tag{2.20}$$

The three parameters introduced here, α , β and κ are summarized in Table 2.2. 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 non-linear prediction function and the resulting mean and covariance can be calculated.

$$\mathcal{X}_i^+ = f(\mathcal{X}_i, u, t) \quad i = 0, \dots, 2n \tag{2.21}$$

$$\hat{x} = \sum_{i=0}^{2n} W_i^m \mathcal{X}_i^+ \tag{2.22}$$

$$P_{xx} = \sum_{i=0}^{2n} W_i^c \{ \mathcal{X}_i^+ - \hat{y} \} \{ \mathcal{X}_i^+ - \hat{y} \}^T \tag{2.23}$$

For the measurement update, similar results are obtained, and the equations (2.26)-(2.27) can be connected to equations (2.12)-(2.13).

$$\mathcal{Y}_i = h(\mathcal{X}_i, u, t) \quad i = 0, \dots, 2n \tag{2.24}$$

$$\hat{y} = \sum_{i=0}^{2n} W_i^m \mathcal{Y}_i \tag{2.25}$$

$$P_{yy} = \sum_{i=0}^{2n} W_i^c \{ \mathcal{Y}_i - \hat{y} \} \{ \mathcal{Y}_i - \hat{y} \}^T \tag{2.26}$$

$$P_{xy} = \sum_{i=0}^{2n} W_i^c \{ \mathcal{X}_i - \hat{x} \} \{ \mathcal{Y}_i - \hat{y} \}^T \quad (2.27)$$

As can be seen from the equations in 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 state³, be it for the measurement functions, discussed in Section 2.4, or the time update prediction function - the motion model.

2.3 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, and evaluating the plausibility of each measurement 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 2.1-2.2, where the probability estimate of the sensors' measurements are based on the motion model's prediction. In this section, a motion model is derived and evaluated. The model is presented in its continuous form, since the discretization is made numerically on-line.

2.3.1 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 chosen 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 explicitly stated otherwise. This frame is often referred to as the "world" coordinate system, or " w " for short in equations in 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 z -axis in the downward direction - as depicted in Figure 2.2.

Propeller fixed Each of the propellers are associated with their own frame of reference, P_i , which tracks the virtual tilting of the thrust vector due to flapping, discussed in Section 2.3.3.

³It is not, however, merely a central difference linearization of the functions, as [18] notes.

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 2.4.5.

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 2.1-2.2.

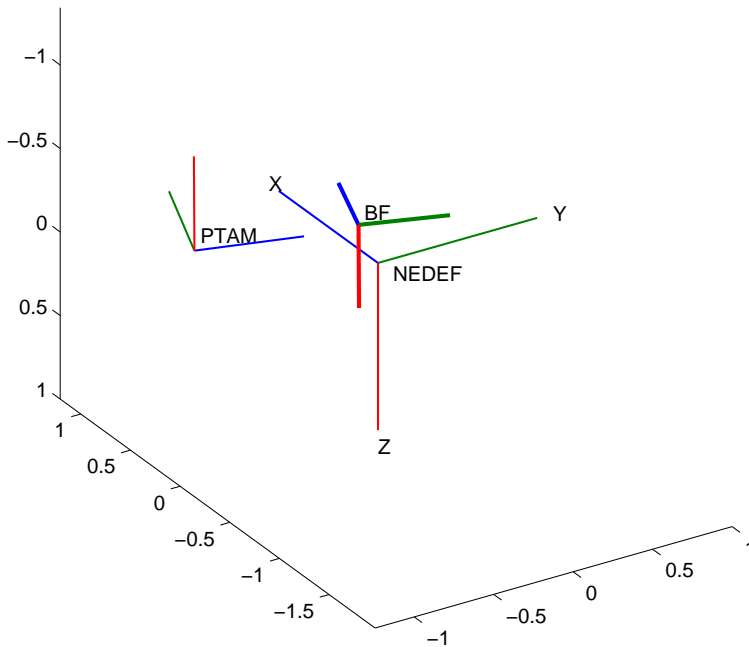


Figure 2.1. The NEDEF, body-fixed and the PTAM coordinate frames are of global importance.

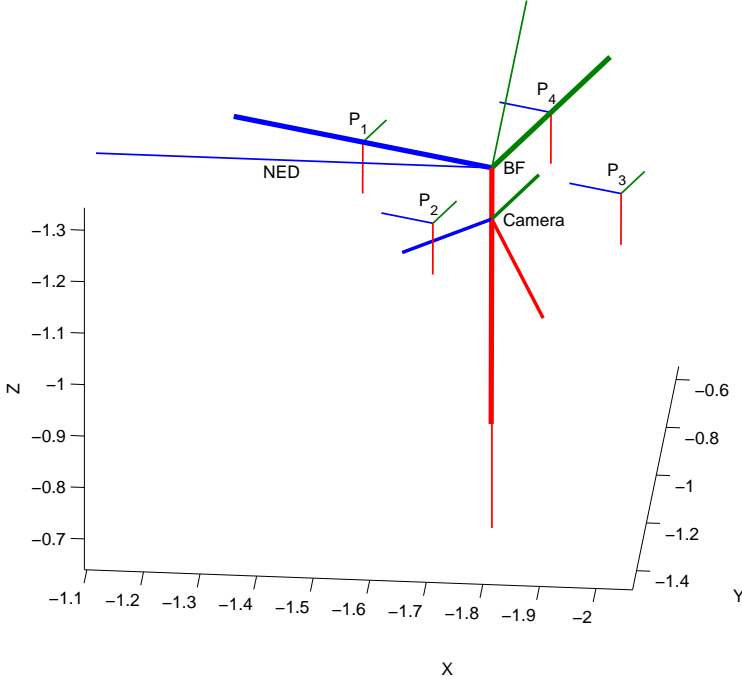


Figure 2.2. Locally, on the quadrotor, there are several coordinate frames used in the thesis. Here, the IMU frame coincides with the body-fixed frame.

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 height h and distance d from the center of gravity as follows

$$D_0 = (d, 0, h)^{BF} \quad (2.28)$$

$$D_1 = (0, -d, h)^{BF} \quad (2.29)$$

$$D_2 = (-d, 0, h)^{BF} \quad (2.30)$$

$$D_3 = (0, d, h)^{BF} \quad (2.31)$$

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 the quaternion q , corresponding to the matrix rotation [23]⁴ given by

$$\begin{pmatrix} q_1^2 + q_i^2 - q_j^2 - q_k^2 & 2q_iq_j - 2q_1q_k & 2q_1q_k + 2q_1q_j \\ 2q_iq_j + 2q_1q_k & q_1^2 - q_i^2 + q_j^2 - q_k^2 & 2q_jq_k - 2q_1q_i \\ 2q_1q_k - 2q_1q_j & 2q_jq_k + 2q_1q_i & q_1^2 - q_i^2 - q_j^2 + q_k^2 \end{pmatrix}. \quad (2.32)$$

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

Rotation quaternions describing the rotation *to frame a from frame b* is commonly denoted q^{ab} , whereas the b may be dropped if the rotation is global, i.e. relative the NEDEF system. a and b are, where unambiguous, replaced by the first character of the name of the reference frame. Full transformations between coordinate systems - including rotation, translation and scaling - are similarly denoted \mathcal{J}^{ab} .

2.3.2 Kinematics

The motions of the quadrotor are described by the following relations [34]:

$$\dot{\xi} = V \quad (2.33a)$$

$$\begin{pmatrix} \dot{q}_0^{wb} \\ \dot{q}_i^{wb} \\ \dot{q}_j^{wb} \\ \dot{q}_k^{wb} \end{pmatrix} = -\frac{1}{2} \begin{pmatrix} 0 & -\omega_x & -\omega_y & -\omega_z \\ \omega_x & 0 & -\omega_z & \omega_y \\ \omega_y & \omega_z & 0 & -\omega_x \\ \omega_z & -\omega_y & \omega_x & 0 \end{pmatrix} \begin{pmatrix} q_0^{wb} \\ q_i^{wb} \\ q_j^{wb} \\ q_k^{wb} \end{pmatrix} \quad (2.33b)$$

In practice, a normalization step also has to be added to account for the unit length constraint on rotation quaternions.

2.3.3 Dynamics

The motions of the quadrotor can be fully mathematically explained by the forces and moments acting on the vehicle. Using the rigid-body assumption, Eulers' extension of Newton's laws of motion for the quadrotor's center of gravity, \mathcal{G} , yields

$$\dot{V} = a_{\mathcal{G}}^w = R(q^{wb}) \frac{1}{m} \sum F \quad (2.34a)$$

$$\dot{\omega} = R(q^{wb}) I_{\mathcal{G}}^{-1} \sum M_{\mathcal{G}} \quad (2.34b)$$

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

$\sum_{i=0}^3 F_{ri}$ 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

$$F_g = mg \cdot e_3^{NED} \quad (2.35)$$

The following sections will describe the rotor thrust and wind forces respectively. Additionally, other minor forces and moments are discussed in Section 2.3.3.

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. The rotational velocity of the propeller is directly influenced by the controller. It may thus be modelled as a first order system - using the time constant τ_{rotor} with the reference velocity as input, as in Eq. (2.36). For testing purposes, or where the control signal is not available, Eq. (2.37) may be used instead.

$$\dot{\omega}_{ri} = \frac{1}{\tau_{rotor}} (\omega_{ri} - r_i) \quad (2.36)$$

$$\dot{\omega}_{ri} = 0 \quad (2.37)$$

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. [34]. 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 quadrotors 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 equations (2.38a)-(2.39) [34, 35, 24].

The momenta induced by the propeller rotation and thrust are described in equations (2.38b)-(2.38c). All equations in this section are given in the body-fixed coordinate system.

$$F_{ri} = C_T \rho A_r R^2 \omega_{ri}^2 \begin{pmatrix} -\sin(a_{1si}) \\ -\cos(a_{1si}) \sin(b_{1si}) \\ -\cos(a_{1si}) \cos(b_{1si}) \end{pmatrix} \quad (2.38a)$$

$$M_{Qi} = -C_Q \rho A R^3 \omega_{ri} |\omega_{ri}| e_3^{\text{NED}} \quad (2.38b)$$

$$M_{ri} = F_{ri} \times D_{ri} \quad (2.38c)$$

The equations for the flapping angles (a_{1si}, b_{1si}) are derived in [34, 35, 24], but are in (2.39) extended to include the velocity relative to the wind. $V_{ri(n)}$ denotes the n'th element of the vector V_{ri} .

