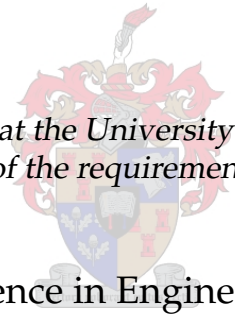
UNIVERSITEIT·STELLENBOSCH·UNIVERSITY
jou kennisvennoot • your knowledge partner

# 3D Tracking between Satellites using Monocular Computer Vision

by

Daniël François Malan



*Thesis presented at the University of Stellenbosch in partial fulfilment of the requirements for the degree of*

Master of Science in Engineering Sciences

Department of Electrical & Electronic Engineering
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Supervisors:

| | |
|---|---|
| Prof. W.H. Steyn | Prof. B.M. Herbst |
| Department of Electrical | Department of |
| & Electronic Engineering | Applied Mathematics |

April 2005

# Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature: .....
D.F. Malan

Date: .....

# Abstract


**3D Tracking between Satellites**
**using Monocular Computer Vision**

D.F. Malan

*Department of Electrical & Electronic Engineering*
*University of Stellenbosch*
*Private Bag X1, 7602 Matieland, South Africa*

Thesis: M.Sc. Eng.Sci. (Electronic)

April 2005

Visually estimating three-dimensional position, orientation and motion, between an observer and a target, is an important problem in computer vision. Solutions which compute three-dimensional movement from two-dimensional intensity images, usually rely on stereoscopic vision. Some research has also been done in systems utilising a single (monocular) camera.

This thesis investigates methods for estimating position and pose from monocular image sequences. The intended future application is of visual tracking between satellites flying in close formation. The ideas explored in this thesis build on methods developed for use in camera calibration, and structure from motion (SfM). All these methods rely heavily on the use of different variations of the Kalman Filter.

After describing the problem from a mathematical perspective we develop different approaches to solving the estimation problem. The different approaches are successfully tested on simulated as well as real-world image sequences, and their performance analysed.

# Uittreksel


**3D Afskatting tussen Satelliete**
**met behulp van ’n enkel Digitale Kamera**

*(”3D tracking between satellites*
*using Monocular Computer Vision”)*

D.F. Malan

*Departement Elektriese & Elektroniese Ingenieurswese*
*Universiteit van Stellenbosch*
*Privaatsak X1, 7602 Matieland, Suid Afrika*

Tesis: M.Sc. Ing.Wet. (Elektronies)

April 2005

Visuele afskatting van driedimensionele posisie, oriëntasie en beweging, tussen ’n waarnemer en ’n teiken, is ’n belangrike probleem in verksie navorsingsvelde. Metodes wat driedimensionele beweging vanuit tweedimensionele intensiteitsbeelde bereken, maak normaalweg op stereoskopiese visie staat. Daar is egter ook al navorsing gedoen oor oplossings wat ’n enkele kamera se beelde gebruik.

Dié tesis ondersoek metodes om relatiewe posisie en oriëntasie vanaf ’n enkele kamera se beelde te bepaal. Die beplande toepassing is vir waarneming tussen satelliete, tydens nabye formasievlug. Die idees wat in hierdie tesis ontwikkel word bou voort op konsepte wat gebruik word in kamerakalibrasie, en struktuur vanaf beweging ("SfM"). Verskillende weergawes van die Kalman Filter speel ’n belangrike rol in die algoritmes wat ontwikkel word.

Nadat die probleem vanuit ’n wiskundige hoek beskryf is, ontwikkel ons verskillende benaderinge tot die oplos van die afskattingsprobleem. Die verskillende metodes word suksesvol op gesimuleerde asook egte beeldreekse getoets , en die metodes se werkverrigting word geanaliseer.

# Acknowledgements

I am grateful to the University of Stellenbosch and specifically the Electronic Systems Laboratory (ESL) for the infrastructure and support which made the completion of this thesis possible. Furthermore, I would like to thank my study leaders, Prof. Herman Steyn and Prof. Ben Herbst, for the encouraging feedback and friendly advice they gave me during the time I was their student.

I especially thank my good friends and fellow students at Stellenbosch. You know who you are. Thank you for all the time spent together, encouragement in difficult times, all the fun things we did, and for keeping me more or less sane.

The financial assistance of the National Research Foundation (NRF) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to the NRF.

# Dedication

Aan my pa, Danie

# Contents

# List of Figures

# List of Tables

# Nomenclature

**Acronyms:**

| | |
|---|---|
| CCD | Charge Coupled Device (A grid of light-sensitive pixels) |
| CCS | Camera Coordinate System |
| CMOS | Complementary Metal-Oxide Semiconductor (an alternative to a CCD) |
| DCM | Direction Cosine Matrix |
| EKF | Extended Kalman Filter |
| HOT | Higher Order Terms |
| KF | Kalman Filter (in general) |
| LED | Light Emitting Diode |
| LKF | Linear Kalman Filter |
| OCS | Object Coordinate System |
| SfM | Structure from Motion |
| UKF | Unscented Kalman Filter |

**Constants:**

| | |
|---|---|
| $\pi =$ | 3,141 59 ... |
| $\mathrm{e} =$ | 2,718 28 ... |
| $\mathbf{0}_{m \times n}$ | $m \times n$ matrix of zeros |

$\boldsymbol{I}_{m\times n}$ $m \times n$ matrix of zeros except for main diagonal which contains ones. Identity matrix if $m = n$

**Notational convention:** [1]

$a$ scalar value $a$

$A$ scalar value $A$

$\boldsymbol{a}$ column vector $\boldsymbol{a}$

$\boldsymbol{A}$ matrix $\boldsymbol{A}$

$\boldsymbol{a}^T$ transpose of the vector $\boldsymbol{a}$

$\boldsymbol{A}^T$ transpose of the matrix $\boldsymbol{A}$

$a_i$ $i$th vector component of $\boldsymbol{a}$

$\boldsymbol{a}_i$ $i$th row of $\boldsymbol{A}$

$\boldsymbol{a}^{(i)}$ $i$th element of a set of vectors (used exclusively for the UKF)

$A_{ij}$ $(i, j)$-th element of $\boldsymbol{A}$

$\boldsymbol{a}_k$ value of $\boldsymbol{a}$ at time step $k$

$\bar{\boldsymbol{a}}_k$ predicted value of $\boldsymbol{a}$ for time step $k$

$\hat{\boldsymbol{a}}_k$ estimated value of $\boldsymbol{a}$ at time step $k$

$\Delta a$ increment or deviation of $a$ (assumed to be small)

$\dot{a}$ first time derivative of $a$

$\ddot{a}$ second time derivative of $a$

[1] Some deviations from these rules are occasionally permitted (where indicated), so as to be consistent with the general literature on those subjects.

# Chapter 1

# Introduction

## 1.1 Background

Although satellite constellations are in widespread use, the application of satellites flying in close formation (separated by less than 200 m) is relatively new. The University of Stellenbosch's "Electronic Systems Laboratory" is currently looking at using close formation flight as part of future micro-satellite missions. The use of close formation flight between imaging satellites can be used for stereo earth imaging, and is also useful for the external observation of a "mother" satellite by a smaller "slave" satellite.

Accurate information concerning relative position and motion is required to maintain formation flight between satellites, and might be useful for applications such as satellite docking. Information concerning relative orientation might be important when doing inspection, for alignment of antennae, and also for docking. Together, these requirements stipulate the need for accurate position and pose estimation. Optical computer vision could represent an affordable and practical way to determine position and orientation between an observer and a target satellite.

We propose that the position and pose estimates be computed locally by one of the satellites (the observation satellite). This approach is desirable, since it circumvents any transmission delays, and is not dependent on observability from the ground. The onboard processing power of a micro-satellite should be sufficient for this task.

In their daily lives, people naturally understand and operate in a three-dimensional world. This is actually curious, considering that we only directly sense 2D projections of the world around us. The seemingly effortless task of inferring 3D from 2D images is the result of not only stereo vision, but also of complex processing by our neural system.

Research focusing on motion estimation from monocular image sequences over the past fifteen years often addressed the problem of structure from motion (SfM). SfM attempts to estimate both the 3D motion as well as the 3D structure of a target by analysing its 2D image (video)

projection. A problem with SfM is that it generates estimates that are typically scale invariant. The shape of the target can be determined, but the unknown size of the target results in an unknown distance from the observer. This problem is unique to monocular estimation where triangulation is impossible. In our application we are explicitly interested in the absolute 3D position, and pose, of the target. Furthermore, the dimensions and structure of a satellite target will be known to the observer. By incorporating the knowledge of the target being viewed, we can attempt to apply the techniques of SfM to estimate the absolute translational and rotational state of the target, using solely a monocular image sequence.

Two fundamentally distinct approaches exist for obtaining 2D measurement data for motion estimation. Optical flow based methods represent 2D motion in the image plane as a continuous velocity field. This approach has proved useful in applications where a complex scene needs to be segmented into moving and stationary components. Feature based methods, on the other hand, rely on the recognition of a set of corresponding feature points as they occur in consecutive image frames.

A seminal paper by by Broida, Chandrashekhar and Chellappa [1] focused on using corresponding feature points between successive images to estimate both 3D motion and structure. Measurements of 3D accuracy were not obtained due to scale invariance of the structure estimator, and lack of "ground truth" measurements for the real-world motion sequences. Blostein and Chann [2] used the idea of [1], in combination with methods of optical flow, to achieve SfM estimation for a simulated image sequence. The incorporation of optical flow into feature-based algorithms does not seem to be used widely, even though [2] shows that noticeable gains in accuracy can be achieved. An attempt at absolute position and motion estimation by using line features was investigated by Goddard [3]. In this case absolute 3D accuracy was measured, but only for uniform motion with a target placed close to the observer. Attempts at SfM estimation by using automatically[1] extracted features points was investigated by Venter [4], and developed further by Rautenbach [5]. All of these investigations used various nonlinear extensions of the Kalman Filter as a central part of the estimation algorithms.

## 1.2 Outline of proposed solution

We propose the use of a single digital camera (monocular vision) to track visual markers (point features) on the target. The digital camera's detector might, for example, consist of a 1 megapixel monochrome CMOS or CCD imager. The marker are affixed at known positions on the target, which should pose no practical problems since the target (satellite) is rigid and has a known structure.

Each observation (image frame) will provide us with a certain number of visible feature points. The first step in the tracking algorithm would be to identify the feature points in the 2D im-

[1] using a two-dimensional KLT (Kanade-Lucas-Tomasi) feature tracker

age frame. The second step involves identifying each feature and matching it with its known position on the target. The third step is to incorporate these feature matches to an estimate of position and pose. We can use an algorithm to combine all features of a given image frame to a single "measurement" of position and pose, or we can treat each point feature as a separate measurement – thereby avoiding the use of algorithms which depend on the number of visible features. Both approaches will be explored.

The nature of the observations imply that each individual 2D feature point, and therefore each observation of translation and rotation, could be corrupted by a significant amount of noise - especially when looking at an object which is far away. We assume that the satellite will obey well-known kinematic equations in a predictable inertial or orbital environment, and will therefore be describable by a relatively simple and dependable dynamic system model.

The above mentioned scenario is an ideal application for Kalman filtering – a recursive estimator which optimally combines predicted and observed data, by making certain assumptions about the noise distribution and system model. As in previous research on this topic we will use the Kalman filter as an integral component of our tracking algorithms.

## 1.3 Other possible solutions

There are plausible alternatives to estimating relative 3D position and pose between satellites.

Optical tracking based on stereo vision (as is human vision) has the advantage of enabling 3D triangulation – a much more simple way of determining 3D position than monocular tracking. Triangulation is especially well suited to identifying the structure of an unknown object. As mentioned earlier this approach is not ideally suited for our application due to the distances involved, possible space and weight limitations and the fact that we will be tracking a known object.

If each satellite measures its own pose and spatial position, the relative pose and position can be computed by comparing the two measurements. Most satellites measure their own orientation by various means (magnetometers, gyroscopes, star cameras, etc.), whereas GPS can be used to determine 3D translation. To let this scheme work, the satellites must be able to communicate with each other directly or via a base station. This is not always viable.

Range between two satellites can be measured directly with a radar or laser range finder. This, however, gives no attitude information.

# Chapter 2

# Mathematical Models: Describing the real world

## 2.1 Imaging process: From 3D object to 2D image

We choose to model the imaging process as a perspective projection. Most cameras are adequately described by this model, and it is well suited for our use. An ideal camera which performs a perspective projection is called a "perspective camera" or "pinhole camera". For a lens of good quality with a narrow field of view (less than 50°) this model is quite sufficient.

A perspective camera lets all light entering the lens pass through a single point – the optical centre. Schematics of the process are shown in figures 2.1 and 2.2. This implies that the image being viewed is always in focus, and also the absence of lens distortion. A real lens does have some distortion, but this can be corrected for, and will be shown in section §2.1.2.

We use the camera's optical centre as the origin for the camera-based coordinate system (CCS). This simplifies the perspective equations.[1] The most straightforward way to examine the 3D to 2D imaging process is to look at the way in which a single point is projected. A point is the smallest and most basic unit in our geometry. In some computer vision applications it might be easier to detect visible lines rather than points, but it is useful to keep in mind that a line might be thought of as a set of points. The choice of coordinate systems is shown in figure 2.2.

The perspective projection which relate a 3D object point $[X, Y, Z]^T$ to its 2D equivalent $[x, y]^T$ can be written as follows:

$$x = -f\frac{X}{Z} \qquad\qquad y = -f\frac{Y}{Z} \tag{2.1.1}$$

, where $f$ is the camera's focal length. When dealing with digital images we measure image

[1] Venter [4] places the origin on the focal plane, which makes the equations more laborious.

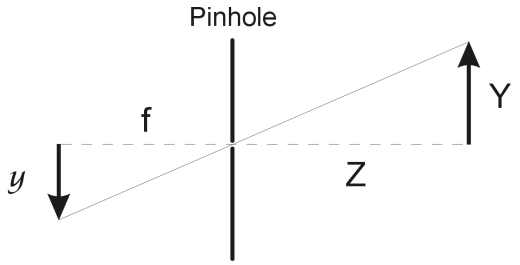



**Figure 2.1:** The Perspective (pinhole) Projection

coordinates in units of pixels. If we ignore the units, we can define the *effective focal length* as the value of $f$ which will satisfy equation (2.1.1) when the image point $[x\ y]^T$ and 3D coordinate $[X\ Y\ Z]^T$ are expressed in their respective units (typically pixels and meters). Note that this projection "mirrors" the image in the $x$ and $y$ axes. Since digital cameras automatically negate the mirroring of the image, the effective focal length will be a negative number. We can repre-

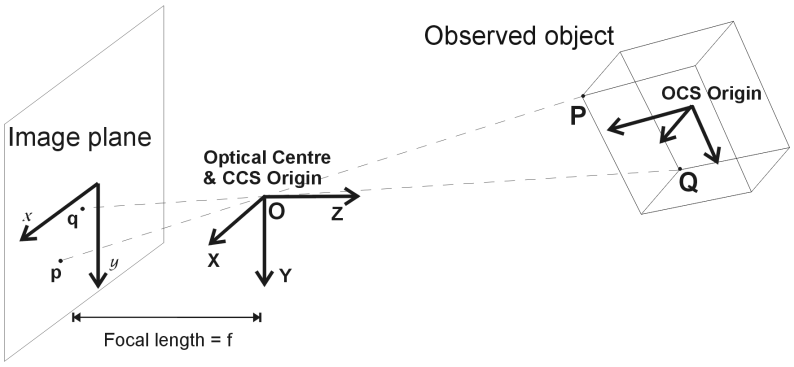



**Figure 2.2:** Image formation and axes of our camera model

sent this projection in linear algebra with the help of homogeneous coordinates (appendix C). As a matrix equation the projection can then be written as:

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} \equiv \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

where $s$ is a scaling factor, which cancels when the image is represented as a 2D Euclidean vector.