$$V_{\text{rel}} = V - V_{\text{wind}} \quad (2.39a)$$

$$V_{ri} = V_{\text{rel}} + \Omega \times D_{ri} \quad (2.39b)$$

$$\mu_{ri} = \frac{\|V_{ri(1,2)}\|}{\omega_i R} \quad (2.39c)$$

$$\psi_{ri} = \arctan \left(\frac{V_{ri(2)}}{V_{ri(1)}} \right) \quad (2.39d)$$

$$\begin{pmatrix} a_{1s} i \\ b_{1s} i \end{pmatrix} = \begin{pmatrix} \cos(\psi_{ri}) & -\sin(\psi_{ri}) \\ \sin(\psi_{ri}) & \cos(\psi_{ri}) \end{pmatrix} \begin{pmatrix} \frac{1}{1 - \frac{\mu_{ri}^2}{2}} \mu_{ri} (4\theta_{twist} - 2\lambda_i) \\ \frac{1}{1 + \frac{\mu_{ri}^2}{2}} \frac{4}{3} \left(\frac{C_T}{\sigma} \frac{2}{3} \frac{\mu_{ri} \gamma}{a} + \mu_{ri} \right) \end{pmatrix} + \begin{pmatrix} \frac{-\frac{16}{\gamma} \left(\frac{\omega_{\theta}}{\omega_{ri}} \right) + \left(\frac{\omega_{\psi}}{\omega_{ri}} \right)}{1 - \frac{\mu_{ri}^2}{2}} \\ \frac{-\frac{16}{\gamma} \left(\frac{\omega_{\psi}}{\omega_{ri}} \right) + \left(\frac{\omega_{\theta}}{\omega_{ri}} \right)}{1 + \frac{\mu_{ri}^2}{2}} \end{pmatrix} \quad (2.39e)$$

$$\lambda_i = \mu \alpha_{si} + \frac{v_{1i}}{\omega_i R} \quad (2.39f)$$

$$v_{1i} = \sqrt{-\frac{V_{rel}^2}{2} + \sqrt{\left(\frac{V_{rel}^2}{2}\right)^2 + \left(\frac{mg}{2\rho A_r}\right)^2}} \quad (2.39g)$$

$$C_T = \frac{\sigma a}{4} \left\{ \left(\frac{2}{3} + \mu_{ri}^2 \right) \theta_0 - \left(\frac{1}{2} + \frac{\mu^2}{2} \right) \theta_{twist} + \lambda \right\} \quad (2.39h)$$

$$\alpha_{si} = \frac{\pi}{2} - \arccos \left(-\frac{V_{rel} \cdot e_z}{\|V_{rel}\|} \right) \quad (2.39i)$$

$$C_Q = \sigma a \left[\frac{1}{8a} (1 + \mu_{ri}^2) \bar{C}_d + \lambda \left(\frac{1}{6} \theta_0 - \frac{1}{8} \theta_{twist} + \frac{1}{4} \lambda \right) \right] \quad (2.39j)$$

Wind

For describing the wind's impact on the quadrotor motion, a simple wind model is applied where the wind is modelled with a static velocity that imposes forces and moments on the quadrotor. The wind velocity vector is estimated by the observer and may thus still vary in its estimation through the measurement update. The wind velocities in the filter are given in the NEDEF reference frame.

The wind drag force is calculated using equation (2.40), whereas the moments are given by equations (2.41). 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.

$$F_{wind} = F_{wind,body} + \sum_{i=0}^3 F_{wind,ri} \quad (2.40a)$$

$$F_{wind,body} = -\frac{1}{2} C_D \rho A V_{rel} \|V_{rel}\| \quad (2.40b)$$

$$F_{wind,ri}^{BF} = -\frac{1}{2} \rho C_{D,r} \sigma A_r (V_{ri} \cdot e_{P_{ri}3}^{BF}) \|V_{ri} \cdot e_{P_{ri}3}^{BF}\| e_{P_{ri}3}^{BF} \quad (2.40c)$$

Symbol	Expression	Description	Unit
a	$\frac{dC_L}{d\alpha} \approx 2\pi$	Slope of the lift curve.	$\frac{1}{\text{rad}}$
α_{si}	-	Propeller 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_L	-	Coefficient of lift.	1
C_T	*	Coefficient of thrust. This is primarily the scaling factor for how the thrust is related to the square of ω_i , as per Eq. 2.38a.	1
C_{T0}	-	Linearization point for thrust coefficient.	1
C_Q	*	Torque coefficient. This constant primarily is the scaling factor relating the square of ω_i to the torque from each rotor.	1
γ	$\frac{\rho ac R^4}{I_b}$	γ is the Lock Number [24], 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_i	*	λ_i denotes the air inflow to the propeller.	1
R	-	Rotor radius.	m
ρ	-	Air density.	$\frac{\text{kg}}{\text{m}^3}$
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
$\omega_\phi, \omega_\theta, \omega_\psi$	-	The rotational, body-fixed, velocity of the quadrotor.	$\frac{\text{rad}}{\text{s}}$
ω_{ri}	-	The rotational velocity of propeller i .	$\frac{\text{rad}}{\text{s}}$
μ_{ri}	-	The normalized, air-relative, blade tip velocity.	1

Table 2.2. 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_{Dr}	-	Propeller's coefficient of drag.

Table 2.3. Table of symbols used in the wind equations

$$M_{\text{wind}} = M_{\text{wind,body}} + \sum_{i=0}^3 M_{\text{wind},ri} \quad (2.41a)$$

$$M_{\text{wind,body}} \approx 0 \quad (2.41b)$$

$$M_{\text{wind},ri} = D_{ri}^{BF} \times F_{\text{wind},ri}^{BF} \quad (2.41c)$$

The wind model applied in this thesis is a decaying model that tends towards zero if no measurements tell otherwise. This decaying model is presented in Eq. (2.42).

$$\dot{V}_{\text{wind}} = -\epsilon V_{\text{wind}} \quad (2.42)$$

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 [4]. Unless where explicitly noted, annotation is similar to Section 2.3.3.

Hub Force

$$C_H = \sigma a \left[\frac{1}{4a} \mu \bar{C}_d + \frac{1}{4} \lambda \mu \left(\theta_0 + \frac{\theta_{twist}}{2} \right) \right] \quad (2.43)$$

$$F_{\text{hub},i} = -C_H \rho A R^2 \omega_i^2 \hat{x} \quad (2.44)$$

$$M_{\text{hub},i} = D_i \times F_{\text{hub},i} \quad (2.45)$$

Rolling Moment

$$C_{\text{RM}} = -\sigma a \mu \left[\frac{1}{6} \theta_0 + \frac{1}{8} \theta_{twist} - \frac{1}{8} \lambda_i \right] \quad (2.46)$$

$$M_{\text{RM},i} = C_{\text{RM}} \rho A R^3 \omega_i^2 \quad (2.47)$$

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.

$$T_{\text{IGE}} = \frac{1}{1 - \frac{R^2}{16z^2}} T \quad (2.48)$$

Gyro Effects and Counter-Torque I_{rotor} is the propeller inertia.

$$\begin{pmatrix} \dot{\omega}_{\theta}\dot{\omega}_{\psi}(I_{yy} - I_{zz}) + I_{\text{rotor}}\dot{\omega}_{\theta} \sum_{i=0}^4 \omega_{ri} \\ \dot{\omega}_{\theta}\dot{\omega}_{\psi}(I_{zz} - I_{xx}) + I_{\text{rotor}}\dot{\omega}_{\theta} \sum_{i=0}^4 \omega_{ri} \\ \dot{\omega}_{\theta}\dot{\omega}_{\phi}(I_{xx} - I_{yy}) + I_{\text{rotor}} \sum_{i=0}^4 \dot{\omega}_{ri} \end{pmatrix} \quad (2.49)$$

2.4 Sensor Models

This Section relates the estimated state of the quadrotor to the expected sensor measurements, \hat{y} . The first four sensors - accelerometers, gyroscopes, magnetometers and the pressure sensor - are described quite briefly, while the estimation of the camera's frame of reference requires some more detail.

It is common to include a GPS, providing world-fixed measurements, to prevent drift in the filtering process. Here, the GPS is replaced with a camera, which will be discussed in Section 2.4.5.

In most equations below, a zero-mean Gaussian term 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.

2.4.1 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 centre of gravity used as centre of the measured vehicle, which motivates correction for angular acceleration and rotation.

$$\hat{y}_{\text{acc}} = a_{\mathcal{G}} + \dot{\omega} \times r_{\text{acc}/\mathcal{G}} + \omega \times (\omega \times r_{\text{acc}/\mathcal{G}}) + e_{\text{acc}} \quad (2.50)$$

2.4.2 Gyroscopes

Gyroscopes, or rate gyroscopes specifically, measure the angular velocity of the sensor. Unlike acceleration, the angular rate is invariant of the relative position of the gyro to the center of gravity. However, gyroscope measurements are associated with a bias which may change over time. This bias term is introduced as a state variable which is modelled constant, Eq. (2.52), through the time update, leaving its adjustment to the filter measurement update.

$$y_{\text{gyro}} = \omega + b + e_{\text{gyro}} \quad (2.51)$$

$$\dot{b} = 0 + e_{\text{bias}} \quad (2.52)$$

2.4.3 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.

$$y = R(q)m^e + e_m \quad (2.53)$$

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	$\frac{\text{J}}{\text{mol}\cdot\text{K}}$
T_0	Standard temperature at sea level.	288.15	K

Table 2.4. Table of Symbols used in the pressure equation, (2.55)

The Earth’s magnetic field can be approximated using the World Magnetic Model[7], which for Linköping, Sweden, yields

$$m^e = \begin{pmatrix} 15.7 & 1.11 & 48.4 \end{pmatrix}_{NEDEF}^T \mu T. \quad (2.54)$$

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.

2.4.4 Pressure Sensor

The air pressure, p , can be related to altitude using the following equation[31]

$$p = p_0 \left(1 - \frac{L \cdot h}{T_0} \right)^{\frac{g \cdot M}{R \cdot L}} + e_p \quad (2.55)$$

2.4.5 Camera

To estimate the position of the camera using the captured images, the PTAM-library is used. Because the main application of the PTAM library is reprojection of augmented reality into the image, consistency between a metric world-fixed coordinate frame (such as the NEDEF-system used on the LinkQuad), and the internally used coordinate system is not of importance - and thus not implemented in the PTAM positioning.

The measurements from the camera consists of the transform from the PTAM “world”-coordinates to the camera lens, in terms of

- translation, X^{PTAM} ,
- and orientation, $q^{\text{PTAM},c}$.

However, since the quite arbitrary[21] coordinate system of PTAM is neither of the same scale nor aligned with the quadrotor coordinate system, we need to estimate the affine transformation between the two in order to get useful results.

The transformation is characterized by

- a translation T to the origin, $\mathcal{O}_{\text{PTAM}}$,
- a rotation R by the quaternion q^{Pw} ,
- and a scaling S by a factor s .

These are collected to a single transformation in Eq. 2.56, forming the full transformation from the global NEDEF system to the PTAM coordinate frame.

$$x^{\text{PTAM}} = \underbrace{S(s)R(q^{Pw})T(-\mathcal{O}_{\text{PTAM}})}_{\triangleq \mathcal{J}^{Pw}, \text{transformation from camera to PTAM}} x^{\text{NEDEF}} \quad (2.56)$$

E.g. [14] studies this problem in the offline case, whereas the method used in this thesis extends the idea to the on-line case where no ground truth is available. This is performed by using the first measurement to construct an initial guess which then is filtered through time using the observer.

While PTAM exhibit very stable positioning, it has a tendency to move its origin due to association errors. To provide stable position measurements, we need to detect these movements and adjust the camera transformation accordingly. Initialization and tracking is dealt with in Section 2.4.5 and 2.4.5 respectively, whereas the teleportation problem is discussed in Section 2.4.5.