The above equations assume that the 3D position of the visible point is known in the CCS.

We can represent the position and pose of the OCS relative to the CCS as a rotation followed by a translation. The rotation is around the origin of the OCS (to align the axes). The translation is

relative to the CCS (to align the origins).

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} \equiv \begin{bmatrix} -f & 0 & 0 \\ 0 & -f & 0 \\ 0 & 0 & 1 \end{bmatrix} \left( \boldsymbol{R} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \boldsymbol{t} \right), \tag{2.1.2}$$

where $\boldsymbol{R}$ is the $3 \times 3$ rotation DCM and $\boldsymbol{t}$ is the $3 \times 1$ translation vector.

Although equation (2.1.2) is expressed in terms of a DCM rotation, we will eventually be using the quaternion representation (section §D.4) when developing our 2D to 3D reconstruction algorithms in chapter 5. We rewrite equation (2.1.2) by expressing $\boldsymbol{R}$ in terms of quaternion elements, as described in section §D.6. Multiplying out and converting from homogeneous to Euclidean coordinates then yields the full state vector to image equation (5.6.1).

### 2.1.1 Alternatives to using point projections

In some applications it might be easier to detect visible lines rather than single points. This is especially true for cases in which marker points cannot be placed on the target, and general feature extraction methods therefore have to be used. Lines are however more complex to represent mathematically. A method using line representations have been developed for 3D tracking by Goddard [3]. The advantages of using lines as opposed to points are not pronounced, and according to [3], seem to be limited to cases in which the object is close to the centre of the image.

### 2.1.2 Correcting for Lens Distortion

Lens distortion is practically always present for a system of optical lenses, albeit not always significant. Distortion for most lenses can be adequately modeled by considering the following mapping from undistorted image $[x\ y]^T$ to distorted image $[\tilde{x}\ \tilde{y}]^T$:

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} + L(r) \begin{bmatrix} x - x_c \\ y - y_c \end{bmatrix},$$

with

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2}, \qquad L(r) = 1 + \kappa_1 r + \kappa_2 r^2.$$

The function $L(r)$ can be seen as a truncated Taylor series, which approximates distortion depending only on the radius from the centre of distortion, $(x_c, y_c)$.

Correcting for radial distortion can be done by applying the inverse multiplication

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_c \\ y_c \end{bmatrix} + \frac{1}{L(r)} \begin{bmatrix} \tilde{x} - x_c \\ \tilde{y} - y_c \end{bmatrix},$$

with

$$r = \sqrt{(x - x_c)^2 + (y - y_c)^2}, \qquad L(r) = 1 + \kappa_1 r + \kappa_2 r^2.$$

Note that $L(r)$ is still a function of $[x, y]$, which is the answer we seek! We can overcome this slight catch-22 by using $[x, y] \approx [\tilde{x}, \tilde{y}]$ to approximate $L(r)$, and iterating when the more accurate value for $[x, y]$ is computed. We found that this method converges very quickly (only 3 or 4 iterations typically needed).

Barrel distortion (see figure 2.3) is usually most visible at wide angles and pincushion at telephoto (although less common).

A simple and efficient way of computing and correcting for this type of lens distortion is described in [6]. The proposed method relies on identifying a distorted line which should appear straight (since the real-world edge is straight). The distortion parameters are then optimized in a way that minimizes some cost function. We chose the norm of the residual of linear regression (through the line points) as a cost function.

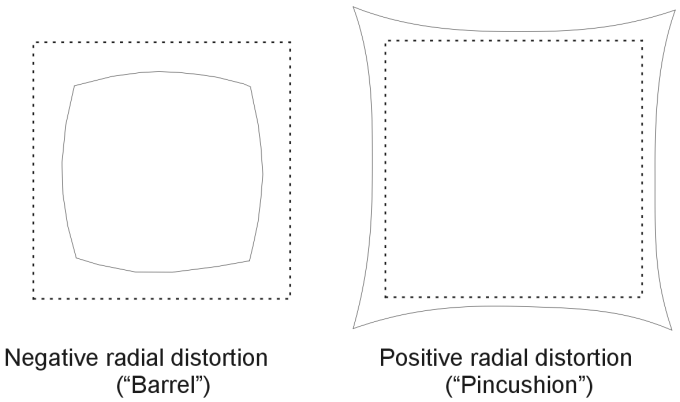



**Figure 2.3:** The effects of radial distortion on an image.

The distorted image of a straight line is obtained by photographing a scene which contains a straight line of high contrast (figure 2.4). The line is identified by computing the spatial derivative (using the Sobel operator) of the image, and thresholding the result. This step is often called "edge detection". The result of edge detection is usually a rather thick line (depending on the sharpness of the colour transition). The thresholded edge can be thinned to a line of single-pixel width by using a morphological thinning technique as described in [7].

A robust and simple alternative is to trace the distorted line by hand (as was used with figure 2.4).

We took sample images with a commercially available compact digital camera (Canon S45) to measure distortion. For this camera we found no measurable distortion at telephoto (actual focal length 21.3 mm), and very little barrel distortion at wide angle (actual focal length 7.1 mm). The maximum deflection of a straight line relative to the length of the line joining its end points (photographed at the periphery of the image) is in the order of 1%. This value is

typical for cosumer-level digital cameras according available camera reviews [8]. We found the distortion for our camera to have the following parameters at wide angle: $[x_c\ y_c\ k_1\ k_2] = [\ 0\quad 0\quad -0.0362\quad 0\ ]$. Note that we did not work in pixel coordinates, but rather rescaled the image to fit tightly in a $[-1,1] \times [-1,1]$ area (preserving aspect ratio). This was done so that the distortion parameters are not dependent upon the image resolution.

The image of figure 2.4 was photographed with a Canon S45 digital camera, and figure 2.5 shows how the barrel distortion was corrected for. Note how the curvature of the lines near the image's edges are most affected. It is also interesting to see how the boundaries of the image are warped by the distortion correction. Note that these warped boundaries can create an optical illusion which still causes the corrected lines to appear bent. (Using a ruler verifies that they are now indeed straight.)

**Figure 2.4:** Original photo with visible barrel distortion

**Figure 2.5:** Distortion-corrected image (lines now straight)

## 2.2 Kinematic equations of rigid 3D motion

We will now describe the equations which model the motion of a rigid 3D object (i.e. the satellite or object being tracked). Wertz [9] provides the continuous-time equations for describing inertial movement and rotation of a rigid body. These equations are later approximated in discrete-time for the purpose of discrete Kalman filtering.

We will refer to the target object as "the satellite" although the model obviously holds for any rigid target. The following definitions will be used:

| | | |
|---|---|---|
| $M$ | : | Mass of the satellite (kg) |
| $\boldsymbol{I}$ | : | $3 \times 3$ Inertia matrix of the satellite $(\mathrm{kg\,m^2})$ |
| $\boldsymbol{F}$ | : | $3 \times 1$ External linear force vector exerted on the satellite $(\mathrm{N})$ |
| $\boldsymbol{\Gamma}$ | : | $3 \times 1$ External torque vector exerted on the satellite $(\mathrm{N\,m})$ |
| $\boldsymbol{s}$ | : | $3 \times 1$ Translation vector of the satellite $(\mathrm{m})$ |
| $\boldsymbol{q}$ | : | $4 \times 1$ Rotation quaternion of the satellite (no unit) |
| $\boldsymbol{v}$ | : | $3 \times 1$ Linear velocity vector of the satellite $(\mathrm{m\,s^{-1}})$ |
| $\omega_0$ | : | Angular rate of the satellite in its circular orbit $(\mathrm{s^{-1}})$ |
| $\boldsymbol{\omega}$ | : | $3 \times 1$ Angular velocity vector of the satellite along the OCS's three axes.$(s^{-1})$ |
| $\Delta t$ | : | Time interval used for computation $(s)$ |

Note that the rotation, translation, linear velocity, and linear force, relate to the CCS (fixed to the observing camera). We assume that the observer's motion is known and/or can be compensated for. We are only interested in the relative position and pose of the target with respect to the observer. The angular velocity and torque are expressed in the OCS.

### 2.2.1 Linear motion

The equations for linear motion can be described very concisely. From Newton's second law of motion:

$$\ddot{\boldsymbol{s}} = \dot{\boldsymbol{v}} = \frac{\boldsymbol{F}}{M}$$

The following discrete-time solutions are exact if $\boldsymbol{F}$ is constant over the time increment $\Delta t$:

$$\begin{aligned} \boldsymbol{s}(t + \Delta t) &= \boldsymbol{s}(t) + \boldsymbol{v}(t)\Delta t + \tfrac{1}{2M}\boldsymbol{F}(t)\Delta t^2 \\ \boldsymbol{v}(t + \Delta t) &= \boldsymbol{v}(t) + \tfrac{1}{M}\boldsymbol{F}(t)\Delta t \end{aligned}$$

### 2.2.2 Angular motion

We use quaternions to represent 3D rotation (attitude). Quaternions have distinct advantages over certain competing attitude representations. A discussion on popular attitude representation schemes can be found in appendix D. There is indeed more than one way to define quaternion attitude coordinates – we choose to use the definition used by [9, 10], of which the definition can be found in section §D.4.

We can express the time derivative of $\boldsymbol{q}$ as a matrix multiplication, or more succinctly as a quaternion product[2]:

$$\dot{\boldsymbol{q}} = \frac{1}{2}\left(\boldsymbol{\omega} * \boldsymbol{q}\right)$$

The following discrete-time closed-form solution is exact if $\boldsymbol{\omega}$ is constant over the time interval (see [9]):

$$\boldsymbol{q}(t+\Delta t) \quad = \quad \left[\cos\left(\frac{\|\boldsymbol{\omega}(t)\|\Delta t}{2}\right)\boldsymbol{I}_4 + \frac{1}{\|\boldsymbol{\omega}(t)\|}\sin\left(\frac{\|\boldsymbol{\omega}(t)\|\Delta t}{2}\right)\Omega(t)\right]\boldsymbol{q}(t),$$

where $\boldsymbol{I}_4$ is the $4\times 4$ identity matrix, $\|\boldsymbol{\omega}\| = \sqrt{\omega_1^2+\omega_2^2+\omega_3^2}$ and

$$\Omega = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix}.$$

Assuming that the inertia matrix $I$ is diagonal (the axes can be chosen as such), we can write down Euler's moment equation as three scalar equations:

$$\begin{aligned} \dot{\omega}_x &= \frac{1}{I_{xx}}\left(\Gamma_x - \omega_y\omega_z(I_{zz}-I_{yy})\right) \\ \dot{\omega}_y &= \frac{1}{I_{yy}}\left(\Gamma_y - \omega_x\omega_z(I_{xx}-I_{zz})\right) \\ \dot{\omega}_z &= \frac{1}{I_{zz}}\left(\Gamma_z - \omega_x\omega_y(I_{yy}-I_{xx})\right) \end{aligned}$$

The equations above are used to update the angular velocity with the following discrete-time approximation:

$$\boldsymbol{\omega}(t+\Delta t) \approx \boldsymbol{\omega}(t) + \dot{\boldsymbol{\omega}}\Delta t.$$

[2] $\boldsymbol{\omega}$ is $\boldsymbol{\omega}$ interpreted as a quaternion, by adding a zero scalar component

## 2.3 Orbital motion between Satellites

We will assume that the observing and target satellites are in near-identical and near-circular orbits. This is a fair assumption, since one could argue that a large percentage of satellites are kept in near-circular orbits. A circular orbit dictates constant angular velocity and altitude, which is an advantage to imaging and station keeping, and makes formation flight considerably easier.

Let's now examine the relative motion of two satellites in near-circular orbits.

We will define a right-handed coordinate system as shown in figure 2.6. The $x$-axis is pointing along the velocity vector (tangential to the orbit), the $z$-axis points towards nadir in the orbital plane, with the $y$-axis perpendicular to the orbital plane and completing the right-handed set. Note that the definition of the axes differs slightly from the convention used for defining the CCS. [3] The reason behind this definition is to stay consistent with the conventional orbital axis definition.

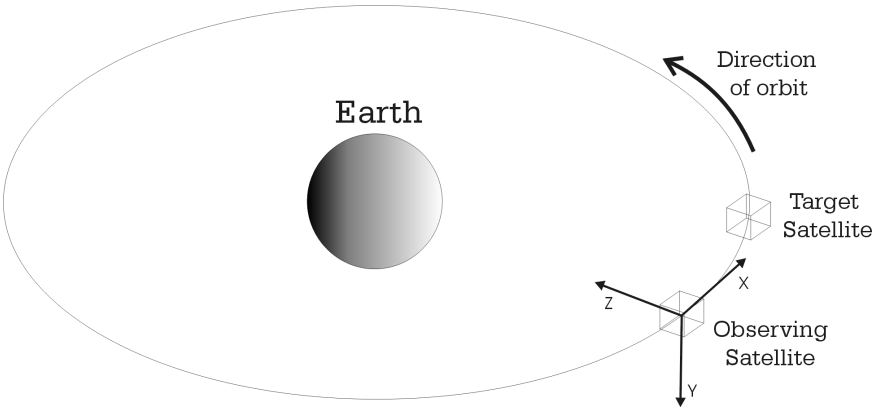



**Figure 2.6:** Orbital model: Definition of coordinate system

The relative motion of two satellites with respect to one another, when both are in circular or near-circular orbits, are described by Hill's equations (also called the Clohessey-Wiltshire equations) [11]. These equations are based on a linearised model. A solution to relative motion for more general elliptical orbits have been proposed by [12], but the applicability to practical problems was not clearly shown.

[3] The definition used here is as if the CCS was set up for a camera looking constantly at nadir and not, say, at the target.