Initialization

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

First, however, the orientation of the PTAM coordinate system is calculated from the estimated quadrotor orientation and the measurement in the PTAM coordinate frame;

$$q^{Pw} = q^{PTAM,c} q^{cb} q^{bw}. \quad (2.57)$$

q^{bc} , the inverse of q^{cb} of Equation 2.57, describes the rotation from camera coordinates to body-fixed coordinates, taking into account the differing definitions between the PTAM library camera coordinate system and that used in this thesis. With known camera pitch and yaw - Θ_c and Ψ_c respectively - this corresponds to four consecutive rotations, given in Eq. (2.58) as rotations around gives axes.

$$q^{bc} = \text{rot}(\Theta_c, \hat{y}) * \text{rot}(\Psi_c, \hat{z}) * \text{rot}(\pi/2, \hat{z}) * \text{rot}(\pi/2, \hat{x}) \quad (2.58)$$

To determine the distance to this plane according to the current estimation, Equation (2.59) is solved for λ in accordance with Eq. 2.60.

$$\begin{cases} \mathcal{O}_{\text{PTAM}} &= \xi + R(q^{wb})r_{\text{camera}/g} + \lambda R(q^{wP}) \frac{x^{\text{PTAM}}}{\|x^{\text{PTAM}}\|} \\ \mathcal{O}_{\text{PTAM}} \cdot \hat{z} &= 0 \end{cases} \quad (2.59)$$

$$\lambda = - \frac{(\xi + R(q^{wb})r_{\text{camera}/\mathcal{G}}) \cdot \hat{z}}{\left(R(q^{wP}) \frac{X^{PTAM}}{|X^{PTAM}|}\right) \cdot \hat{z}} \quad (2.60)$$

By comparing the approximated distance with the distance measured in the PTAM coordinate system, we obtain a starting guess for the scaling factor, s ;

$$s = \frac{|X^{PTAM}|}{|\lambda|}. \quad (2.61)$$

Together, these parameters form an initial estimate of the transformation connecting the PTAM reference frame. This estimate could be inserted into the global observer filter - introducing eight new states containing q^{Pw} , s and \mathcal{O}_{cam} - although little benefit of this has been observed in simulated testing. Because the PTAM coordinate system is defined fixed in the global NEDEF coordinate system, the transform parameters are unchanged in the observer's time update.

Continuous Refinement

The measurement update is made separate from the update of the inertial sensors, using the measurement equations in (2.62), expanding the equation derived in Eqs. (2.56) and (2.57).

$$\hat{X}^{\text{PTAM}} = \mathcal{J}^{Pw}(\xi + R(q^{wb})r_{\text{camera}/\mathcal{G}}) + e_{\text{PTAM},X} \quad (2.62a)$$

$$\hat{q}^{\text{PTAM},c} = q^{Pw} q^{wb} q^{bc} + e_{\text{PTAM},q} \quad (2.62b)$$

Teleportation

The PTAM tracking may sometimes exhibit a “teleporting” behaviour. 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 behaviour is detected using simple thresholding, as in Eq. (2.63), although no action is currently implemented to recover. In Eq. (2.63), the estimated motion is scaled by the factor s from the tranformation from PTAM coordinates, and thresholded by a configurable parameter ϵ .

$$|X_t^{\text{PTAM}} - X_{t-1}^{\text{PTAM}}| \cdot s > \epsilon \quad (2.63)$$

Chapter 3

Non-linear Control

To control a quadrotor's movements, a non-linear controller is applied to the physical system, using a model of the system to calculate the best (in a sense well defined in this chapter) signals of control to each of the engines driving the propellers.

The controller approach chosen in this thesis is based on the Linear Quadratic (LQ) controller, the theory of which is presented in Section 3.1. An extension to the technique of *gain-scheduling* is discussed in Section 3.2.

The physical model of the system was derived in Section 2.3. In Section 3.3, 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.

3.1 The Linear Quadratic Controller

In this section, the theory of Linear Quadratic control is presented, considering the continuous linear plant in (3.1), with control signal u and reference r .

$$\dot{x} = Ax + Bu \quad (3.1a)$$

$$z = Mx \quad (3.1b)$$

$$e = z - r \quad (3.1c)$$

The basic LQ controller, described in e.g. [12], uses a linear state-space system model and weights 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 (3.2).

$$\mathcal{J} = \int_0^{\infty} e^T(t)Qe(t) + u^T(t)Ru(t)dt. \quad (3.2)$$

Thus, by varying the elements of the cost matrices Q and R respectively, the solution to the optimization will yield control signals that will steer 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, (3.2) can be solved analytically, resulting in a linear feedback,

$$u_t = -L\hat{x}_t \quad (3.3)$$

$$L = R^{-1}B^TS, \quad (3.4)$$

where S is the Positively Semi-Definite (PSD) solution to the Continuous Algebraic Riccati Equation (CARE)[12], stated in (3.5).

$$A^TS + SA + M^TQM - SBR^{-1}B^TS = 0 \quad (3.5)$$

To improve the reference following abilities of the controller, the reference may be brought into the control signal by a scaling matrix L_r , which is chosen so that the static gain of the system is equal to identity [12]. In the case of equal number of control signals as controlled states, Eqs. 3.6-3.7 are obtained.

$$u_t = -L\hat{x}_t + L_rr_t \quad (3.6)$$

$$L_r = [M(BL - A)^{-1}B]^{-1}. \quad (3.7)$$

3.2 State-Dependent Riccati Equations and LQ Gain Scheduling

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 non-linear control is 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 on-line is treated under the subject of *State-Dependent Riccati Equations*, SDRE's. The basic formulation of the problem is covered in e.g. [36], and an extensive survey is presented in [6]. Implementation details are detailed and evaluated in e.g. [11, 1, 38].

The LQ theory is extended to the non-linear case using the Taylor expansion of a general motion model is exploited to form the general result of Equation (3.9). The linearization of the motion model is presented in Equation (3.8).

$$\dot{x} = f(x, u) \approx \underbrace{f(x_0, u_0) + \left. \frac{\partial f}{\partial x} \right|_{\substack{x=x_0 \\ u=u_0}} \underbrace{(x-x_0)}_{\Delta x}}_A + \underbrace{\left. \frac{\partial f}{\partial u} \right|_{\substack{x=x_0 \\ u=u_0}} \underbrace{(u-u_0)}_{\Delta u}}_B \quad (3.8)$$

In the standard formulation of LQ, the linearization is made around a stationary point (x_0, u_0) , where $f(x_0, u_0) = 0$. In a more general formulation, it is possible to lift this constraint using a homogenous state [36];

$$\dot{X} = \begin{bmatrix} \dot{x} \\ 0 \end{bmatrix} = \begin{bmatrix} A & f(x_0, u_0) - Ax_0 \\ 0 & 0 \end{bmatrix} \underbrace{\begin{bmatrix} x \\ 1 \end{bmatrix}}_X + \begin{bmatrix} B \\ 0 \end{bmatrix} \Delta u. \quad (3.9)$$

Eq. 3.9 is a linear system for which the ordinary LQ problem can be solved, using Eq. (3.10)-(3.7).

The linearized output signal, Δu , is then added to u_0 to form the controller output, as in Equation (3.10).

$$u = u_0 + \Delta u = u_0 - L\bar{X} + L_r r \quad (3.10)$$

However, the linearizing extension of the affine controller in (3.9) 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 (3.2) 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 retain stability. This guarantees that (3.2) tends to zero.

Equation (3.9) is thus really implemented as in (3.11).

$$\dot{X} = \begin{bmatrix} \dot{x} \\ 0 \end{bmatrix} = \begin{bmatrix} A & f(x_0, u_0) - Ax_0 \\ 0 & -\mathbf{10}^{-9} \end{bmatrix} \underbrace{\begin{bmatrix} x \\ 1 \end{bmatrix}}_X + \begin{bmatrix} B \\ 0 \end{bmatrix} \Delta u. \quad (3.11)$$

3.3 Control Model

In Chapter 2, 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. (3.9) by providing the Jacobi matrices with regards to x and u , and removing states which will not be used in the controller.

The Jacobians of the system are aquired numerically by using central difference

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \quad (3.12)$$

While all states are of importance to the dynamics of the quadrotor, it should be noted that only a subset of the states in x is used for control. We thus need to form new matrices 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 (†).

The new matrices need to be formed only with the rows that are to be used for control. Extracting those rows from Equation (3.8) yields Equation (3.13).

$$\begin{aligned} \dot{\bar{x}} = \bar{f}(x, u) &\approx \bar{f}(x_0, u_0) + \bar{A}(x - x_0) + \bar{B}\Delta u \\ &= \bar{f}(x_0, u_0) - \bar{A}^\square \bar{x}_0 - \bar{A}^\dagger \bar{x}_0^\dagger + \bar{A}^\square \bar{x} + \bar{A}^\dagger \bar{x}^\dagger + \bar{B}\Delta u \\ &= \bar{f}(x_0, u_0) - \bar{A}^\square \bar{x}_0 + \bar{A}^\square \bar{x} + \bar{B}\Delta u \end{aligned} \quad (3.13)$$

In the last equality of Eq. (3.13), 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 Equation (3.13) can be directly fitted into Equation (3.9) to form the new matrices for the controller. Note also that the derivation in (3.13) is completely analogous in the discrete-time case.

Chapter 4

Monocular SLAM

In the interest of extracting positioning information from a video stream of a general, unknown, environment, the common approach is to use SLAM, *Simultaneous Localisation And Mapping*. In the mathematical SLAM framework, features 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 is often 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 increase 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 on movement in only a single camera to calculate the depth of the features in the image. 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 [15].

4.1 Filtering Based Solutions

One novel approach for camera tracking, used by for instance [8], is to use the EKF-SLAM framework and couple the feature and camera tracking in a time- and measurement update for each consecutive frame ([10] does this with FastSLAM 2.0). The *Scenelib* library described in [8] 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 arrives, the change of position of each feature is measured and the filter is updated accordingly. Thus, one must take care to avoid misassociation of the features, as this leads to false measurements that will mislead the filter.

It is trivial to include motion models into the filtering approaches, since the general algorithm utilize a classical state-based filter time-update.

4.2 Keyframe-based SLAM

A fundamentally different approach is presented in [21], where the focus is put 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 4.1 - and the new features detected 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 recorded keyframes which by extension gives the updated position of the camera.

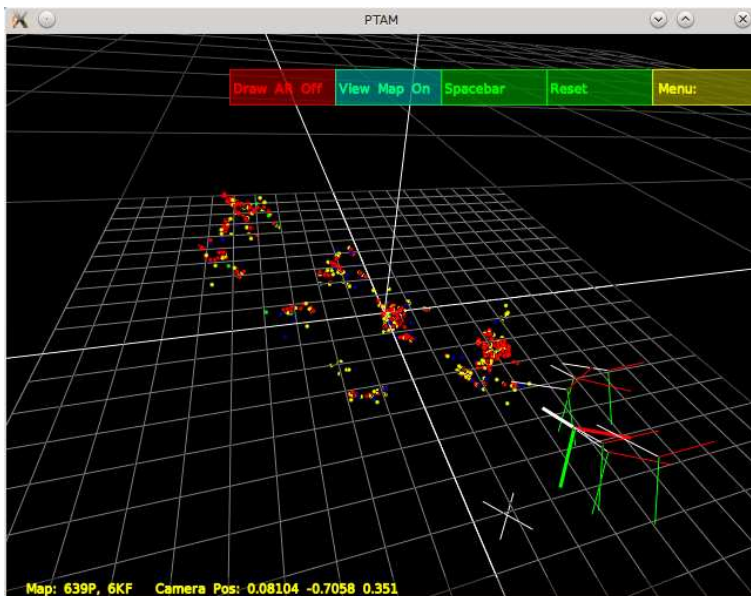


Figure 4.1. Keyframes containing features - here represented by dots - are recorded and the camera's position at the time of the frame's capture is stored and visualized as coordinate axes. The thick coordinate system displays the camera's current position estimate.

4.3 The PTAM Library

PTAM - Parallel Tracking And Mapping is a software library for camera tracking developed for Augmented Reality (AR) applications in small workspaces [21]. It has only recently been applied for quadrotor state estimation [43], although as the library is intended for AR applications, the connection with the world's coordinate system is loose, as duly discussed in Chapter 2.4.5. Several libraries exist extending the functionality of the PTAM library [30].

As recently published, new algorithms - presented in [29] - surpass the library's performance at the expense of computational cost and the demand for a high-profile graphics card. The performance is nonetheless proven¹, and improves the usability over preceding libraries, such as *Scenelib*, by including a camera calibration tool, estimating parameters describing the lens properties.

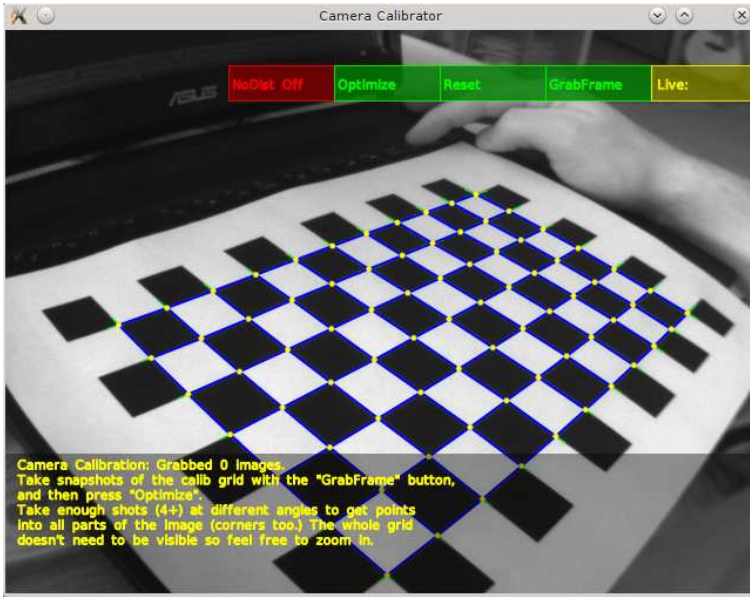


Figure 4.2. A checker-board pattern is used for calibrating the lens-specific parameters of the camera.

4.3.1 Operation

In the tracking procedure, the PTAM library randomly projects previously recorded features into the captured video frame, visualized in Figure 4.3. It could be argued that the feature projection in the PTAM library could make more use of external sensors and position estimates in this process. This, however, remains as future work.

¹Examples of projects using PTAM are listed at <http://ewokrampage.wordpress.com/projects-using-ptam/>

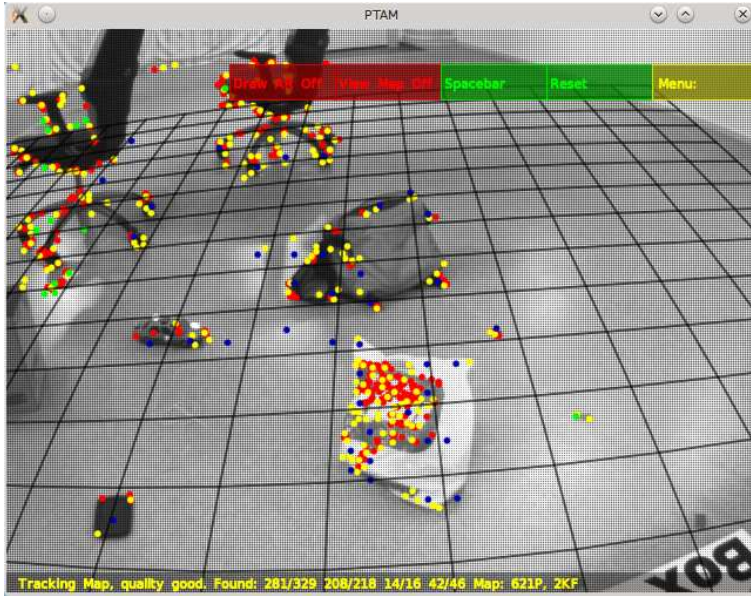


Figure 4.3. Features - visualized as dots - are projected into the image and sought for. The calculated offset from the features' original positions then yield a position estimate.

As for mapping; Instead of continuously updating the keyframe's position estimates - as in the filtering solution - the PTAM library presented in [21] separates this task from the tracking entirely to a parallel, less time-critical, processing thread². As this update does not have to be performed in real-time each video-frame, this parallelization allows for the use of a more advanced adjustment technique [21], namely Bundle Adjustment[26]. The bundle adjustment problem performs a non-linear least-squares optimization over the available keyframe positions, utilizing the sparse structure of the optimization problem that is solved to make the solution computationally tractable [25].

4.3.2 Modifications to the Library

The PTAM library is originally interfaced with a included graphical interface. However, as the source-code is provided for non-commercial use, the library was modified to interface over serial port with the CRAP framework and also extended with automatic initialization and full non-graphical use. This to enable full automatization on the development platform.

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 then captured, and features from the original image are tracked between the two to perform a stereo initialization to extract the ground plane and a placement for

²Hence its name; *Parallel Tracking And Mapping*

the origin. The tracked scene should be planar to retrieve a good estimation of the true ground plane, although the tracking will work regardless.



Figure 4.4. 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.

In the graphical interface, the tracked features are visualized during the process with lines in the camera image, as in Figure 4.4. 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 first³ 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 losing 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.

4.3.3 Practical Use

The PTAM library is designed to use a wide-angle lens, with performance dropping should such not be available [21]. In its current state however, PTAM does not support fish-eye lenses, and although the LinkQuad-mounted camera features a changeable lens, all available wide-angle lenses are fish-eye. It is possible to use the PTAM library with a standard lens, although tracking is easier lost.

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

³First in the list of features from the first image still found in the latest frame.

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 features tracked is also dependent on the processing power available, and because the complexity of the mapping problem grows fast with the number of keyframes, the library may have difficulties extending the map to unexplored areas.

Chapter 5

Finite State-Machines

In projects related to the LinkQuad quadrotor, generic state-machine frameworks have been developed and researched[28, 46]. On the LinkQuad, for the purpose of this thesis, a stripped down state-machine engine was implemented in the *CRAP* framework presented in Appendix A.

The state machine is responsible for transitioning between the states predefined in an action sequence (*mode*), ultimately providing reference signals for the controller. Figure 5.1 exemplifies the notation used in this chapter.

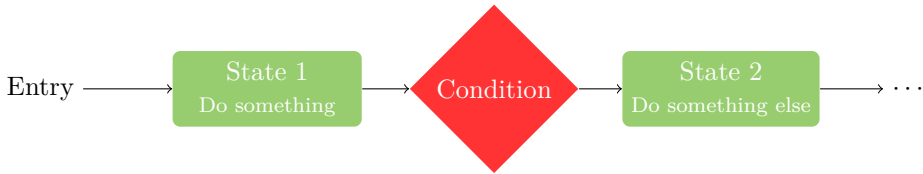


Figure 5.1. State-machine example. An active state is active and continuously invoked until a transition condition evaluates true. The next state is then activated.

5.1 Implemented Modes

In this Section, four basic modes are presented which were implemented in the timeframe of this thesis.

5.1.1 Hovering

In the hover mode, the position of the quadrotor is noted at the time of the activation, and three independent PID-controllers are initialized to generate reference signals to the main controller, working to keep the quadrotor at the place where it was first initialized.

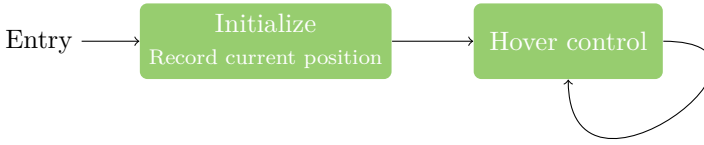


Figure 5.2. Hovering scheme. The position of the quadrotor is noted when the mode is activated. The hover state is then immediately activated to keep the quadrotor in this position.

5.1.2 PTAM initialization

Entering this mode starts an automated initialization process to set up the initial PTAM coordinate system. Commands are sent to the PTAM module to initialize tracking, after which the quadrotor should be moved sideways for the stereo initialization performed by the PTAM library.

5.1.3 Free Flight

In the free flight mode, the control reference is forwarded from the joystick reference provided over the serial interface from the ground station.

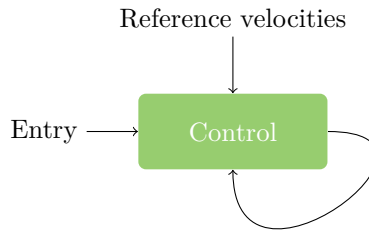


Figure 5.3. Freeflight scheme. The freeflight mode merely forwards controller reference signals received from the interface.

5.1.4 Landing

There have been several studies of autonomous quadrotor landing, in e.g.[27, 5]. [5] implements a landing scheme closely related to that which is proposed in this thesis. The algorithm used can be summarized in the following steps:

- Detection,
- Refinement,

- Descent,
- Landig detection.

In the detection phase, the environment is searched for a suitable landing place. 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 certainty of the positioning.

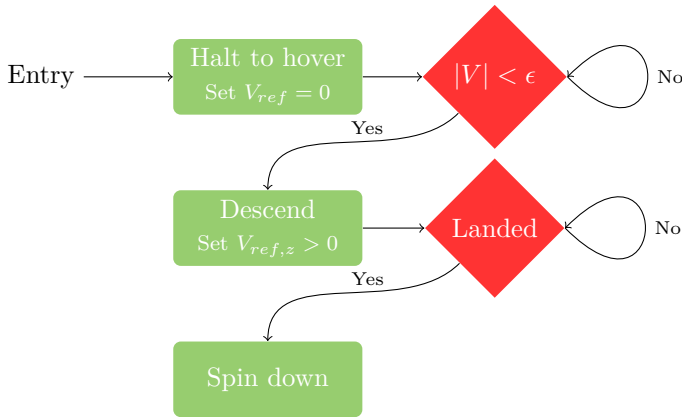


Figure 5.4. Landing scheme. The landing mode consists of three stages to perform and detect landing.

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

Landing Detection

Since we recognize that our estimated position - initialized at our starting point - is not necessarily consistent with the true Height Over Ground (HOG) at the landing site, it is necessary to choose a more robust method to detect the completion of the landing procedure than simply halting at zero height. The approach is to use the measurements to determine when movement has stopped; that is, when the quadrotor has reached ground. Detection theory, as discussed in e.g. [40, 32], provides several tools for detecting the non-linear event that the quadrotor can descend no further.

In the physical model presented in Chapter 2, two terms is of specific interest for the detection. The first - and the obvious - is the velocity in the gravity-aligned z-axis. 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 - oblivious to the forces imposed by the ground contact - will

explain the lack of movement by a drastic increase in the estimated wind acting in the z-direction. This observer state - the second of interest - is easily monitored and could e.g. be filtered using the Cumulative Sum (CUSUM) algorithm to increase detection confidence, or simply thresholded to detect landing.

Chapter 6

Results

This chapter contains results evaluating the theory of 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 Experiment Setup

The data used for the evaluation of the model and the filter in this Chapter was recorded on the LinkQuad quadrotor in the Witas 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, where not noted otherwise, 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 in Figure 6.1. The camera settings were tuned to minimize the disturbance from lightsources 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 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, the model was then used for simulated control and landing, which is used in Section 6.4 to validate the control. For the validation, simulated wind, random Gaussian system noise and random Gaussian measurement noise was injected into the system.

6.2 Modelling

The verification of a complex model is best done in small parts. It is however, with the model given in Chapter 2, 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 calibration and a second for validation.

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 modelling of Chapter 2 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, leaving the results to be merely directional. As depicted in Figure 6.2, the model does leave clearly trended residuals, not least in the X- and Y-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 residuals with zero-close means.