Hill's equations for the coordinate system of figure 2.6 are

$$\begin{aligned} \ddot{x} &= 2\omega_0\dot{z} + f_x \\ \ddot{y} &= -\omega_0^2 y + f_y \\ \ddot{z} &= -2\omega_0\dot{x} + 3\omega_0^2 z + f_z \end{aligned} \tag{2.3.1}$$

where $\omega_0$ is the orbital angular velocity of the satellites (assumed practically equal), and $f_x$, $f_y$ and $f_z$ are the components of the disturbance specific force along the coordinate axes.

Note that the in-plane and out-of-plane motion are uncoupled. The out-of-plane motion is equivalent to the undamped motion of a mass on a massless spring with eigenvalues $\pm\omega_0 j$. The eigenvalues of the in-plane system are $0, 0, \pm\omega_0 j$. The associated modes are constant separation along the $x$-axis, constant velocity along the $x$-axis, and periodic motion of one satellite around the other [11]. A constant out-of-plane disturbance results in displaced undamped oscillatory motion along the y-axis. An (inertially) constant in-plane disturbance (such as a differential solar radiation force) cycles periodically with frequency $\omega_0$ when expressed in the coordinate system of figure 2.6. This forces the in-plane relative position to grow unbounded until the linearised equations (2.3.1), (2.3.1) and (2.3.1) no longer apply.
A closed-loop controller can be implemented to maintain proper separation [11, 13]. We are now only concerned with pose and position estimation, and therefore the control aspects of formation flying are not considered further.

If we consider the case in which the disturbance specific force is zero, we can solve Hill's equations analytically as follows:

$$\begin{aligned} z &= (4z_0 - 2\frac{\dot{x}_0}{\omega_0}) + (\frac{2\dot{x}_0}{\omega_0} - 3z_0)\cos(\omega_0 t) + \frac{\dot{z}_0}{\omega_0}\sin(\omega_0 t); \\ y &= \frac{\dot{y}_0}{\omega_0}\sin(\omega_0 t) + y_0\cos(\omega_0 t); \\ x &= (x_0 + 2\frac{\dot{z}_0}{\omega_0}) + (4\frac{\dot{x}_0}{\omega_0} - 6z_0)\sin(\omega_0 t) - 2\frac{\dot{z}_0}{\omega_0}\cos(\omega_0 t) - (3\dot{x}_0 - 6\omega_0 z_0)t \end{aligned} \tag{2.3.2}$$

From the above analytical solution we see that the $x$-axis motion is only periodic if the initial conditions satisfy $\dot{x}_0 = 2\omega_0 z_0$.

When we model the relative motion of two satellites for tracking purposes, we can assume that some formation keeping control system will be used. One of the tasks of the control system would be to negate perturbations to the relative orbits. The net effect is that the disturbance specific force can be modeled as zero, and equations equation (2.3.2) can be used. If the in-plane disturbance specific force is not zero the orbits might easily diverge, as shown in [11].

# Chapter 3

# Calibration Algorithms

Several successful algorithms have been developed to estimate translation and pose from 2D-3D correspondences. Most of these algorithms assume the perspective camera model or extensions thereof. Algorithms which non-recursively process an observation of $n$ 2D-3D point-pairs at a single instant, are usually called calibration algorithms, since an observation of $n$ points may be used to calibrate all the extrinsic and intrinsic parameters of a camera (explained shortly).

The intrinsic parameters include the focal length, lens distortion parameters, focal plane offset and conversion between pixel and real coordinates. We assume that all intrinsic parameters are known to us, since they will not change with time and may be measured at any stage (or obtained from the manufacturer).

The extrinsic parameters consist of the translation and rotation of the camera relative to our (usually inertial) world coordinate system. This is the same as finding the position and pose of the OCS relative to the CCS, and are exactly the parameters we want to estimate.

Salvi et al. [14] gives an excellent overview of some of the popular calibration algorithms in use today. We have implemented two of these algorithms for pose estimation – an intuitive direct least squares solution, and the method of Faugeras (without lens distortion).

Note that the essentially nonlinear equations of the perspective projection become algebraically linear when using homogeneous coordinates [14] – hence matrix equations can be used.

## 3.1 Method of Hall

Hall's method [14] is akin to finding (by least squares approximation) the best $3 \times 4$ matrix $\boldsymbol{A}$ to solve the typically over-determined equation

$$\boldsymbol{p} = \boldsymbol{AP}$$

where $\boldsymbol{P}$ is a $4 \times n$ matrix consisting of $n$ 3D points (in homogeneous coordinates) and $\boldsymbol{p}$ is the $3 \times n$ matrix of 2D observations (in homogeneous coordinates). Because the equation is written in homogeneous coordinates (see appendix C) the matrix $\boldsymbol{A}$ has 11 degrees of freedom. No other constraints are placed of $\boldsymbol{A}$.

The advantage in this lack of constraints is that any type of linear projection can be modeled. The disadvantage is that there is no direct way of interpreting the projection matrix $\boldsymbol{A}$ in terms of a known translation and rotation. We will not consider Hall's method further.

## 3.2 Direct solution by Least Squares

An intuitive approach is to interpret the matrix $\boldsymbol{A}$ more literally as a rotation and translation, followed by a perspective projection. We have shown in section §2.1 that we can represent the 3D-2D projection as

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} -f & 0 & 0 \\ 0 & -f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

where $R_{11} \ldots R_{33}$ and $[t_x \; t_y \; t_z]$ describe the (as yet unknown) rotation and translation. The parameters $f$, $[X \; Y \; Z]^T$ and $[x \; y]^T$ are known. The nonzero scaling parameter $s$ falls away and is inconsequential.

The matrices can be multiplied to obtain

$$\begin{bmatrix} sx \\ sy \\ s \end{bmatrix} = \begin{bmatrix} -fR_{11} & -fR_{12} & -fR_{13} & -ft_x \\ -fR_{21} & -fR_{22} & -fR_{23} & -ft_y \\ R_{31} & R_{32} & R_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix},$$

which can be rewritten as follows:[1]

$$x = -f\frac{R_{11}X + R_{12}Y + R_{13}Z + t_x}{R_{31}X + R_{32}Y + R_{33}Z + t_z} \quad \text{and} \quad y = -f\frac{R_{21}X + R_{22}Y + R_{23}Z + t_y}{R_{31}X + R_{32}Y + R_{33}Z + t_z}.$$

The variables are now rearranged so that

$$R_{11}fX + R_{12}fY + R_{13}fZ + R_{31}xX + R_{32}xY + R_{33}xZ + t_xf + t_zx = 0$$

$$R_{21}fX + R_{22}fY + R_{23}fZ + R_{31}yX + R_{32}yY + R_{33}yZ + t_yf + t_zy = 0.$$

The above two equations are assumed valid for every 3D-2D observation pair, so that by stacking the equations for each point we form the linear matrix equation

$$\boldsymbol{Q}\boldsymbol{v} = \boldsymbol{0},$$

where

$$\boldsymbol{Q} = \begin{bmatrix} fX_1 & fY_1 & fZ_1 & 0 & 0 & 0 & x_1X_1 & x_1Y_1 & x_1Z_1 & f & 0 & x_1 \\ 0 & 0 & 0 & fX_1 & fY_1 & fZ_1 & yX_1 & y_1Y_1 & y_1Z_1 & 0 & f & y_1 \\ fX_2 & fY_2 & fZ_2 & 0 & 0 & 0 & x_2X_2 & x_2Y_2 & x_2Z_2 & f & 0 & x_2 \\ 0 & 0 & 0 & fX_2 & fY_2 & fZ_2 & yX_2 & y_2Y_2 & y_2Z_2 & 0 & f & y_2 \\ & \vdots & & & \vdots & & & \vdots & & & \vdots & \\ fX_N & fY_N & fZ_N & 0 & 0 & 0 & x_NX_N & x_NY_N & x_NZ_N & f & 0 & x_N \\ 0 & 0 & 0 & fX_N & fY_N & fZ_N & yX_N & y_NY_N & y_NZ_N & 0 & f & y_N \end{bmatrix}$$

and

$$\boldsymbol{v} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & R_{21} & R_{22} & R_{23} & R_{31} & R_{32} & R_{33} & t_x & t_y & t_z \end{bmatrix}^T.$$

Singular value decomposition (SVD) is now used to determine the vector which minimizes the magnitude of $\boldsymbol{Q}\boldsymbol{v}$, and thus solves the (typically) overdetermined equation $\boldsymbol{Q}\boldsymbol{v} = \boldsymbol{0}$ in a least squares sense.

Note that this solution does not explicitly force the rotation matrix to be in the proper orthonormal form. The first step would be to manually normalize the matrix so that it has unit norm, and then to compute a rotation quaternion from the resulting matrix. The lack of constraints on $R_{11} \dots R_{33}$ makes the use of at least 6 point pairs a necessity since we are dealing with 12 degrees of freedom when computing the SVD.

---

[1] The equations are now recognizable in the familiar Euclidean form.

## 3.3 Method of Faugeras

The method of Faugeras [14] is very similar to the direct least squares approach, but computes the parameters in a slightly different way.

We start with the same equation as in Hall's method:

$$x = -f\frac{R_{11}X + R_{12}Y + R_{13}Z + t_x}{R_{31}X + R_{32}Y + R_{33}Z + t_z} \quad \text{and} \quad y = -f\frac{R_{21}X + R_{22}Y + R_{23}Z + t_y}{R_{31}X + R_{32}Y + R_{33}Z + t_z}$$

We now factor the equations with respect to the unknowns:

$$x = -f\boldsymbol{P}\cdot\left(\frac{\boldsymbol{r_1}}{t_z}\right) - f\left(\frac{t_x}{t_z}\right) - x\boldsymbol{P}\cdot\left(\frac{\boldsymbol{r_3}}{t_z}\right) \quad \text{and} \quad y = -f\boldsymbol{P}\cdot\left(\frac{\boldsymbol{r_2}}{t_z}\right) - f\left(\frac{t_y}{t_z}\right) - y\boldsymbol{P}\cdot\left(\frac{\boldsymbol{r_3}}{t_z}\right)$$

where $\boldsymbol{P} = [X\ Y\ Z]^T$ and $\boldsymbol{r_i}$ indicates the $i$th row of the $3 \times 3$ rotation matrix.

We now consider the set of five unknown parameters

$$(\boldsymbol{\rho_1}, \boldsymbol{\rho_2}, \boldsymbol{\rho_3}, \tau_1, \tau_2) \equiv \left(\frac{\boldsymbol{r_1}}{t_z}, \frac{\boldsymbol{r_2}}{t_z}, \frac{\boldsymbol{r_3}}{t_z}, \frac{t_x}{t_z}, \frac{t_y}{t_z}\right).$$

Note that the vector of unknowns has 11 degrees of freedom – one less than in the direct approach, while still retaining physical interpretation of the rotation and translation.

We can now use a least squares solution for solving the following equation:

$$\boldsymbol{Q}\boldsymbol{v} = \boldsymbol{b},$$

where

$$\boldsymbol{Q} = -\begin{bmatrix} fX_1 & fY_1 & fZ_1 & 0 & 0 & 0 & x_1X_1 & x_1Y_1 & x_1Z_1 & f & 0 \\ 0 & 0 & 0 & fX_1 & fY_1 & fZ_1 & y_1X_1 & y_1Y_1 & y_1Z_1 & 0 & f \\ fX_2 & fY_2 & fZ_2 & 0 & 0 & 0 & x_2X_2 & x_2Y_2 & x_2Z_2 & f & 0 \\ 0 & 0 & 0 & fX_2 & fY_2 & fZ_2 & y_2X_2 & y_2Y_2 & y_2Z_2 & 0 & f \\ & \vdots & & & \vdots & & \vdots & & & \vdots & \\ fX_N & fY_N & fZ_N & 0 & 0 & 0 & x_NX_N & x_NY_N & x_NZ_N & f & 0 \\ 0 & 0 & 0 & fX_N & fY_N & fZ_N & y_NX_N & y_NY_N & y_NZ_N & 0 & f \end{bmatrix},$$

$$\boldsymbol{v} = \begin{bmatrix} \boldsymbol{\rho_1} & \boldsymbol{\rho_2} & \boldsymbol{\rho_3} & \tau_1 & \tau_2 \end{bmatrix}^T,$$

and

$$\boldsymbol{b} = \begin{bmatrix} x_1 & y_1 & x_2 & y_2 & \dots & x_N & y_N \end{bmatrix}^T.$$

The solution can be obtained by using the pseudo-inverse

$$\boldsymbol{v} = \left(\boldsymbol{Q}^T\boldsymbol{Q}\right)^{-1}\boldsymbol{Q}^T\boldsymbol{b}.$$

We know that

$$\boldsymbol{v_1}\boldsymbol{v_2}^T = \|\boldsymbol{v_1}\|\,\|\boldsymbol{v_2}\|\cos\theta \quad \text{and} \quad \|\boldsymbol{v_1}\otimes\boldsymbol{v_2}\| = \|\boldsymbol{v_1}\|\,\|\boldsymbol{v_2}\|\sin\theta,$$

and also (since a rotation matrix is orthonormal)[2],

$$\boldsymbol{r_i}\boldsymbol{r_j}^T = 0, \quad i \neq j \qquad\qquad ||\boldsymbol{r_i}\otimes\boldsymbol{r_j}|| = 1, \quad i \neq j$$

$$\boldsymbol{r_i}\boldsymbol{r_j}^T = 1, \quad i = j \qquad\qquad ||\boldsymbol{r_i}\otimes\boldsymbol{r_j}|| = 0, \quad i = j.$$

We now compute the extrinsic parameters $t_x, t_y, t_z, R_{11}, R_{12}, \ldots, R_{33}$ as follows:

$$\begin{aligned}
t_x &= \frac{3\tau_1}{||\boldsymbol{\rho_1}|| + ||\boldsymbol{\rho_2}|| + ||\boldsymbol{\rho_3}||} \\
t_y &= \frac{3\tau_2}{||\boldsymbol{\rho_1}|| + ||\boldsymbol{\rho_2}|| + ||\boldsymbol{\rho_3}||} \\
t_z &= \frac{3}{||\boldsymbol{\rho_1}|| + ||\boldsymbol{\rho_2}|| + ||\boldsymbol{\rho_3}||} \\
\boldsymbol{r_1} &= \frac{||\boldsymbol{\rho_3}||}{||\boldsymbol{\rho_1}\otimes\boldsymbol{\rho_3}||}\left(\boldsymbol{\rho_1} - \frac{\boldsymbol{\rho_1}\boldsymbol{\rho_3}^T}{||\boldsymbol{\rho_3}||^2}\boldsymbol{\rho_3}\right) \\
\boldsymbol{r_2} &= \frac{||\boldsymbol{\rho_3}||}{||\boldsymbol{\rho_2}\otimes\boldsymbol{\rho_3}||}\left(\boldsymbol{\rho_2} - \frac{\boldsymbol{\rho_2}\boldsymbol{\rho_3}^T}{||\boldsymbol{\rho_3}||^2}\boldsymbol{\rho_3}\right) \\
\boldsymbol{r_3} &= \frac{\boldsymbol{\rho_3}}{||\boldsymbol{\rho_3}||}
\end{aligned}$$

The computed rows of the DCM are always orthonormal, through the use of Gram-Schmidt orthonormalisation in the equations above. Note that the computation of $t_x$ and $t_y$ differs substantially from the original method proposed in [14]. It is the author's opinion that the above method is more stable and intuitive than original method. This was confirmed with tests on actual images, but the reader should remember that we assume a known offset and scaling. In the more general case, where these are also unknown, we will have to resort to the equations in [14].

The computation of the rotation could be seen as "starting" with the computation of $\boldsymbol{r_3}$ as reference, and then computing the other two rows of the matrix orthogonal to $\boldsymbol{r_3}$. There is a valid reason for doing it this way. We may include terms for horizontal and vertical scaling of the image, and horizontal and vertical offsets relative to the optical centre (see [14]). In this case we cannot compute $\boldsymbol{r_1}$ or $\boldsymbol{r_2}$ as easily as $\boldsymbol{r_3}$, and therefore choose $\boldsymbol{r_3}$ as our "reference". The equations above ensure that the resulting direction cosine matrix $R$ will be orthonormal.

[2]Note that the vectors $\boldsymbol{r_i}$ and $\boldsymbol{\rho_i}$ are *row* vectors.

## 3.4 Number of matching points required

The output from each of the above algorithms is a $3 \times 1$ translation vector and a $3 \times 3$ "direction cosine matrix" (DCM) – a total of 12 parameters. These parameters are computed by doing a least squares fitting on 12 parameters (direct solution) or 11 parameters (Faugeras' method). Note, however, that a 3D rotation has only 3 degrees of freedom. In principle we would therefore only need to determine 6 independent parameters. The algorithms presented, however, do not properly take the structure of the rotation matrix into account when computing the parameters, even though they enforce constraints afterwards.

Since each point-correspondence provides us with two equations (one for X, and one for Y) we know that we need *at least* 6 corresponding pairs to find a reliable solution. In practice we could need more, since the points are not necessarily linearly independent.

## 3.5 Robustness of Direct- and Faugeras' methods

Both of the discussed calibration methods conceptually rely on the same mathematical principle. Since Faugeras' method solves the least squares problem for an unknown vector with 11 instead of 12 components, we can expect this method to be slightly more tolerant towards a shortage of points. The individual methods' robustness against noise is not immediately obvious – we will examine this through experimentation.

We chose a simulated cubical object, placed (on average) at the reference CCS coordinates of $[4 \ -7 \ 40]$ meters. A simulated perspective camera with an effective focal length of $-1000$ units (see section §2.1) was used to simulate test images. The cubical target had a side length of 1.5 meters – corresponding roughly to the simulated target of chapter 6. For each iteration, we changed the reference position by a Gaussian random translation, having a standard deviation of 5% of the reference. The reference orientation was chosen randomly for each iteration. For each set of paramters (pixel noise or number of features), a total of 500 iterations were run. The RMS error values (computed as described in section §6.1) are shown in figures 3.1 and 3.2. We chose to use uniform additive pixel noise, since digitization noise is the most likely source of 2D spatial image noise. Broida et al. [1] notes that a uniform distribution is a better approximation (than a Gaussian distribution) for modeling digitization errors.

We see in figures 3.1 and 3.2, that the direct least squares method consistently provides better 3D translation estimates than Faugeras' method, except for the case where only five features are visible. The translation error scales more or less linearly with the additive pixel noise. Experimentation showed that the translation errors along each axis were proportionally identical. Only the RMS distance between the true and estimated positions (as a percentage of the reference 3D position) are therefore shown.

Especially in figure 3.2, we see that Faugeras' method yields more accurate and consistent rota-

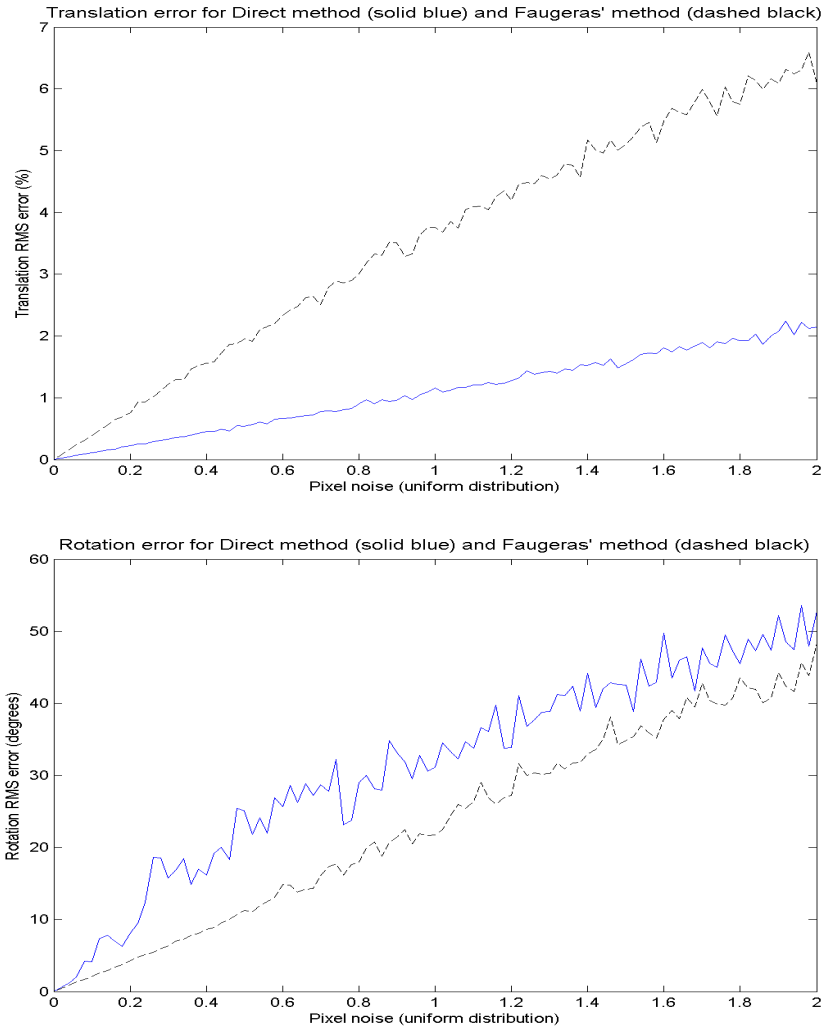



**Figure 3.1:** Calibration – Translation and rotation estimation errors for varying additive pixel noise (8 visible feature points)

tion estimates than the direct method. The proportional difference between the two methods' rotation estimates is, however, not as marked as with the translation estimates.

It is abundantly clear from figure 3.2 that both methods start to fail when fewer than six features are visible – although the direct method suffers the worst. The superior performance of Faugeras' method when few features are visible is as expected. This can be attributed to the fact that the vector solution to the matrix equation has one fewer element than with the direct method. It is clear that there is no big advantage in using more than 8 visible points, when using either of the two calibration algorithms.

Assuming that accurate position estimates might be more important to satellite formationkeeping than orientation estimates, and that we will have more than six visible features, the direct method might be more applicable to our task.

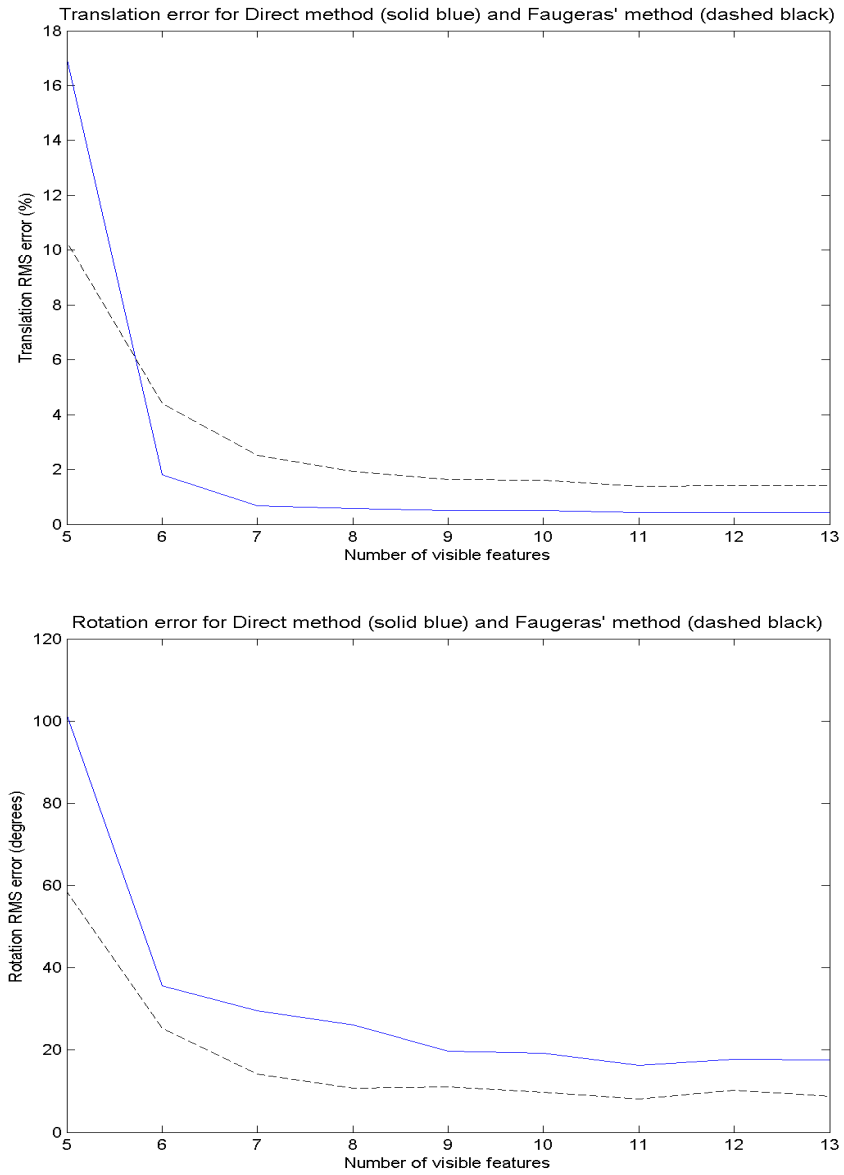



**Figure 3.2:** Calibration – Translation and rotation estimation errors for varying number of visible feature points. (Additive uniform noise of 0.5 pixels)

## 3.6 Overcoming a debilitating shortage of matching points

The standard calibration algorithms fail if fewer than 6 points are visible, and especially when the points are coplanar. Even calibration with six visible points starts to fail when there is only one point which lies out of the plane described by the other five.

We propose two methods for adapting the direct solution to cope with fewer point correspondences. The first method still allows us to compute a pose, while the second method only computes a translation.

### 3.6.1 Reconstructing the 3rd row of the DCM

We can commonly assume that the object's translation along the CCS $z$-axis is much larger than the offset of any object-point in the OCS (usually centered at the centre of mass). [3] Lets look at the equations derived in (section §3.2):

$$x = -f\frac{R_{11}X + R_{12}Y + R_{13}Z + t_x}{R_{31}X + R_{32}Y + R_{33}Z + t_z} \quad \text{and} \quad y = -f\frac{R_{21}X + R_{22}Y + R_{23}Z + t_y}{R_{31}X + R_{32}Y + R_{33}Z + t_z}.$$

Remember that the vector $\boldsymbol{P} = [X\ Y\ Z]^T$ describes the object point relative to the OCS, while the vector $\boldsymbol{T} = [t_x\ t_y\ t_z]^T$ describes the translation of the OCS relative to the CCS. The elements of the DCM which describe the rotation are given by $R_{11} \ldots R_{33}$.

The denominator will be dominated by $t_z$ if $\|\boldsymbol{P}\|$ is significantly smaller than $t_z$. The rotation has little effect on the relative sizes of the denominator's terms, since rotation preserves the norm of the vector being rotated.

We can therefore make the following approximation:

$$x \approx -f\frac{R_{11}X + R_{12}Y + R_{13}Z + t_x}{t_z} \quad \text{and} \quad y \approx -f\frac{R_{21}X + R_{22}Y + R_{23}Z + t_y}{t_z}.$$

We now have only 9 unknowns to compute – which means that we should be able to get along with only 5 matching 2D-3D pairs.

It is worth noting that we should make sure of the orthonormality of $\boldsymbol{r_1} = [R_{11}\ R_{12}\ R_{13}]$ and $\boldsymbol{r_2} = [R_{21}\ R_{22}\ R_{23}]$ before doing further computations. We can enforce orthogonality by Gram-Schmidt orthonormalisation, namely

$$\begin{aligned} \hat{\boldsymbol{r}}_1 &= \frac{\boldsymbol{r_1}}{\|\boldsymbol{r_1}\|} \\ \hat{\boldsymbol{r}}_2 &= \frac{\|\boldsymbol{r_1}\|}{\|\boldsymbol{r_2} \otimes \boldsymbol{r_1}\|}\left(\boldsymbol{r_2} - \frac{\boldsymbol{r_2}\boldsymbol{r_1}^T}{\|\boldsymbol{r_1}\|^2}\boldsymbol{r_1}\right). \end{aligned}$$

We can now compute the three missing parameters by making use of the othonormality property of the third DCM row.

$$[R_{31}\ R_{32}\ R_{33}]^T = \hat{\boldsymbol{r}}_1 \otimes \hat{\boldsymbol{r}}_2.$$

We have now computed the translation as well as the DCM representing the rotation.

### 3.6.2 Computing only the translation

If we have very few 2D-3D point correspondences we might consider ignoring the pose information. This situation might occur when the target is too far away to properly distinguish

[3] Think of a satellite with a 2m diameter being viewed from a distance of 50m

features (in which case information about the pose is less important than the translation).

Assuming that the object's translational parameters in the CCS is much larger than the offset of any object-point in the OCS, we can simplify the image projection to

$$x \approx -f\frac{t_x}{t_z} \quad \text{and} \quad y \approx -f\frac{t_y}{t_z}.$$

This is equivalent to assuming that every object point is located at the origin of the OCS (usually the object's centre of mass).

We now have only three parameters to obtain, since we are no longer interested in the rotation. We should therefore be able to get along with only 2 matching point pairs, although we can expect low accuracy due to the coarse approximation made.

Note that the relative 3D positions, $[X\ Y\ Z]^T$ for each of the the object's features, are no longer used. This means that we can no longer determine the scale of $t_x$, $t_y$ and $t_z$ directly. We can estimate the correct scaling by forcing the computed image to have approximately the same size as the observed image. (we could, for example, use the area spanned by all the visible points as a measure of "size")

## 3.7 Computational Complexity

We will now quickly look at the the number of computations which are required to execute the calibration algorithms. We will express the complexity of the calibration algorithm in terms of the number of feature points (designated by $N$) to be processed.