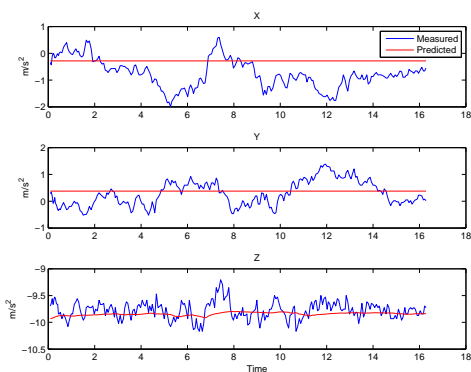


Figure 6.2. Measured and predicted accelerations in the NEDEF system.

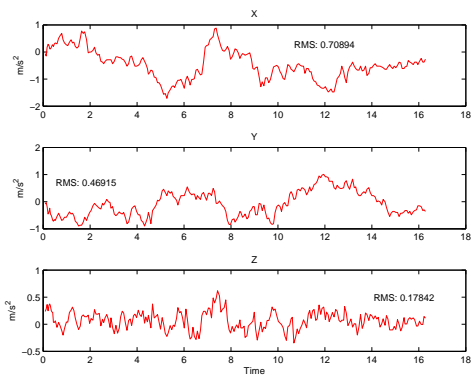


Figure 6.3. Residuals between measured and predicted accelerations.

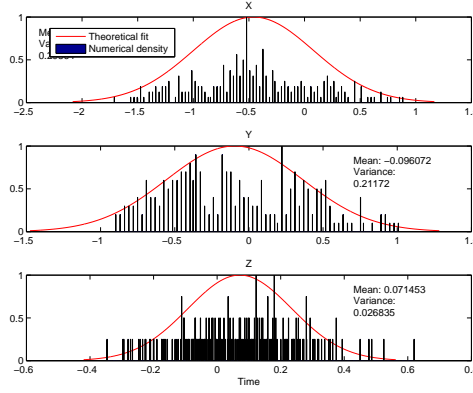


Figure 6.4. Accelerometer residuals fitted to a normal PDF. Both theoretical and numerical values have been normalized to a maximum height of one.

6.2.2 Gyroscopes

It is clear, from Figure 6.5, that the model well describes the angular velocity of the quadrotor. The residuals, described by Figures 6.6 and 6.7 display a behaviour which is adequately well described by a random, normally distributed, variable, which is expected from the standard Kalman filter framework.

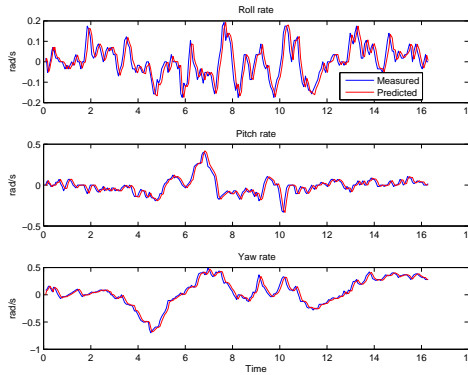


Figure 6.5. Measured and predicted angular rates, in the body-fixed coordinate frame.

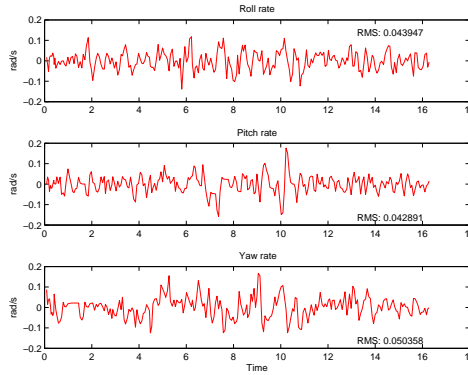


Figure 6.6. Residuals between measured and predicted angular rates.

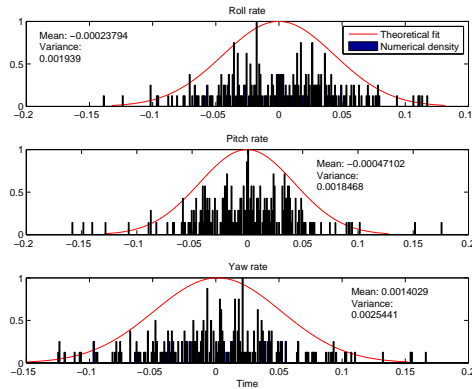


Figure 6.7. Gyroscope residuals fitted to a normal PDF. Both theoretical and numerical values have been normalized to a maximum height of one.

6.2.3 Pressure Sensor

The pressure sensor is, as clearly seen in Figure 6.8, associated with a great amount of noise. 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 is however likely to be improved by firther filter tuning.

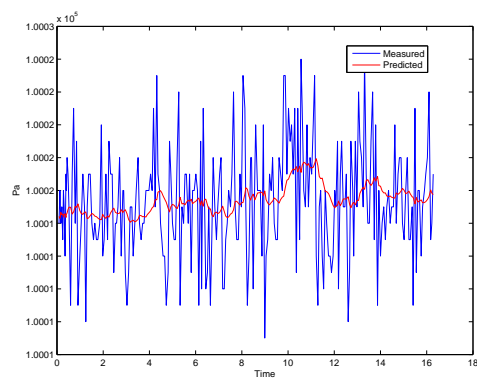


Figure 6.8. Measured and predicted pressure.

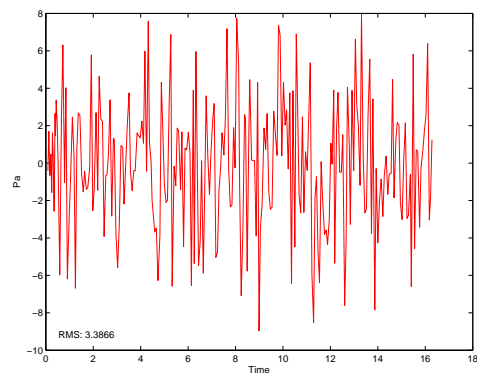


Figure 6.9. Residuals between measured and predicted pressure.

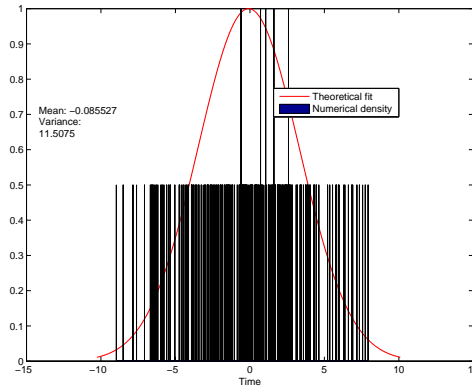


Figure 6.10. Pressure residuals fitted to a normal PDF. Both theoretical and numerical values have been normalized to a maximum height of one.

6.2.4 Camera

The camera tracking, displayed 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 does not only counter the drift in derived observer states, but also, not least through its accurate measurements of orientation angles exhibited in Figure 6.11. It should be noted that the results, especially in Figures 6.12-6.13 should be scaled approximately by a factor of three to correspond to metric quantities.

When the PTAM library is initialized, it tries to determine the ground plane. Thus we are able to verify, in Figure 6.14, the initialization process and transformation by confirming that the Z-axes are approximately parallel. The slight tilting observed in Figure 6.14 is caused by a misplacement of the ground plane in the initialization process. Knowing the pose at the initialization allows us to compensate for this in the transformation, making the positioning less sensitive for PTAM initialization errors. Exact positioning on the moment of initialization is still of importance, however.

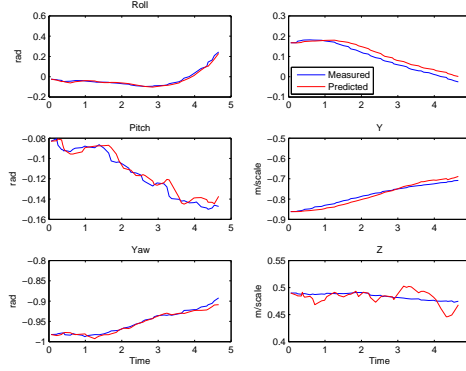


Figure 6.11. Measured and predicted angles and positions, in the PTAM coordinate frame. Measures should be multiplied by a scale of approximately 3 for metric comparison.

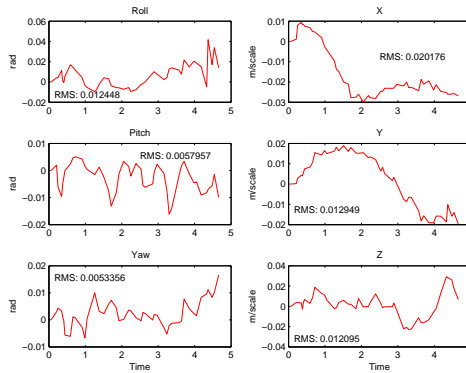


Figure 6.12. Residuals between measured and predicted angles and positions, in the PTAM coordinate frame. Measures should be multiplied by a scale of approximately 3 for metric comparison.

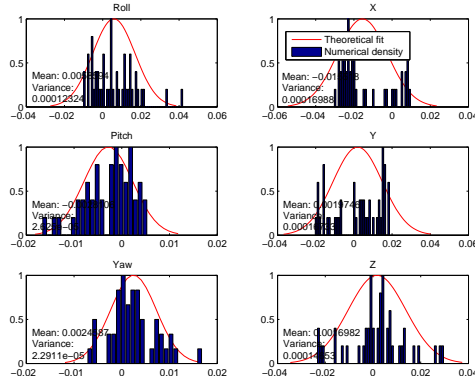


Figure 6.13. Residuals fitted to a normal PDF. Both theoretical and numerical values have been normalized to a maximum height of one. Measures should be multiplied by a scale of approximately 3 for metric comparison.

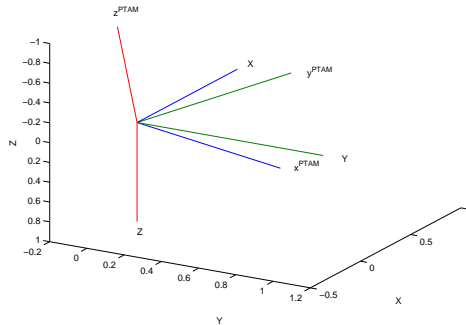


Figure 6.14. The PTAM coordinate system is somewhat askew from its theoretical orientation, with the z-axis parallel to the ground. This can also be seen in Figure 4.3, which is captured from the same dataset.

6.3 Filtering

6.3.1 Positioning

While the altitude positioning, shown in Figure 6.15, exhibit disturbances correlated with pressure sensor noise, the positioning generally exhibit very good performance. The position state is observed in the state-estimation more or less directly by the camera and the stability of the camera positioning is thus of course reflected here.

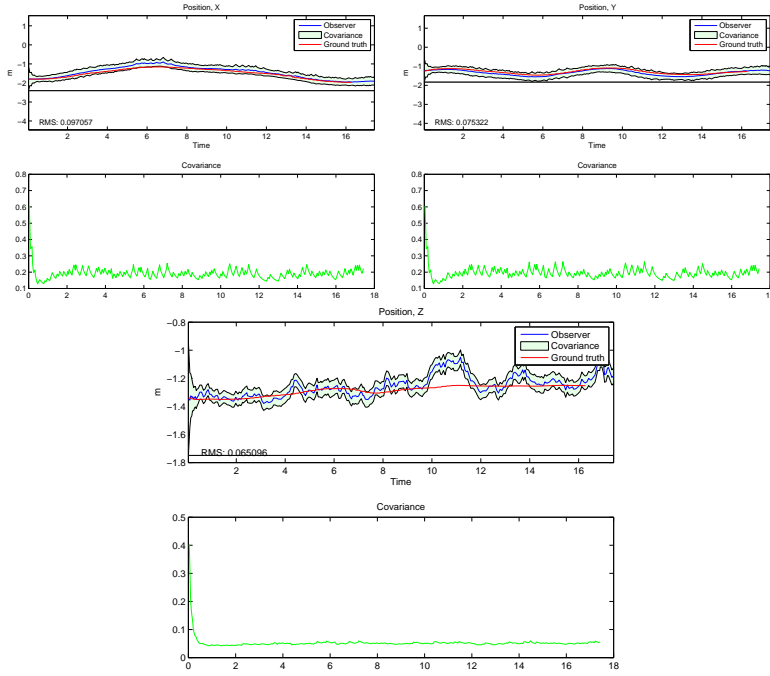


Figure 6.15. Positioning in the X- and Y-direction mostly well corresponds to ground truth, thanks to the camera positioning. The positioning in the Z-direction shows signs of the noise from the pressure sensor.