The solution to the calibration problem is found by computing the least-squares approximation to the nullspace of the matrix $\boldsymbol{Q}$ my means of the SVD algorithm. Computing the SVD of an $m \times n$ matrix requires $O(m^2n + mn^2)$ computations. Therefore this step requires $O(N^2)$ computations, since number of columns of $\boldsymbol{Q}$ is fixed.

The method of Faugeras in section §3.3 works according to the same principle as the Direct method. The computationally most expensive step is to compute the pseudo-inverse of $\boldsymbol{Q}$ to solve for the vector $\boldsymbol{v}$. This can be done in $O(N^2)$ computations.

The number of computations will therefore increase with the square of the number of matching feature points. In practice we will most likely work with fewer that twenty feature point pairs (we only need six or more). For a small number of features like this, the asymptotic bound of $O(N^2)$ is not of much relevance. The important point is that the number of computations required is not prohibitive.

# Chapter 4

# The Kalman Filter: Interpreting measurements

Filtering is the process by which estimates of a set of states (the state vector) are computed concurrently with new measurements becoming available. The states are often “hidden" – in the sense that they are only observed *indirectly* via measurements. These measurements are sometimes linear, but often a nonlinear function of the state to be estimated. The problem of state estimation is of paramount importance in many fields of science, engineering and finance.

The Kalman Filter, first presented by R.E. Kalman [15] in 1960 proves an extremely useful tool for tackling this problem from a Gaussian statistical point of view. The original filter was applicable only to linear problems, but several linearisations have been developed for use in nonlinear problems. Refer to sections §4.1, §4.2 and §4.3 as well as appendices A and B for in-depth explanations and some derivations of the Linear, Extended and Unscented Kalman Filters.

A visualization of the way in which a (scalar) Gaussian estimate and (scalar) Gaussian measurement can be combined to form a new Gaussian estimate is shown in figure 4.1. This is conceptually the same way in which the Linear Kalman filter treats multidimensional estimates, and measurements of linear systems.

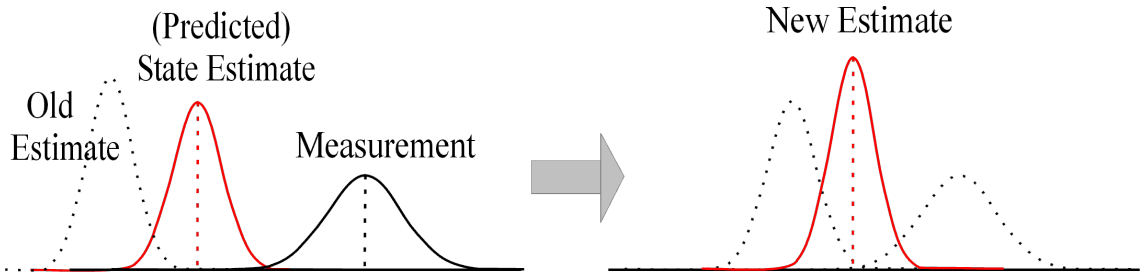



**Figure 4.1:** The concept behind Kalman’s filter: Combining two Gaussian density functions

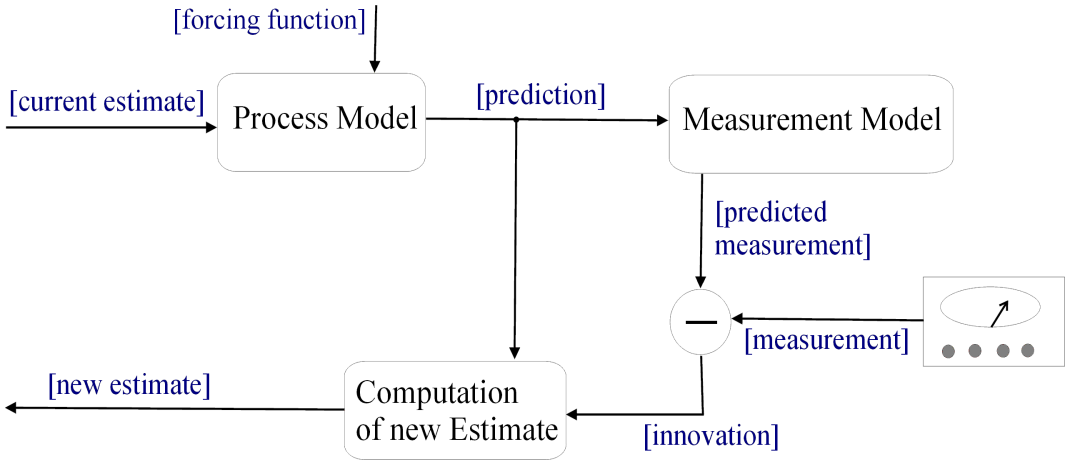



**Figure 4.2:** The basic elements of recursive filtering

Filtering is treated as a recursive computation, since estimates have to be computed as measurements become available. We model the time evolution of the system, and also the measurement process. Additionally, we model the noise inherent to both the system model and the measurements. These models typically exhibit complex nonlinearities, thus precluding analytical solution. The idea behind recursive filtering is shown in figure 4.2. In the case of the Kalman Filter each estimate is represented by a Gaussian distribution. The Kalman Filter consists of a set of rules which propagate the current estimate as a Gaussian distribution, and then updates this Gaussian estimate when measurements become available, as illustrated in figure 4.3.

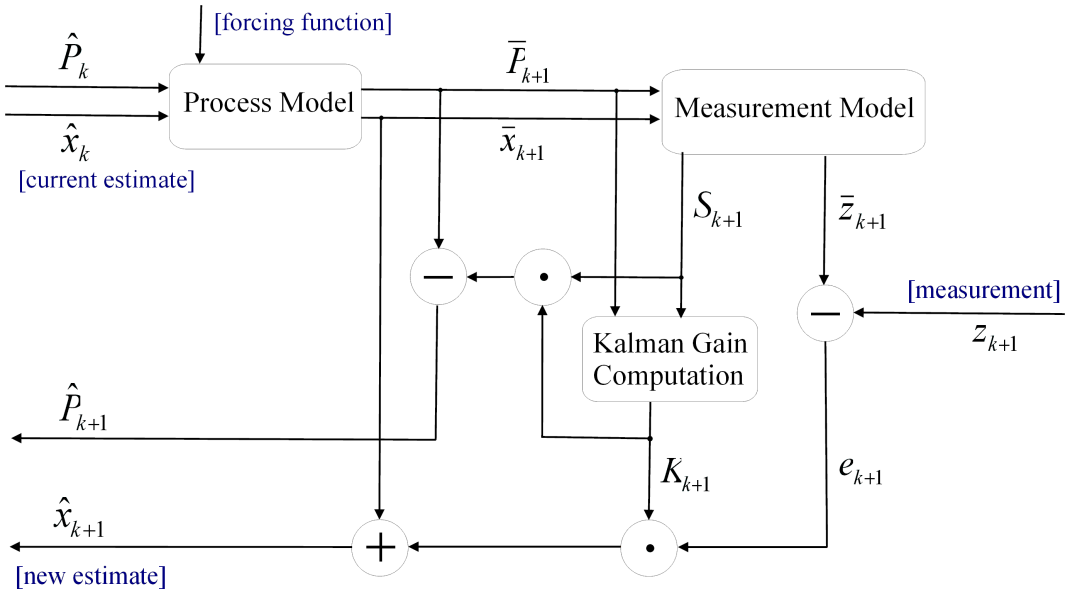



**Figure 4.3:** The operation of the Kalman filter: an overview

## 4.1 The Linear Kalman Filter (LKF)

The LKF is used to compute an optimal state estimate when the process model and measurement function are both linear. The filter assumes that the process and measurement noise are both Gaussian. Refer to appendix A for the derivation of the discrete LKF.

We start with an initial state estimate $\boldsymbol{x}_0$ with covariance $\boldsymbol{P}_0$. The state transition matrix is given by $\boldsymbol{\Phi}$. The covariance matrix of the process noise is given by $\boldsymbol{Q}$ and the covariance of the measurement noise by $\boldsymbol{R}$. $\boldsymbol{\Gamma}$ is the "process noise distribution matrix", which transforms the process noise $\boldsymbol{Q}$ to the coordinates of the state vector $\boldsymbol{x}$.[1] The external force (driving force) acting on the system is expressed by $\boldsymbol{u_k}$, while $\boldsymbol{\Psi}$ converts the external force to its perturbating effect on the state vector.[2] The linear measurement of the state vector is performed by multiplying by $\boldsymbol{H}$.

The LKF algorithm [16, 17] is iteratively executed by repeating the following steps :

$$\bar{\boldsymbol{x}}_{k+1} = \boldsymbol{\Phi}_k \hat{\boldsymbol{x}}_k + \boldsymbol{\Psi}_k \boldsymbol{u}_k \tag{4.1.1}$$

$$\bar{\boldsymbol{P}}_{k+1} = \boldsymbol{\Phi}_k \hat{\boldsymbol{P}}_k \boldsymbol{\Phi}_k^T + \boldsymbol{\Gamma}_k \boldsymbol{Q}_k \boldsymbol{\Gamma}_k^T \tag{4.1.2}$$

$$\bar{\boldsymbol{z}}_{k+1} = \boldsymbol{H}_{k+1} \bar{\boldsymbol{x}}_{k+1} \tag{4.1.3}$$

$$\boldsymbol{S}_{k+1} = \boldsymbol{H}_{k+1} \bar{\boldsymbol{P}}_{k+1} \boldsymbol{H}_{k+1}^T + \boldsymbol{R}_{k+1} \tag{4.1.4}$$

$$\boldsymbol{K}_{k+1} = \bar{\boldsymbol{P}}_{k+1} \boldsymbol{H}_{k+1}^T \boldsymbol{S}_{k+1}^{-1} \tag{4.1.5}$$

$$\hat{\boldsymbol{x}}_{k+1} = \bar{\boldsymbol{x}}_{k+1} + \boldsymbol{K}_{k+1}(\boldsymbol{z}_{k+1} - \bar{\boldsymbol{z}}_{k+1}) \tag{4.1.6}$$

$$\hat{\boldsymbol{P}}_{k+1} = \bar{\boldsymbol{P}}_{k+1} - \boldsymbol{K}_{k+1} \boldsymbol{S}_{k+1} \boldsymbol{K}_{k+1}^T \tag{4.1.7}$$

In equation (4.1.1) we predict the state at the next time-step. The state covariance is predicted by equation (4.1.2). The new measurement is predicted in equation (4.1.3). Now, when a measurement becomes available, we compute the Kalman gain by equation (4.1.5). We update our state estimate by adding the product of the Kalman gain, and the innovation to the predicted value in equation (4.1.6). Finally the state covariance matrix is updated in equation (4.1.7).

[1] It is possible to include the noise distribution in $\boldsymbol{Q}$ so that $\boldsymbol{\Gamma}$ is not explicitly shown. This is mostly the case, since proper noise values are commonly obtained heuristically or experimentally.

[2] It is commonly assumed that $\boldsymbol{u_k} = \boldsymbol{0}$ since external disturbances are usually unknown, and can be included in the noise model.

## 4.2 The Extended Kalman Filter (EKF)

The EKF was developed to extend the methodology of the LKF to nonlinear problems. The EKF is based upon the principle of linearizing the measurement and state evolution models, by using Taylor series expansions around the current state estimate. Refer to appendix B for the derivation of the discrete EKF.

Assume that the state's (nonlinear) time update between measurements is $\boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k)$, and that the nonlinear measurement equation is given by $\boldsymbol{z}_k = h_k(\boldsymbol{x}_k)$.

Due to the computation of Jacobians it is possible for the filter to diverge if the initial estimates or system model are poor, or if the time steps are too large. Recent research [18] suggest furthermore that there are some serious pitfalls to avoid when using the EKF. The system may diverge when the cross correlation between state elements skew predictions.

The Extended Kalman-filtering algorithm [19, 17, 20] is iteratively executed by repeating the following steps:

$$\bar{\boldsymbol{x}}_{k+1} = f_k(\hat{\boldsymbol{x}}_k) \tag{4.2.1}$$

$$\boldsymbol{F}_k = \left.\frac{\partial f_k(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_k} \tag{4.2.2}$$

$$\bar{\boldsymbol{P}}_{k+1} = \boldsymbol{F}_k \hat{\boldsymbol{P}}_k \boldsymbol{F}_k^T + \boldsymbol{Q}_k \tag{4.2.3}$$

$$\bar{\boldsymbol{z}}_{k+1} = h_{k+1}\left(\bar{\boldsymbol{x}}_{k+1}\right) \tag{4.2.4}$$

$$\boldsymbol{H}_{k+1} = \left.\frac{\partial h_{k+1}(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\bar{\boldsymbol{x}}_{k+1}} \tag{4.2.5}$$

$$\boldsymbol{S}_{k+1} = \boldsymbol{H}_{k+1} \bar{\boldsymbol{P}}_{k+1} \boldsymbol{H}_{k+1}^T + \boldsymbol{R}_{k+1} \tag{4.2.6}$$

$$\boldsymbol{K}_{k+1} = \bar{\boldsymbol{P}}_{k+1} \boldsymbol{H}_{k+1}^T \boldsymbol{S}_{k+1}^{-1} \tag{4.2.7}$$

$$\hat{\boldsymbol{x}}_{k+1} = \bar{\boldsymbol{x}}_{k+1} + \boldsymbol{K}_{k+1}(\boldsymbol{z}_{k+1} - \bar{\boldsymbol{z}}_{k+1}) \tag{4.2.8}$$

$$\hat{\boldsymbol{P}}_{k+1} = \bar{\boldsymbol{P}}_{k+1} - \boldsymbol{K}_{k+1} \boldsymbol{S}_{k+1} \boldsymbol{K}_{k+1}^T \tag{4.2.9}$$

We see that these equations are very similar in form to the LKF. Notable exceptions are the nonlinear time update and measurement functions, and the use of linearisations (Jacobians) to update the error covariance matrices.

The update of $\boldsymbol{P}$ (in the last step) is in slightly different form than usual [16, 19, 20], but should be numerically superior [17] since it guarantees symmetry by its symmetric formulation.

## 4.3 The Unscented Kalman Filter (UKF)

It has recently been argued [21, 22, 23, 4] that a new version of the Kalman Filter – the Unscented Kalman Filter (UKF) – provides superior performance to the EKF under certain conditions. Julier and Uhlmann [21] have shown that the UKF can lead to more accurate results than the EKF, and that it generates much better estimates of especially the state covariance (the EKF supposedly underestimates this quantity). Some proponents [24] argue that the UKF provides computational savings due to the absence of Jacobian matrices. Other researchers [25] claim the opposite result for tracking quaternion motion – similar or reduced accuracy compared to the EKF, and at a *higher* computational cost.