6.3.2 Velocities

The velocities, being closely coupled with the camera observed position, also exhibit good performance in Figure 6.16. There are shortcomings to the estimation's horizontal precision, although this could probably be significantly improved with further filter tuning.

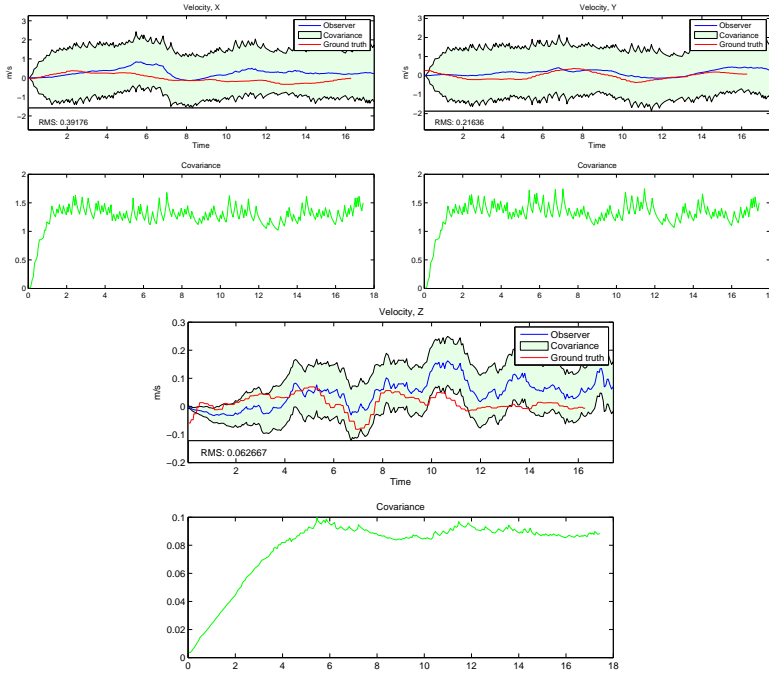


Figure 6.16. Velocity estimates are generally adequate, but with more tuning, the results are likely to improve.

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 depicted in Figure 6.19, they are.

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.

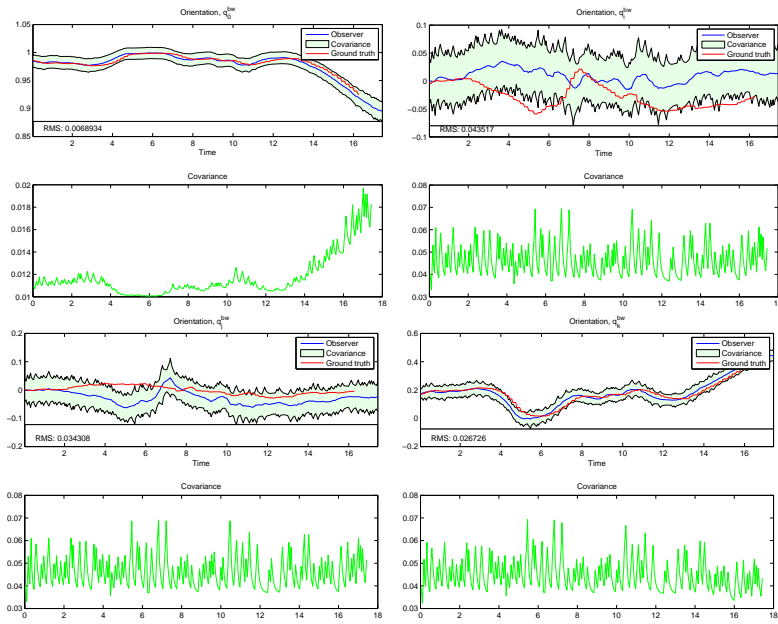


Figure 6.17. The orientation of the quadrotor was estimated with good accuracy.

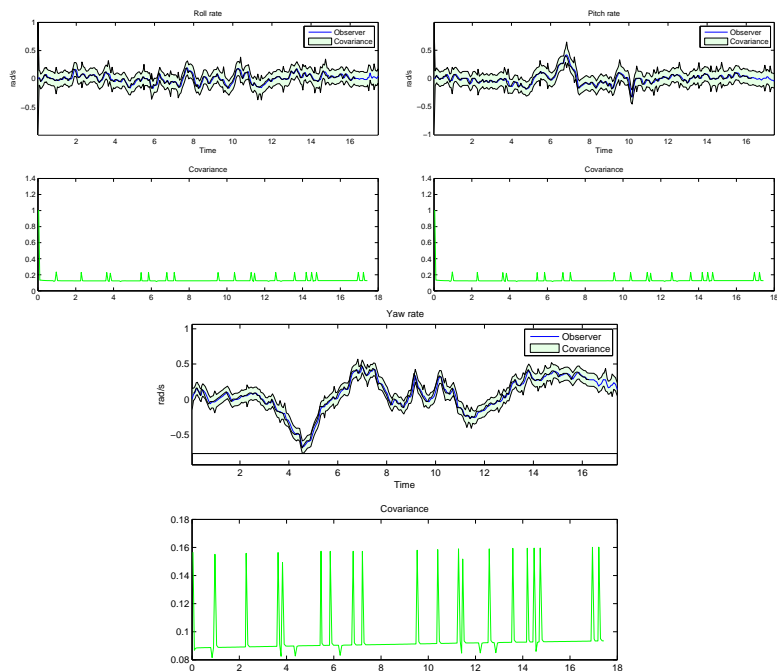


Figure 6.18. The predicted angular velocities corresponds very well to the gyro measurements.

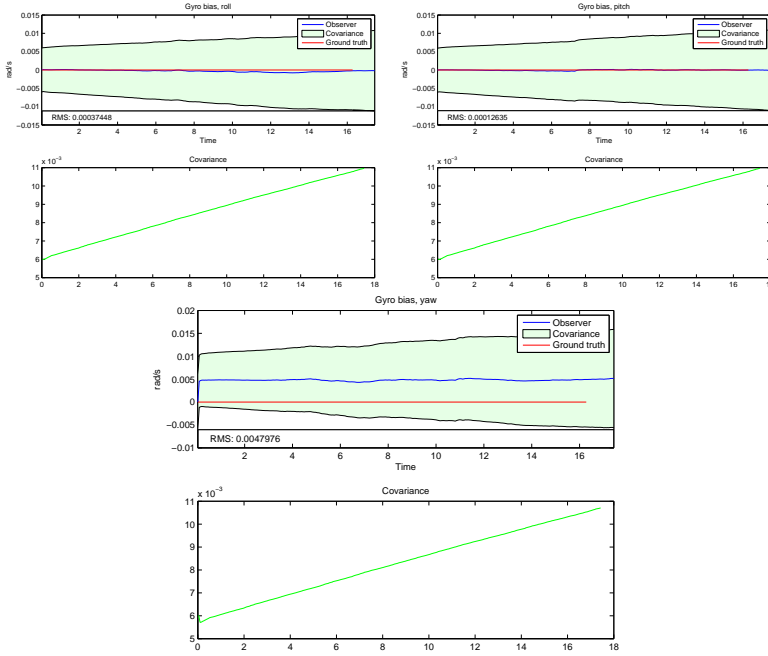


Figure 6.19. The gyroscopes' drift was removed prior to entering the filter, and does not change during the short recording of data.

6.3.4 Wind force

As the tests were performed inside, the filter was tuned to basically keep the wind constant. Thus, it is difficult to come to any conclusions regarding the wind impact on the model. Figure 6.20 show them to be correctly estimated to zero in the collected dataset, although in the case with simulated data with wind, shown in Figure 6.21, results are poor.

When a landing is simulated, Figure 6.22 exhibits an interesting property where a notable bump occurs at the time of the landing. While this is not the expected behaviour - the estimate should be negative and constant - it does show that the wind may be useful as a detector for landing.

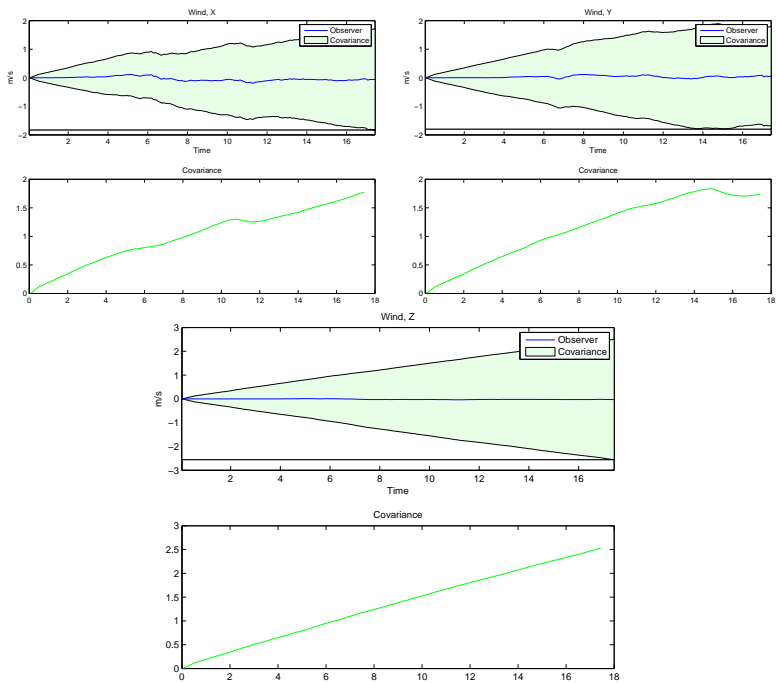


Figure 6.20. Wind estimates from recorded test-data.

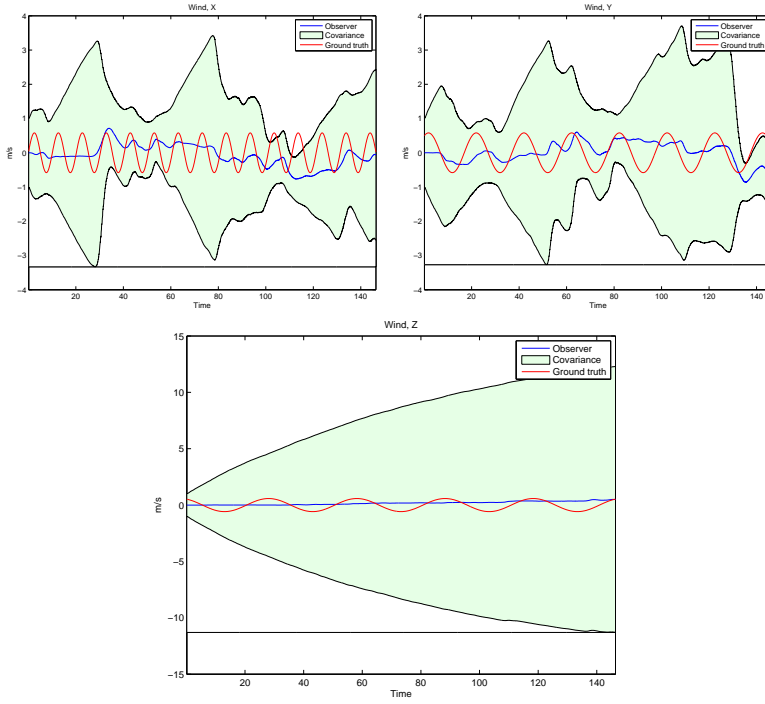


Figure 6.21. Wind from simulated test-flight.

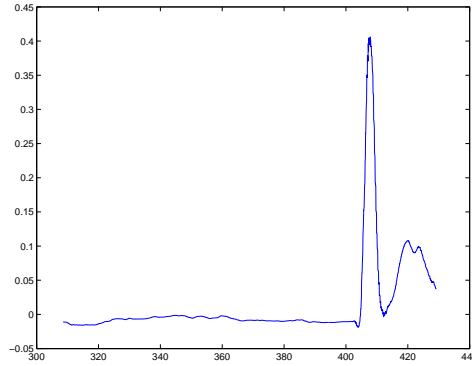


Figure 6.22. Even in simulation, the landing was detectable in the vertical wind estimate, albeit in this case not as expected.

6.3.5 Propeller Velocity

As the filter evaluation was performed without the use of the controller, the control signal is unavailable. Thus, Eq. 2.37 was used as motion model in the filter

validation, effectively leaving the estimation of the propeller velocities to the measurement update. It is evident, in Figure 6.23 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 plausability for correctness of Eq. 2.38a, which is otherwise hard to verify using the available data.

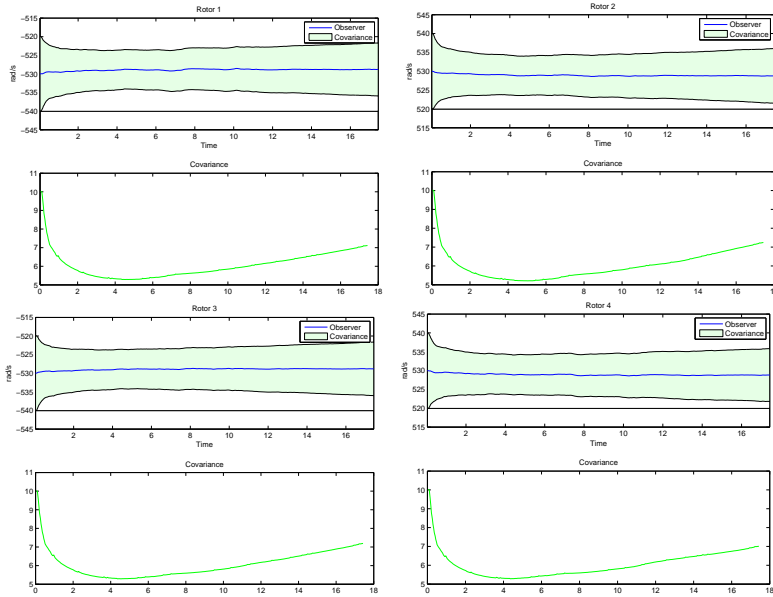


Figure 6.23. Propeller angular rate estimates could not be properly verified with the data available, although do exhibit a reasonable value range given the model parameter settings.

6.4 Control

The control algorithm was tested in simulation on the model presented of Chapter 2 and verified in Section 6.2. The control, very basically 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 is used.

In Figure 6.24, 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, when the velocity is below the threshold to begin descent. Landing is detected by the shortfall of velocity after just over 400 seconds.

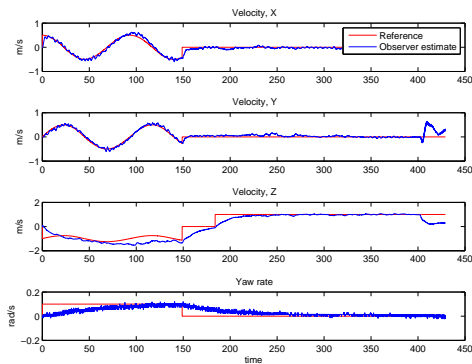


Figure 6.24. Control of the simulated flight was stable, although - being untuned - somewhat slow. The actuation of the yaw rate is very weak, which is evident in slow reaction of the yaw rate control.

Chapter 7

Discussion

In this chapter, the results of the thesis are discussed, analyzing the contents of Chapter 6 and providing a basis for later discussions on further work.

7.1 Modelling

It is arguable whether the advanced physical modelling presented in Chapter 2 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. 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 nature and extensiveness of the model does however open for e.g. future outdoor applications, handling wind and other disturbances. The state-space approach also allows for simple addition of further modelling and added states.

The model, as presented in this thesis, needs further in-detail verification and tuning. The flapping equations presented in Section 2.3.3 were in fact even disabled during the verification in Chapter 6 due to the difficulty of verification. In its current state, the model shows good promise but the performance of the model must also be weighed against the computational complexity.

The advanced model used in the state estimation does however 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, nearly every part of the model has been possible to re-use, be it for state-estimation, control, simulation of reality or sensor measurements. Having developed the more advanced model, and given time for verification and tuning, the evaluation of less complex models is much simplified in future work.

A model simplification that is believed to be rewarding is to replace the physical thrust equations using model identification to provide a simpler model. An-

other approach would be to replace the control interface for a more high-level control approach, using existing structure for controlling rotational velocities of the quadrotor. This would however remove the advantages of using the theory of optimal control.

7.2 Filtering

The results for the state estimation algorithms of this thesis were presented in Sections 6.2-6.3 of Chapter 6. As previously noted in Section 6.2.1, the parameters of the motion model were set to reasonably guessed values but otherwise untuned. As were the filter parameters. It should also be duly noted that the test data is recorded from a very limited test case. Limiting factors include:

- Video could only be recorded for 20 seconds (30 Hz mode) or 40 seconds (15 Hz mode),
- Only limited movement was possible in the experiment setup,
- Unperturbed ground truth was used for initialization.

7.2.1 EKF vs. UKF

In this thesis, two non-linear filter algorithms were studied. 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 an unstable model such as for a quadrotor, evaluating the model for 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 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, but with the inherent instability of the model, the UKF is still inherently 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 - not least the angle and angular velocity estimates. 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 modelling may increase the utility of the pressure sensor.

With the filter structure in place, what I believe to be the single most rewarding factor to increase the performance of the filtering is 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 is provided in the implementation.

Currently, the integration of the continuous model from Chapter 2 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, is expensive, 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.

7.3 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 the library to which PTAM was compared in its original paper [21]. The Scenelib library, using the filter approach described in Chapter 4.1, has the advantage to use metric measures. This could potentially have simplified 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 was unfortunately impossible with the resources available. 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, in the recorded test case, tracking was stable at the low capture rate of 15 Hz, which is promising as an implementation on the gumstix might be limited to those speeds considering the computational limits.

7.4 Controller

The advantages and disadvantages of different control techniques and its applications to quadrotor control have been previously discussed and evaluated in [3]. The conclusions are ambiguous, but the state-space control do have potential.

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 or may not always be available in flight-conditions.

The SDRE approach used in this thesis is also notably rare - if at all existing - in literature, and is an interesting topic for further research. Lacking of a full evaluation however, the results can only be directional but they do show promise. The control does of course suffer from the same lack of parameter tuning as the filtering that would be necessary for real in-flight control.

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.

Even though very little of the work was focused on tuning the control, the simulated control was unexpectedly good. The control handled moderate simulated winds and disturbances without much trouble, and followed velocity references adequately. Since the actuating of the yaw rate control is very weak, poor 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 5.1.4. Although the plot in Figure 6.22 is made from simulated data, 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. The received result was unexpected, and debugging should be performed prior to discarding the idea. Hence, no well-founded conclusions can be drawn as for the suitability of wind estimation as a means of landing detection. Further work is needed but it is believed that with proper filter tuning, the variable in question may be a contributing factor in a more advanced landing-detection system.

7.6 Real-time Performance

Simulations on the gumstix indicate that real-time performance could not be achieved 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. Furthermore, the system was evaluated using the reference implementation of BLAS¹, which could possibly be replaced by a tuned implementation such as ATLAS². Model simplifications may also be necessary to achieve real-time performance on the existing hardware.

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 little work remain before test-flights can be performed, the most important work beeing tuning of state-estimation and control.

With extensions to the state-machine logic, the system is prepared to perform advanced missions, including such planned by a future on-board artificially intelligent mission planner.

8.1 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.

8.1.1 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 modelling 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 inclusion of other types of optimal control, such as Model Predictive Control, MPC.

8.1.2 Mission Planning

With relatively little work, the state-machine logic could be extended with autonomous mission planning, using parallel state machines and e.g. PDDL planning tools. The graphical front-end could be replaced or extended with tools suitable for mission control.

8.1.3 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-work positioning using video feedback.

There is also a potential to improve the PTAM library. One could, for instance, consider closing the loop, using the observer's pose estimate and motion model to improve the quality of the camera tracking. The PTAM library was specifically developed for hand-held cameras without any other sensors available, yet the algorithm would benefit of such information. There are also some performance issues when the internal map of PTAM features grows. This could be improved by for instance using R-trees for keyframe indexing to sort out relevant keyframes to reproject into the video feed.

8.2 Conclusions

The achievements and theory presented in this thesis represent a set of higher level algorithms than is common in the SUAV field. This is highly intentional, as I believe it is the untested solutions that need to be tried in order to advance research in the area. While one may argue that a simpler solution is “good enough”, superiority is not achieved by mimicking results but by expanding the ideas of the cutting edge technology presented by a world of hobbyists, developers and scientists.

To be able to stand on the shoulders of previous work it is important that research is made publically available, both as theoretical reports but also their programmatic implementation. It is my firm belief that a scientific field will shine only with implementations available freely to form the base of something bigger. Therefore, the full implementation of this thesis is hereby released under the GNU General Public License¹.

¹GPLv3 or, by users choice, any later version.

Bibliography

- [1] Peter Benner. Accelerating Newton’s Method for Discrete-Time Algebraic Riccati Equations. In *In Proc. MTNS 98*, pages 569–572, 1998.
- [2] Michael Blösch, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Vision based MAV navigation in unknown and unstructured environments. In *ICRA*, pages 21–28, 2010.
- [3] 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.
- [4] Samir Bouabdallah and Roland Siegwart. Full control of a quadrotor. In *IROS’07*, pages 153–158, 2007.
- [5] 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.
- [6] Tayfun Çimen. State-Dependent Riccati Equation (SDRE) Control: A Survey. In *Proceedings of the 17th IFAC World Congress*, pages 3761–3775, 2008.
- [7] National Geophysical Data Center. The World Magnetic Model. <http://www.ngdc.noaa.gov/geomag/WMM/DoDWMM.shtml>, March 2012.
- [8] Andrew J. Davison. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *ICCV*, pages 1403–1410. IEEE Computer Society, 2003.
- [9] Patrick Doherty and Piotr Rudol. A UAV Search and Rescue Scenario with Human Body Detection and Geolocalization, 2007.
- [10] 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.