The EKF computes the Jacobian of the state vector, to linearly approximate the error transition matrix $\mathbf{\Phi}$. This is done to propagate the state covariance matrix $\boldsymbol{P}$. The UKF resolves this problem by *simulating* the propagation of the covariance matrix $\boldsymbol{P}$. A set of so-called sigma points are generated, which has the same mean and covariance as $\boldsymbol{P}$. The points are then nonlinearly propagated (using the full nonlinear model), and the mean and covariance of the new distribution is computed to update $\boldsymbol{P}$. An added advantage is the fact that no Jacobian has to be computed, although the computation of the sigma points offsets this advantage.

The UKF relies on the *scaled unscented transform* [22] to propagate the mean and the covariance of the state vector. There are three parameters which must be specified prior to the execution of the UKF filter, namely $\kappa$, $\alpha$ and $\beta$.

- Choose $\kappa \geq 0$ to guarantee positive semi-definiteness of the covariance matrix. A good default choice is $\kappa = 0$.
- Choose $0 \leq \alpha \leq 1$. $\alpha$ controls the “size" of the sigma point distribution and should be small if the non-linearities are strong.
- Choose $\beta \geq 0$. $\beta$ is a term for weighing the zeroth sigma point, and is optimally chosen as $\beta = 2$ for Gaussian distributions.
- Compute $\lambda = \alpha^2 (n + \kappa) - n$, where $n$ is the dimension of the state vector.
- Compute the weights (superscript used for index):
  $W_m^{(0)} = \lambda/(n + \lambda)$,
  $W_c^{(0)} = \lambda/(n + \lambda) + (1 - \alpha^2 + \beta)$,
  $W_m^{(i)} = W_c^{(i)} = 1/(2(n + \lambda)), \qquad i = 1, \ldots, 2n.$

The UKF algorithm [21, 17] is iteratively executed by repeating the following steps:

$$\begin{aligned}
\boldsymbol{U}_k &= \sqrt{(n+\lambda)\hat{\boldsymbol{P}}_k} && (4.3.1)\\
\sigma_j &\leftarrow n \text{ columns of } \boldsymbol{U}_k \text{ for } j = 1, 2, \ldots, n && (4.3.2)\\
\boldsymbol{\chi}_k^{(0)} &= \hat{\boldsymbol{x}}_k, \qquad \boldsymbol{\chi}_k^{(2j-1)} = \hat{\boldsymbol{x}}_k + \sigma_j, \qquad \boldsymbol{\chi}_k^{(2j)} = \hat{\boldsymbol{x}}_k - \sigma_j && (4.3.3)\\
\bar{\boldsymbol{\chi}}_{k+1}^{(i)} &= f_k\left(\boldsymbol{\chi}_k^{(i)}\right), \quad i = 1, 2, \ldots, 2n && (4.3.4)\\
\bar{\boldsymbol{x}}_{k+1} &= \sum_{i=0}^{2n} W_m^{(i)} \bar{\boldsymbol{\chi}}_{k+1}^{(i)} && (4.3.5)\\
\bar{\boldsymbol{P}}_{k+1} &= \sum_{i=0}^{2n} W_c^{(i)} [\bar{\boldsymbol{\chi}}_{k+1}^{(i)} - \bar{\boldsymbol{x}}_{k+1}][\bar{\boldsymbol{\chi}}_{k+1}^{(i)} - \bar{\boldsymbol{x}}_{k+1}]^T + \boldsymbol{Q} && (4.3.6)\\
\bar{\boldsymbol{\zeta}}_{k+1}^{(i)} &= h_{k+1}\left(\bar{\boldsymbol{\chi}}_{k+1}^{(j)}\right), \quad i = 1, 2, \ldots, 2n && (4.3.7)\\
\bar{\boldsymbol{z}}_{k+1} &= \sum_{i=0}^{2n} W_m^{(i)} \bar{\boldsymbol{\zeta}}_{k+1}^{(i)} && (4.3.8)\\
\boldsymbol{S}_{k+1} &= \sum_{i=0}^{2n} W_c^{(i)} [\bar{\boldsymbol{\zeta}}_{k+1}^{(i)} - \bar{\boldsymbol{z}}_{k+1}][\bar{\boldsymbol{\zeta}}_{k+1}^{(i)} - \bar{\boldsymbol{z}}_{k+1}]^T + \boldsymbol{R}_{k+1} && (4.3.9)\\
\boldsymbol{C}_{k+1} &= \sum_{i=0}^{2n} W_c^{(i)} [\bar{\boldsymbol{\chi}}_{k+1}^{(i)} - \bar{\boldsymbol{x}}_{k+1}][\bar{\boldsymbol{\zeta}}_{k+1}^{(i)} - \bar{\boldsymbol{z}}_{k+1}]^T && (4.3.10)\\
\boldsymbol{K}_{k+1} &= \boldsymbol{C}_{k+1} \boldsymbol{S}_{k+1}^{-1} && (4.3.11)\\
\hat{\boldsymbol{x}}_{k+1} &= \bar{\boldsymbol{x}}_{k+1} + \boldsymbol{K}_{k+1}(\boldsymbol{z}_{k+1} - \bar{\boldsymbol{z}}_{k+1}) && (4.3.12)\\
\hat{\boldsymbol{P}}_{k+1} &= \bar{\boldsymbol{P}}_{k+1} - \boldsymbol{K}_{k+1} \boldsymbol{S}_{k+1} \boldsymbol{K}_{k+1}^T && (4.3.13)
\end{aligned}$$

The matrix root equation (4.3.1) is computed via Cholesky factorization.[3] The sigma points are defined in equation (4.3.3), and propagated by equation (4.3.4). The predicted mean and covariance are computed in equations (4.3.5) and (4.3.6). The measurement of each propagated sigma point is computed in equation (4.3.7) and the predicted measurement in equation (4.3.8). The Kalman gain is computed in equation (4.3.11). The updated estimate of the state mean and covariance are computed in equations (4.3.12) and (4.3.13).

[3] Cholesky factorization is usually order $n^3/6$ computations, but Van der Merwe [22] claims that it can be computed in order $n^2$ if the covariance matrices are expressed recursively.

## 4.4 Computational complexity

We will now look at the the number of computations[4] which are required to execute each of the aforementioned filters. Let's assume a linear system with an $n$-dimensional state vector, and a measurement vector with dimension $m$.

### 4.4.1 Computational complexity of the LKF

We now refer to the LKF equations of section §4.1.

- The state prediction equation (4.1.1) involves a matrix multiplication between the $n \times n$ transition matrix $\mathbf{\Phi}$ and the $n$-dimensional vector $\boldsymbol{x}$, which amounts to $O(n^2)$ computations[5].
- The state covariance prediction equation (4.1.2) involves matrix multiplication between $n \times n$ matrices, which amounts to $O(n^3)$ computations.
- The measurement prediction equation (4.1.3) is $O(mn)$.
- The measurement covariance prediction equation (4.1.4) requires $O(mn^2 + m^2n)$ operations.
- The computation of the Kalman gain in equation (4.1.5) requires inversion of an $m \times m$ matrix, which is $O(m^3)$, coupled with matrix multiplication of $O(mn^2 + m^2n)$.
- The state update equation (4.1.6) involves mainly a matrix-vector product, and is $O(mn)$.
- The state covariance update equation (4.1.7) involves a matrix product of $O(mn^2 + m^2n)$.

In a typical implementation each of the above steps will be executed once per sampling interval, resulting in a $O(n^3 + m^3)$ computations per iteration. Note that many of the matrices in practical applications are often sparse, which might reduce the actual number of computations in a clever implementation.

---

[4]We will only look at the asymptotic behaviour – expressed in $O(\cdot)$ notation [26]. $f(n) = O(g(n)$ states that the function $f(n)$ is *asymptotically* bounded from above by $g(n)$.

[5]The (optional) forcing function $\boldsymbol{u}$ will, as a rule, be of lower dimension than the state vector, and therefore also $O(n^2)$.

### 4.4.2 Computational complexity of the EKF

The EKF is a linearisation of the LKF, and therefore every step in the LKF algorithm has an equivalent in the EKF. The potentially most intensive computational steps of the LKF, equations (4.1.2) and (4.1.4), are repeated in the EKF as equations (4.2.3) and (4.2.6).

One apparent difference is the way in which the state is propagated in equation (4.2.1) and the measurement is predicted in equation (4.2.4). This is done by means of the full non-linear propagation and measurement functions, of which the complexity depend fully on the problem at hand. Due to the fact that a single point is propagated and "measured" it is highly unlikely that these two equations will play a dominant role in the computational complexity of the EKF. Even numerically integrating the time derivative of the state, to compute equation (4.2.1), is unlikely to exceed $O(n^3)$ computations.

Of special interest is the cost involved in computing the system Jacobian and measurement Jacobian of equations (4.2.2) and (4.2.5). This depends wholly on the nature of the nonlinear system model and the nonlinear measurement function. Typically each element of the Jacobian can be expressed in terms of a fixed small number of terms, and does not represent a huge burden to compute. Rarely the Jacobian's elements can depend on all the elements of the state vector, which makes the computation of each element roughly $O(n)$ (provided that the cross terms are few). Therefore the computation of the system Jacobian and measurement Jacobian could conceivably be $O(n^3)$ and $O(mn^2)$ respectively. This implies that the expected asymptotic computational cost of the EKF stays $O(n^3 + m^3)$ per iteration. It is worth repeating that the cost of the computation of the Jacobian matrices can vary greatly according to the application, and the "constant" by which the asymptotic cost is multiplied will most likely play a major role, since the dimension of the state $n$ and the dimension of the measurement vector $m$ is typically small (less than, say, thirty).

### 4.4.3 Computational complexity of the UKF

Lastly we look at the computational complexity of the UKF.

The computation of the weights are of complexity $O(n)$ and only executed once during initialization. The first interesting step is to look at equation (4.3.1) in which the matrix root (factorization) of a the weighted state covariance matrix is computed. The Cholesky factorization, as noted in section §4.3, can be computed in $O(n^3)$.

The next step is to propagate each of the $2n + 1$ sigma points as shown in equation (4.3.4). The computational load of this step is, as with the EKF, totally dependent on the specific nonlinear propagation function used. The big difference is that a total of $2n + 1$ points need to be propagated instead of only one. This step is therefore much more expensive than the propagation step of the EKF (possibly even when the computation of the EKF's Jacobians is taken into account). The weighing of the sigma points in equation (4.3.5) is not much work – $O(n^2)$ to be exact.

The predicted state covariance is computed in equation (4.3.6). Manually computing the covariance of the $2n + 1$ weighted propagated sigma points is done in $O(n^3)$ computations.

The predicted measurement is computed from the propagated sigma points in equation (4.3.8). As in the EKF the complexity of this step depends on the complexity of the measurement function. Where the nonlinear weighing function is only called once per iteration of the EKF, it has to be called $n + 1$ times for the UKF. This raises the computational burden by a multiple of $n$. The weighing of the predicted measurements in $O(n^2)$.

To compute the predicted measurement covariance in equation (4.3.9) and the predicted cross covariance in equation (4.3.10) requires $O(m^2n)$ and $O(mn^2)$ computations respectively, as was also the case with the LKF.

We see that equations (4.3.11), (4.3.12) and (4.3.13) are identical to the way in which they are handled in both the LKF and the EKF, with computational costs of $O(m^3)$, $O(mn)$ and $O(mn^2 + m^2n)$ respectively.

The end result is a filter which has identical asymptotic computational complexity when compared to the LKF and EKF, namely $O(n^3 + m^3)$. It is to be noted that some potentially computationally intensive operations (like the nonlinear propagation of the state vector and nonlinear measurements) are computed much more often than with the EKF. Although dependent on the application, the UKF will most likely be a lot slower to compute than the EKF. One can refer to chapters 6 and 7 for some practical results.

# Chapter 5

# Position and Pose estimation: 2D to 3D

We now explore different ways in which to estimate the 3D position and pose of the satellite/object being tracked, when we are presented with a 2D image sequence of feature points.

The locations of the 2D features in an image depend on their original coordinates in 3D space, their relative position and motion between the camera and the target, and the camera's internal geometry. We are interested in determining the relative position and motion of the target and the camera (observer).

## 5.1 Representing the state vector

Our state vector will consist of the satellite or object's X, Y and Z translation (in the CCS), its X,Y and Z velocity (in the CCS), its pose relative to the CCS (represented as a quaternion) and its angular velocity (measured around the three axes of the OCS). The state vector therefore has 13 components, and looks as follows:

$$\begin{aligned} \boldsymbol{x} &= \begin{bmatrix} \boldsymbol{t}^T & \boldsymbol{v}^T & \boldsymbol{q}^T & \boldsymbol{\omega}^T \end{bmatrix}^T \\ &= \begin{bmatrix} X & Y & Z & v_X & v_Y & v_Z & q_1 & q_2 & q_3 & q_4 & \omega_1 & \omega_2 & \omega_3 \end{bmatrix}^T \end{aligned}$$

This representation is similar to those used in [1, 4, 25]. Since the rotation is expressed as a unit quaternion it is not ideally suited to unconstrained optimization. Methods discussed by [27, 28, 29] propose incremental representations which could provide greater stability to the Kalman Filtering approach. We chose to enforce normality by manually normalising the quaternion estimate after each update.

## 5.2 A word on the dynamic system and measurement models

The simplest system model will be to assume no external forces acting on the target, and to model only inertial movement as described in sections §2.2.1 and §2.2.2. This might be sufficient for cases in which the target's movement is unpredictable or when movement is little between successive measurements. We will also have to look at using a system model which includes Hill's orbital dynamics of section §2.3, if we want to reasonably model the oscillatory relative translation between two satellites in close circular orbits.

We will use the (nonlinear) perspective projection of section §2.1 as our measurement model, except where the "measurement" is the output of a calibration algorithm (in which case the measurement is a subset of the states to be estimated).

An aspect of Kalman filters is that (approximate) linearisations will be required due to nonlinearities in the system model and/or measurement processes. This means that the estimate at any given instant will not be an optimal statistic for all past data as in the linear case. On the other hand the inaccuracies in the system and measurement models themselves also add to uncertainty. Both these inaccuracies can conveniently be modeled as "system noise" in the Kalman Filter, even though it might not necessarily obey a strictly Gaussian distribution. It might therefore be acceptable to simplify estimation by using an inertial system model rather that a more complex orbital model.

## 5.3 Acquiring image features

We extract the 2D image coordinates of the visible tracking features every time an image becomes available. We assume that the features are easily visible and distinguishable. We might, for example, attach LEDs to carefully chosen positions on the target. We can then threshold the image and compute the centre of each feature point. In figure 5.2 we see an example of a single 2D feature point[1] (left), and the same feature point after thresholding and with its centre of intensity (weighted by brightness) indicated (right). The image was greatly enlarged so that individual pixels can clearly be seen. Note that we are able to attain accuracy to within a fraction of a pixel due to the proportional weighing of the pixels which are above the threshold.