- [11] Evrin Bilge Erdem. ANALYSIS AND REAL-TIME IMPLEMENTATION OF STATE-DEPENDENT RICCATI EQUATION CONTROLLED SYSTEMS BY, 2001.
- [12] T. Glad and L. Ljung. *Reglerteori: flervariabla och olinjära metoder*. Studentlitteratur, 2003.
- [13] F. Gustafsson. *Statistical Sensor Fusion*. Utbildningshuset/Studentlitteratur, 2010.
- [14] Masayuki Hayashi et al. A Study of Camera Tracking Evaluation on TrakMark Data-Set. Technical report, University of Tsukuba, 2011.
- [15] 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.
- [16] Simon J. Julier and Idak Industries. The scaled unscented transformation. In *in Proc. IEEE Amer. Control Conf*, pages 4555–4559, 2002.
- [17] Simon J. Julier and Jeffrey K. Uhlmann. Unscented Filtering and Nonlinear Estimation. *Proceedings of the IEEE*, pages 401–422, 2004.
- [18] Simon J | Uhlmann Jeffrey K | Durrant-Whyte Hugh F Julier. 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] Georg Klein and David Murray. Parallel Tracking and Mapping on a Camera Phone. In *Proc. Eighth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'09)*, Orlando, October 2009.
- [23] J.B. Kuipers. *Quaternions and rotation sequences: a primer with applications to orbits, aerospace, and virtual reality*. Princeton paperbacks. Princeton University Press, 2002.
- [24] J.G. Leishman. *Principles of helicopter aerodynamics*. Cambridge aerospace series. Cambridge University Press, 2002.

- [25] Manolis I.A. Lourakis, editor. *Bundle adjustment gone public*, 10 2011.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] Thanh Nguyen, Christian Sandor, and Jun Park. PTAMM-Plus: Refactoring and Extending PTAMM. *Camera*, pages 84–88, 2010.
- [31] Carl Nordling and Jonny Österman. *Physics Handbook*. Studentlitteratur, 2006.
- [32] Mattias Nyberg and Erik Frisk. *Model Based Diagnosis of Technical Processes*. LiU-tryck, 2011.
- [33] 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.
- [34] Paul Pounds, Robert Mahony, and Peter Corke. Modelling and Control of a Quad-Rotor Robot.
- [35] R.W. Prouty. *Helicopter performance, stability, and control*. Krieger Pub., 1995.
- [36] A. Rantzer and M. Johansson. Piecewise Linear Quadratic Optimal Control, 1999.
- [37] 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.
- [38] 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.
- [39] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Real-time monocular SLAM: Why filter? In *ICRA*, pages 2657–2664, 2010.

- [40] David Törnqvist. *Estimation and Detection with Applications to Navigation*. PhD thesis, Linköping University, 2008.
- [41] 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.
- [42] Rudolph van der Merwe, Arnaud Doucet, Nando de Freitas, and Eric A. Wan. The Unscented Particle Filter. In *NIPS'00*, pages 584–590, 2000.
- [43] 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.
- [44] Wikipedia. Miniature UAV — Wikipedia, The Free Encyclopedia, 2012. [Online; accessed 14-May-2012].
- [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 than two-hundred lines of code.

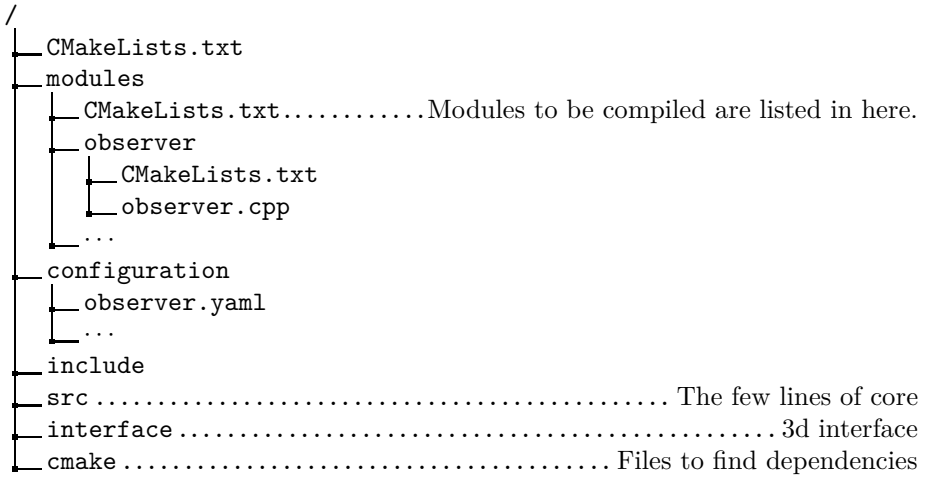
The main ideas of *CRAP* are borrowed from the *Robot Operating System*, *ROS*¹, while implementing these in a more efficient and uniform manner. By constraining the intermodular messaging to C++² and compartmentalizing the modules in separate threads as opposed to processes, *CRAP* significantly reduces 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/>

²ROS allows messaging between Python and C++ modules



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 is 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 a separate *messaging thread*, 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 runtime, 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.

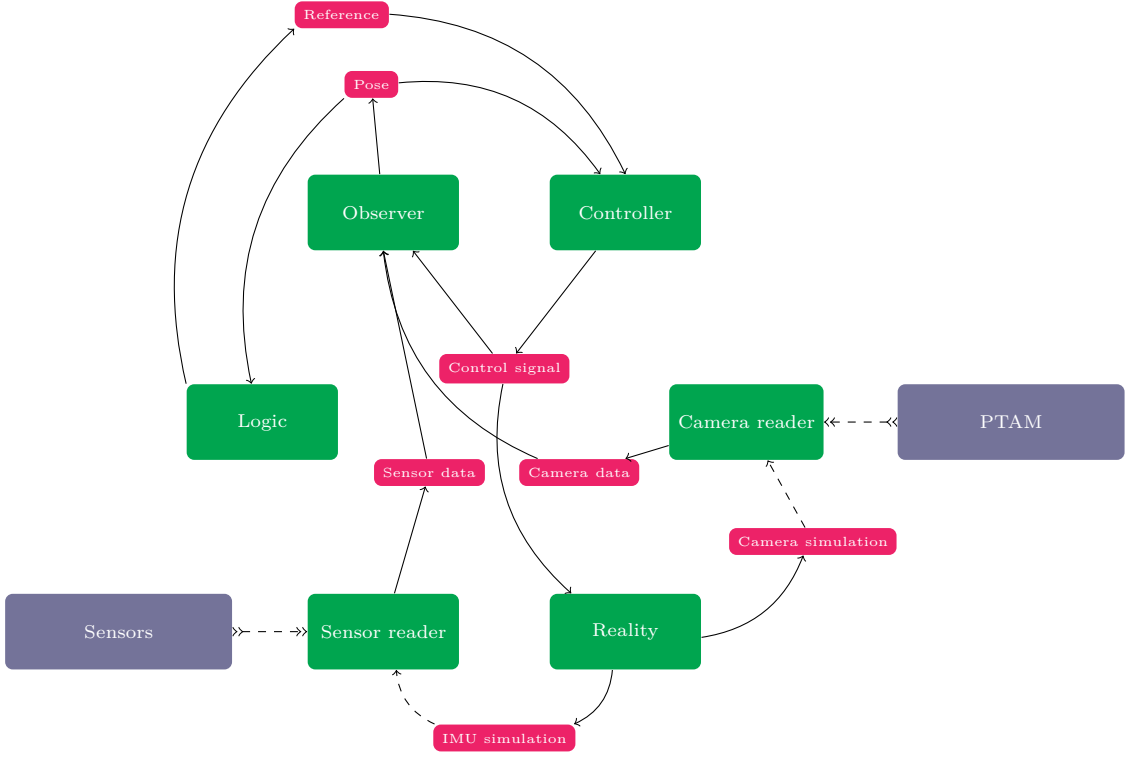


Figure A.1. CRAP schematic. The Figure describes the relationship of modules (green) and their internal communication through messages (red). External modules, connected through a serial interface, are shown in blue. Dashed lines are optional.

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 re-uses 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 2. 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.

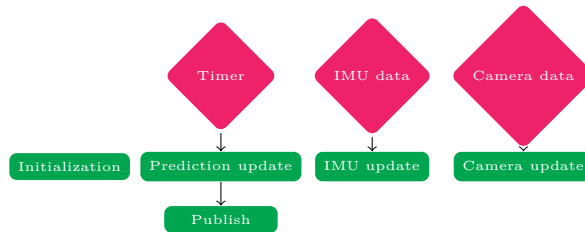


Figure A.2. The observer reacts to three events; A timer executes the prediction update, and incoming messages invoke the measurement updates. The state is published to other modules at a fixed rate immediately after the prediction update.

A.3.2 Controller

The controller implements the non-linear control described in Chapter 3. 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.

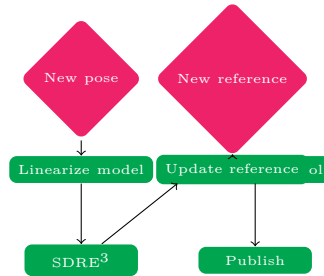


Figure A.3. The controller output is changed only when the a new pose estimate arrives from the observer.

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 structure is as follows:

```

/
├── configuration
│   ├── modes
│   │   ├── hover.yaml .2 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.

³Solve 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 interfaces the graphical front-end of 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 the *CRAP* engine and forwarded by the **freeflight** state-machine mode to the controller as control reference. The interface implements 3D control through the open-source spacenav⁴ drivers for 3DConnexion's 3D-mice.

⁴<http://spacenav.sourceforge.net/>

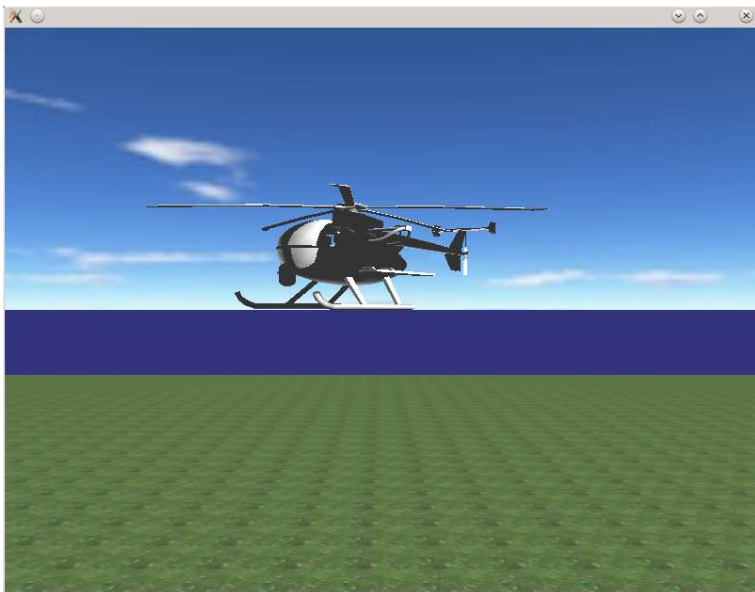


Figure A.4. The *CRAP* framework features a simple graphical viewer to visualize the observer's pose estimates.

Appendix B

GNU General Public License

Copyright © 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Abstract

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal

permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To “convey” a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays “Appropriate Legal Notices” to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

- (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.
- (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to “keep intact all notices”.
- (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
- (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an “aggregate” if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation’s users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

- (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
- (b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this

conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

- (c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
- (d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
- (e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product

is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

- (a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or
- (b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or
- (c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

- (d) Limiting the use for publicity purposes of names of licensors or authors of the material; or
- (e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or
- (f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell,

offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse

you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE

ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program’s name and a brief idea of what it does.>

Copyright (C) <textyear> <name of author>

This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program> Copyright (C) <year> <name of author>
```

```
This program comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.  
This is free software, and you are welcome to redistribute it  
under certain conditions; type 'show c' for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see <http://www.gnu.org/licenses/>.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read <http://www.gnu.org/philosophy/>