At this time it might be worth saying something about the way in which one would compute the "centre" of image point-features. In a digital image there is always some background clutter to get rid of. In a satellite application (and for the pratical experiments in this thesis) the feature points can be significantly brighter than the background. The first step, thresholding, discards the darker background and retains the brighter feature points. The brightness level at which the background and foreground separates can be decided by from the image's histogram – as

[1] In this case a LED marker captured by means of a CCD sensor

shown in figure 5.1.

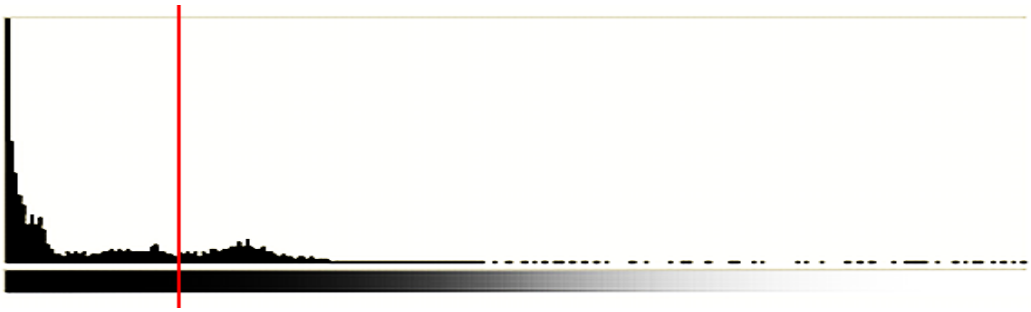


**Figure 5.1:** The histogram of an image (from section §7.2) with possible threshold level

After thresholding, we are left with only foreground (feature) pixels. Each feature would ideally correspond to a contiguous cluster of pixels, but this need not necessarily be the case. Computing the centre of such a feature can be done in a number of different ways.

- The brightest pixel value can be chosen as the "centre". This is very easy and quick to implement, but is prone to generating a centroid which jumps around from frame to frame, and is therefore not very elegant.
- The mean horizontal and vertical position of the feature pixels can be computed, without taking pixel brightness into account. This method is very simple to implement, but by not incorporating image intensity information it is sensitive to outliers.
- Computing the median of the foreground pixels' horizontal and vertical position is very robust against outliers, but computationally more expensive and not as natural or intuitive as a "centre mass" or "centre of intensity" approaches.
- The horizontal and vertical "centre of intensity" can be computed by weighing each pixel's position with its brightness. This method is still quite simple, but safe against disproportional influence by outlying pixels with low intensity, and is the method we chose for our application.

**Figure 5.2:** An example of a single 2D feature point before and after processing. Thresholded result with "centre of intesity" indicated.

## 5.4 Matching 2D features with 3D object points

It is necessary to match each 2D feature with its corresponding 3D feature for each of the tracking methods we develop in sections §5.5 and §5.6. Uniquely identifying point features from a video or image sequence is generally far from trivial. However, this task is relatively easy if the camera is able to distinguish between markers based on distinct intensity, shape or colour. For example – we can use LEDs of different colours as tracking features, and then use a colour camera to record the images. Each point should then easily be matched to its correct 3D companion based on the LED's colour. We unfortunately found that a digital sensor often has trouble in this respect. In section §7.2 the digital camera occasionally struggled to distinguish between yellow and green LEDs, especially when being viewed at an angle.

If the camera is monochrome and the feature intensities similar, we have to resort to geometrical properties of the 2D image points to match them with their corresponding 3D points. This is much harder to do since the perspective projection (described in section §2.1) distorts distances and angles between feature points. We can create clusters of features which contain different numbers of feature points – counting the number of feature points in a cluster is quite robust. Most techniques of this sort are heuristic, and deciding on which one will work best depends on the problem.

We have experimented a bit with the placement of LED markers on a target in order to identify markers based on their relative image positions (see section §7.1). Although our heuristic method worked it was not extremely robust and needed occasional corrections by hand.

## 5.5 3D Estimation using Calibration Algorithms

### 5.5.1 Concept and motivation

We are observing (measuring) the satellite/object being tracked at discrete intervals. After every interval we take a single measurement. Since we are observing point-features with an optical camera, a "measurement" refers to a set of 2D points. We wish to extract 3D information from these 2D points. It would be difficult – in fact impossible – to extract full 3D information from a single 2D measurement. It seems feasible, however, to combine a whole set of 2D measurements with prior knowledge of the object's structure, to obtain 3D information. As was explained in chapter 3 this is exactly what a calibration algorithm does.

The use of calibration algorithms has a second advantage. Because the calibration algorithm creates a "measurement" of the position and pose of the viewed object, we are directly "measuring" the elements of the state vector. This means that the "measurement function" is linear, which greatly simplifies the implementation of a Kalman filter. It will now be possible to use a LKF for the 3D translation, and to use a separate EKF or UKF with a linear measurement model, to filter the 3D rotation.

### 5.5.2 Computing position and pose with "calibration"

At this stage we have a state estimate, a state covariance matrix, and a set of 2D feature points which correspond to a known set of 3D feature points on the satellite or object.

We now use a calibration algorithm of our choice, for example Faugeras' method, to compute a position and pose estimate. (See chapter 3.) Note that the prior state estimate is not used at this stage.

### 5.5.3 LKF for estimating translation

#### 5.5.3.1 Using general inertial dynamics

If we use general inertial dynamics (relative momentum conserved), we see that linear motion along the CCS's perpendicular X, Y and Z axes are uncoupled, and we can split translation estimation into three separate estimation problems. Let's look at how the Y-axis estimate will be handled (X and Z axes will follow similarly):

Our state variable is

$$\boldsymbol{x} = \begin{bmatrix} Y \\ v_Y \end{bmatrix} = \begin{bmatrix} Y \\ \dot{Y} \end{bmatrix}.$$

We denote the Y-axis force [2] by $\boldsymbol{u}(t)$ and the mass by $M$. We can now write the state's continuous-time derivative as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{Y} \\ \ddot{Y} \end{bmatrix} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{G}\boldsymbol{u} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0 \\ \frac{1}{M} \end{bmatrix} \boldsymbol{u},$$

which (assuming $\boldsymbol{u}(t)$ is constant over the time-step $\Delta t$) gives rise[3] to the discrete-time equation

$$\bar{\boldsymbol{x}}_{k+1} = \boldsymbol{\Phi}\hat{\boldsymbol{x}}_k + \boldsymbol{\Psi}\boldsymbol{u}_k = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \hat{\boldsymbol{x}}_k + \frac{1}{M} \begin{bmatrix} \frac{1}{2}\Delta t^2 \\ \Delta t \end{bmatrix} \boldsymbol{u}_k.$$

The measurement model is very simple (since we are directly measuring[4] the translation):

$$\boldsymbol{z_{k+1}} = \boldsymbol{H}\boldsymbol{x}_{k+1} = \begin{bmatrix} 1 & 0 \end{bmatrix} \boldsymbol{x}_{k+1}$$

We clearly see that all the equations are linear in the state variables. Since the process noise consists mainly of unknown disturbance forces we can use $\boldsymbol{\Gamma} = \boldsymbol{\Psi}$. This implies that the Gaussian white noise feeds into the state variables through the forcing function, and is converted to the right coordinates by the matrix $\boldsymbol{\Psi}$.

We now use the LKF as explained in section §4.1 to combine the "measurement" with the prior state estimate, resulting in a new state estimate.

#### 5.5.3.2 Using Hill's orbital dynamics

If we choose to incorporate relative orbital movement according to Hill's equation (2.3.1) into our system model we see that linear motion along the three orthogonal axes are no longer uncoupled. We now have a single LKF for position instead of three separate ones.

Note that Hills's equations are expressed in terms of the orbit-fixed (non-inertial) frame illustrated in figure 2.6. The estimation algorithms compute the position and pose relative to the CCS. If we look at the case where the CCS is aligned with the orbital coordinate system of figure 2.6 so that the camera looks along the velocity vector, we see that the CCS's $Z$ axis corresponds with the orbital $X$ axis (tangential to the orbit), the CCS $X$ axis corresponds to the orbital $-Z$ axis, and the $Y$ axes are the same.

Therefore, in orbit-aligned CCS (as explained in the preceding paragraph), Hill's equation (2.3.1) are to be written as

$$\dot{v}_X = 2\omega_0\dot{Z} + 3\omega_0^2 X, \quad \dot{v}_Y = -\omega_0^2 Y, \quad \dot{v}_Z = -2\omega_0\dot{X}.$$

---

[2] Note that we will normally model no external forces acting on the target, and thus $\boldsymbol{u} = 0$.

[3] See section §A.2.1 for an explanation as to how this is done.

[4] The "measurement" is in fact the translation computed by the calibration algorithm.

Our state variable is

$$\boldsymbol{x} = \begin{bmatrix} X \\ Y \\ Z \\ v_X \\ v_Y \\ v_Z \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ \dot{X} \\ \dot{Y} \\ \dot{Z} \end{bmatrix}.$$

We still denote the external force[5] by $\boldsymbol{u}(t)$ and the mass by $M$. Incorporating Hill's equations we can write the state's continuous-time derivative as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \\ \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \boldsymbol{F}\boldsymbol{x} + \boldsymbol{G}\boldsymbol{u} = \begin{bmatrix} & \boldsymbol{0}_{3\times3} & & & \boldsymbol{I}_{3\times3} & \\ 3\omega_0^2 & 0 & 0 & 0 & 0 & 2\omega_0 \\ 0 & -\omega_0^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2\omega_0 & 0 & 0 \end{bmatrix} \boldsymbol{x} + \frac{1}{M} \begin{bmatrix} \boldsymbol{0}_{3\times3} \\ \boldsymbol{I}_{3\times3} \end{bmatrix} \boldsymbol{u},$$

which (assuming $\boldsymbol{u}(t)$ is constant over the time-step $\Delta t$) gives rise[6] to the discrete-time equation

$$\bar{\boldsymbol{x}}_{k+1} = \boldsymbol{\Phi}\hat{\boldsymbol{x}}_k + \boldsymbol{\Psi}\boldsymbol{u}_k = \boldsymbol{\Phi}\hat{\boldsymbol{x}}_k + \frac{1}{M} \begin{bmatrix} \frac{1}{2}\Delta t^2 \boldsymbol{I}_{3\times3} \\ \Delta t \boldsymbol{I}_{3\times3} \end{bmatrix} \boldsymbol{u}_k$$

with

$$\begin{aligned} \boldsymbol{\Phi} &= \mathrm{e}^{\boldsymbol{F}\Delta t} = \boldsymbol{I}_{6\times6} + \boldsymbol{F}\Delta t + \frac{1}{2!}\boldsymbol{F}^{\boldsymbol{2}}\Delta t^2 + \dots \\ &= \begin{bmatrix} 4-3\mathrm{c} & 0 & 0 & \frac{\mathrm{s}}{\omega_0} & 0 & \frac{2-2\mathrm{c}}{\omega_0} \\ 0 & \mathrm{c} & 0 & 0 & \frac{\mathrm{s}}{\omega_0} & 0 \\ 6\mathrm{s}-6\Delta t\omega_0 & 0 & 1 & \frac{2\mathrm{c}-2}{\omega_0} & 0 & \frac{4\mathrm{s}}{\omega_0} - 3\Delta t \\ 3\omega_0\mathrm{s} & 0 & 0 & \mathrm{c} & 0 & 2\mathrm{s} \\ 0 & -\omega_0\mathrm{s} & 0 & 0 & \mathrm{c} & 0 \\ 6\omega_0(\mathrm{c}-1) & 0 & 0 & -2\mathrm{s} & 0 & 4\mathrm{c}-3 \end{bmatrix} \end{aligned}$$

Where $\mathrm{c} \equiv \cos(\omega_0\Delta t)$, and $\mathrm{s} \equiv \sin(\omega_0\Delta t)$.

The measurement model is still very simple:

$$\boldsymbol{z}_{\boldsymbol{k+1}} = \boldsymbol{H}\boldsymbol{x}_{k+1} = \begin{bmatrix} \boldsymbol{I}_{3\times3} & \boldsymbol{0}_{3\times3} \end{bmatrix} \boldsymbol{x}_{k+1}$$

We see that all the equations are still linear in the state variables, although no longer decoupled.

---

[5] Note that we will normally model no external forces acting on the target, and thus $\boldsymbol{u} = 0$.

[6] See section §A.2.1.

We now use the LKF as explained in section §4.1 to combine the "measurement" with the prior state estimate, resulting in a new state estimate.

### 5.5.4 EKF or UKF for estimating rotation

The rotation parameters propagate in a nonlinear fashion, and therefore we cannot use a LKF to estimate this part of the state variable. This nonlinearity needs to handled by a nonlinear extension of the Kalman filter like the EKF or UKF. These algorithms are described in sections §4.2 and §4.3.

We assume that the three body axes are chosen to correspond to the three principal axes of inertia (see [9]). Thus the inertia tensor is given by

$$\boldsymbol{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}.$$

We denote the external torque around the three axes[7] by

$$\boldsymbol{T}(t) = \begin{bmatrix} T_x & T_y & T_z \end{bmatrix}^T.$$

The time derivative of the rotation state vector follows as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(t) = \begin{bmatrix} (+q_4\omega_1 - q_3\omega_2 + q_2\omega_3)/2 \\ (+q_3\omega_1 + q_4\omega_1 - q_1\omega_3)/2 \\ (-q_2\omega_1 + q_1\omega_2 + q_4\omega_3)/2 \\ (-q_1\omega_1 - q_2\omega_2 - q_3\omega_3)/2 \\ (I_{yy} - I_{zz})\omega_2\omega_3/I_{xx} \\ (I_{zz} - I_{xx})\omega_1\omega_3/I_{yy} \\ (I_{xx} - I_{yy})\omega_1\omega_2/I_{zz} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ T_x/I_{xx} \\ T_y/I_{yy} \\ T_z/I_{zz} \end{bmatrix},$$

which is clearly nonlinear in the state variables. We assume that $T_x = T_y = T_z = 0$.

---

[7] Note that we will normally model no external torque acting on the target, and thus $\boldsymbol{T}(t) = 0$.

We can now compute the system Jacobian (only necessary for an EKF) as follows:

$$\boldsymbol{F} = \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{f}(\boldsymbol{x}) = \frac{1}{2} \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 & q_4 & -q_3 & q_2 \\ -\omega_3 & 0 & \omega_1 & \omega_2 & q_3 & q_4 & -q_1 \\ \omega_2 & -\omega_1 & 0 & \omega_3 & -q_2 & q_1 & q_4 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 & -q_1 & -q_2 & -q_3 \\ 0 & 0 & 0 & 0 & 0 & \frac{2(I_{yy}-I_{zz})\omega_3}{I_{xx}} & \frac{2(I_{yy}-I_{zz})\omega_2}{I_{xx}} \\ 0 & 0 & 0 & 0 & \frac{2(-I_{xx}+I_{zz})\omega_3}{I_{yy}} & 0 & \frac{2(-I_{xx}+I_{zz})\omega_1}{I_{yy}} \\ 0 & 0 & 0 & 0 & \frac{2(I_{xx}-I_{yy})\omega_2}{I_{zz}} & \frac{2(I_{xx}-I_{yy})\omega_1}{I_{zz}} & 0 \end{bmatrix}.$$

The measurement model is linear (since we are directly measuring[8] the rotation quaternion):

$$\boldsymbol{z}_{k+1} = \boldsymbol{h}(\boldsymbol{x}) = \boldsymbol{H}\boldsymbol{x}_{k+1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \boldsymbol{x}_{k+1}$$

We can now implement either an EKF or UKF as described in section §4.2 and section §4.3.

### 5.5.5 Drawbacks

There are three obvious drawbacks to using calibration algorithms as a step in 3D estimation.

- we need to have a certain number of points in order for the calibration algorithm to work. The number of points required is discussed in sections section §3.4 and section §3.6.
- The calibration algorithm is sensitive to correct matching of 2D and 3D feature points. A mismatch will have more serious consequences if a calibration algorithm is used to generate a single "measurement", as when the points are processed separately.
- The calibration induces some extra computational load, although we will see that alternative methods of EKF filtering require the computation of more complicated Jacobians.

[8] The "measurement" is in fact the rotation computed by the calibration algorithm

## 5.6 3D Estimation without Calibration

### 5.6.1 Concept and motivation

One could argue that all the 3D information that is extracted by calibration could, in principle, be extracted by treating each 2D measurement separately. This circumvents the need for processing algorithms such as the calibration algorithms mentioned in chapter 3. Since the system and measurement models are now fully nonlinear we will need to use a single large EKF or UKF for filtering.

We use our time propagation system model to update the predicted state vector(s) between measurements sets. When a new set of 2D points is observed, we incorporate it sequentially without propagating the state vector (since the measurements in a single image arrive simultaneously).

There are two immediate advantages to using this approach

- We can update the state estimate with any number of single-point observations. The more 2D-3D matches we have the better, but even a single point match should suffice.
- An erroneous match or observation will not directly influence the accuracy obtained from the other observations. This method could therefore be expected to be more robust when the observations are corrupted by occasional outliers.
- The computational load of a calibration algorithm is avoided, even though the computation of the Jacobian matrices are more intensive.

### 5.6.2 Single EKF or UKF for estimation

We have seen than the time derivative of the full state vector can be expressed as

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \\ \dot{v}_X \\ \dot{v}_Y \\ \dot{v}_Z \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix} = \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} v_X \\ v_Y \\ v_Z \\ 0 \\ 0 \\ 0 \\ (+q_4\omega_1 - q_3\omega_2 + q_2\omega_3)/2 \\ (+q_3\omega_1 + q_4\omega_1 - q_1\omega_3)/2 \\ (-q_2\omega_1 + q_1\omega_2 + q_4\omega_3)/2 \\ (-q_1\omega_1 - q_2\omega_2 - q_3\omega_3)/2 \\ (I_{yy} - I_{zz})\omega_2\omega_3/I_{xx} \\ (I_{zz} - I_{xx})\omega_1\omega_3/I_{yy} \\ (I_{xx} - I_{yy})\omega_1\omega_2/I_{zz} \end{bmatrix}.$$

Note that (as in the previous cases) we model no external forces in the system model – the terms for handling external forces are not shown here.

As in section §5.5.3.2, if we wish to incorporate relative orbital movement with the Hills equations into our system model, we need only substitute

$$\dot{v}_X = 2\omega_0\dot{Z} + 3\omega_0^2 X, \quad \dot{v}_Y = -\omega_0^2 Y, \quad \dot{v}_Z = -2\omega_0\dot{X}.$$

We can now compute the system Jacobian for either of the two choices, depending on whether we use Hill's relative orbital equations. The results are:

$$\boldsymbol{F} = \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{F}_{trans} & \boldsymbol{0}_{3\times 7} \\ \boldsymbol{0}_{3\times 6} & \boldsymbol{0}_{3\times 7} \\ \boldsymbol{0}_{7\times 6} & \boldsymbol{F}_{rot} \end{bmatrix} \text{(Inertial model)}$$

*or*

$$\boldsymbol{F} = \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{F}_{trans} & \boldsymbol{0}_{3\times 7} \\ \boldsymbol{F}_{vel} & \boldsymbol{0}_{3\times 7} \\ \boldsymbol{0}_{7\times 6} & \boldsymbol{F}_{rot} \end{bmatrix} \text{(using Hill's equations)}$$

With

$$\boldsymbol{F}_{trans} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

and

$$\boldsymbol{F}_{vel} = \begin{bmatrix} 3\omega_0^2 & 0 & 0 & 0 & 0 & 2\omega_0 \\ 0 & -\omega_0^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2\omega_0 & 0 & 0 \end{bmatrix} \qquad \text{(Hills)}$$

and

$$\boldsymbol{F}_{rot} = \frac{1}{2}\begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 & q_4 & -q_3 & q_2 \\ -\omega_3 & 0 & \omega_1 & \omega_2 & q_3 & q_4 & -q_1 \\ \omega_2 & -\omega_1 & 0 & \omega_3 & -q_2 & q_1 & q_4 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 & -q_1 & -q_2 & -q_3 \\ 0 & 0 & 0 & 0 & 0 & \frac{2(I_{yy}-I_{zz})\omega_3}{I_{xx}} & \frac{2(I_{yy}-I_{zz})\omega_2}{I_{xx}} \\ 0 & 0 & 0 & 0 & \frac{2(-I_{xx}+I_{zz})\omega_3}{I_{yy}} & 0 & \frac{2(-I_{xx}+I_{zz})\omega_1}{I_{yy}} \\ 0 & 0 & 0 & 0 & \frac{2(I_{xx}-I_{yy})\omega_2}{I_{zz}} & \frac{2(I_{xx}-I_{yy})\omega_1}{I_{zz}} & 0 \end{bmatrix}.$$

We now have to deal with a highly nonlinear measurement function in terms of the state variable. Each 3D feature point has a known offset in the OCS, $[\ \xi \quad \psi \quad \zeta\ ]^T$, relative to the OCS origin (usually coinciding with the satellite or object's centre of mass). Since the state vector tracks the 3D position of the OCS origin, we will have to include this offset-vector in the measurement function for each feature (as a feature dependent parameter).

We define the following variables:

$$\begin{aligned}
k_1 &\Leftarrow (q_1q_2 + q_3q_4) \\
k_2 &\Leftarrow (q_1q_2 - q_3q_4) \\
k_3 &\Leftarrow (q_1q_3 + q_2q_4) \\
k_4 &\Leftarrow (q_1q_3 - q_2q_4) \\
k_5 &\Leftarrow (q_2q_3 - q_1q_4) \\
k_6 &\Leftarrow (q_2q_3 + q_1q_4) \\
k_7 &\Leftarrow (-q_1^2 - q_2^2 + q_3^2 + q_4^2) \\
k_8 &\Leftarrow (-q_1^2 + q_2^2 - q_3^2 + q_4^2) \\
k_9 &\Leftarrow (q_1^2 - q_2^2 - q_3^2 + q_4^2) \\
Z_p &\Leftarrow (Z + 2(q_1q_3 + q_2q_4)\xi + 2(q_2q_3 - q_1q_4)\psi + (-q_1^2 - q_2^2 + q_3^2 + q_4^2)\zeta).
\end{aligned}$$

For a given feature point, the measurement function, derived from equation (2.1.2), looks as follows:

$$\boldsymbol{h}(\boldsymbol{x}) = \begin{bmatrix} -f\frac{X+\xi k_9+2\psi k_1+2\zeta k_4}{Z_p} \\ \\ -f\frac{Y+2\xi k_2+\psi k_8+2\zeta k_6}{Z_p} \end{bmatrix}. \tag{5.6.1}$$

Please note that if we set the feature's OCS offset to zero, we obtain the original equations for the perspective projection as defined in equation (2.1.1).

If we want to use an EKF with this measurement equation we will have to compute the corresponding measurement Jacobian. This can be done, and yields an laborious $2 \times 13$ matrix:

$$\boldsymbol{H} = \frac{\partial}{\partial \boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}).$$

The first row of $\boldsymbol{H}$ is:

$$\boldsymbol{H}(1,:) = \begin{bmatrix} \frac{-f}{Z_p} \\ 0 \\ f\frac{X+k_9\xi+2k_1\psi+2k_4\zeta}{(Z_p)^2} \\ \\ 0 \\ 0 \\ 0 \\ \frac{f(-2(q_1\xi+q_2\psi+q_3\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)-2(-q_3\xi+q_4\psi+q_1\zeta)(X+k_9\xi+2k_1\psi+2k_4\zeta))}{(Z_p)^2} \\ \frac{f(2(q_2\xi-q_1\psi+q_4\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_4\xi+q_3\psi-q_2\zeta)(X+k_9\xi+2k_1\psi+2k_4\zeta))}{(Z_p)^2} \\ \frac{f(2(q_3\xi-q_4\psi-q_1\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_1\xi+q_2\psi+q_3\zeta)(X+k_9\xi+2k_1\psi+2k_4\zeta))}{(Z_p)^2} \\ \frac{f(-2(q_4\xi+q_3\psi-q_2\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_2\xi-q_1\psi+q_4\zeta)(X+k_9\xi+2k_1\psi+2k_4\zeta))}{(Z_p)^2} \\ \\ 0 \\ 0 \\ 0 \end{bmatrix}^T,$$

and the second row:

$$\boldsymbol{H}(2,:) = \begin{bmatrix} 0 \\ \frac{-f}{Z_p} \\ f\frac{Y+2k_2\xi+k_8\psi+2k_6\zeta}{(Z_p)^2} \\ \\ 0 \\ 0 \\ 0 \\ \\ \frac{f(-2(q_2\xi+q_1\psi+q_4\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)-2(-q_3\xi+q_4\psi+q_1\zeta)(2k_2\xi+Y+k_8\psi+2k_5\zeta))}{(Z_p)^2} \\ \\ \frac{f(-2(q_1\xi+q_2\psi+q_3\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_4\xi+q_3\psi-q_2\zeta)(2k_2\xi+Y+k_8\psi+2k_6\zeta))}{(Z_p)^2} \\ \\ \frac{f(2(q_4\xi+q_3\psi-q_2\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_1\xi+q_2\psi+q_3\zeta)(2k_2\xi+Y+k_8\psi+2k_6\zeta))}{(Z_p)^2} \\ \\ \frac{f(2(q_3\xi-q_4\psi-q_1\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_2\xi-q_1\psi+q_4\zeta)(2k_2\xi+Y+k_8\psi+2k_6\zeta))}{(Z_p)^2} \\ \\ 0 \\ 0 \\ 0 \end{bmatrix}^T,$$

We can now implement either an EKF or UKF as described in section §4.2 and section §4.3.

The matrix $\boldsymbol{H}$ is not pretty to behold, but it should be obvious that some terms are repeated, and thus need only be computed once (per iteration). Approximations could also be made to lower the computational cost. The size of the covariance matrix $\boldsymbol{F}$ might be a drawback, and an attempt is made at avoiding this problem with the next approach.

### 5.6.3 Split EKF or UKF for estimation

A split EKF or UKF for estimating the full 3D state vector from 2D observations relies on the following idea:

1. Assume that the rotation is constant (take the most recent estimate's value) and determine a new estimate for the translation.
2. Assume that the translation is constant (take the most recent estimate's value) and determine a new estimate for the rotation.

The idea of this method is that the combined sizes of the new covariance matrices are smaller than the complete covariance matrix ($6 \times 6$ and $7 \times 7$ as opposed to $13 \times 13$). This not only

saves memory, but could also lead to a more robust algorithm. The drawback is that we make rash assumptions during the measurement updates (assuming first *only* translation and then *only* rotation).

We have seen than the time derivatives of the two halves of the state vector can be expressed as

$$\dot{\boldsymbol{x}}_t = \begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{Z} \\ \dot{v}_X \\ \dot{v}_Y \\ \dot{v}_Z \end{bmatrix} = \boldsymbol{f}_t(\boldsymbol{x}) = \begin{bmatrix} v_X \\ v_Y \\ v_Z \\ 0 \\ 0 \\ 0 \end{bmatrix},$$

$$\dot{\boldsymbol{x}}_r = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \\ \dot{q}_4 \\ \dot{\omega}_1 \\ \dot{\omega}_2 \\ \dot{\omega}_3 \end{bmatrix} = \boldsymbol{f}_r(\boldsymbol{x}) = \begin{bmatrix} (+q_4\omega_1 - q_3\omega_2 + q_2\omega_3)/2 \\ (+q_3\omega_1 + q_4\omega_1 - q_1\omega_3)/2 \\ (-q_2\omega_1 + q_1\omega_2 + q_4\omega_3)/2 \\ (-q_1\omega_1 - q_2\omega_2 - q_3\omega_3)/2 \\ (I_{yy} - I_{zz})\omega_2\omega_3 / I_{xx} \\ (I_{zz} - I_{xx})\omega_1\omega_3 / I_{yy} \\ (I_{xx} - I_{yy})\omega_1\omega_2 / I_{zz} \end{bmatrix}.$$

Note that (as in the previous cases) we model no external forces in the system model – the terms for handling external forces are not shown here.

The system Jacobians (similar to the combined filter) are:

$$\boldsymbol{F}_t = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 3\omega_0^2 & 0 & 0 & 0 & 0 & 2\omega_0 \\ 0 & -\omega_0^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -2\omega_0 & 0 & 0 \end{bmatrix} \qquad \text{(Hills)}$$

and

$$\boldsymbol{F}_r = \frac{1}{2}\begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 & q_4 & -q_3 & q_2 \\ -\omega_3 & 0 & \omega_1 & \omega_2 & q_3 & q_4 & -q_1 \\ \omega_2 & -\omega_1 & 0 & \omega_3 & -q_2 & q_1 & q_4 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 & -q_1 & -q_2 & -q_3 \\ 0 & 0 & 0 & 0 & 0 & \frac{2(I_{yy}-I_{zz})\omega_3}{I_{xx}} & \frac{2(I_{yy}-I_{zz})\omega_2}{I_{xx}} \\ 0 & 0 & 0 & 0 & \frac{2(-I_{xx}+I_{zz})\omega_3}{I_{yy}} & 0 & \frac{2(-I_{xx}+I_{zz})\omega_1}{I_{yy}} \\ 0 & 0 & 0 & 0 & \frac{2(I_{xx}-I_{yy})\omega_2}{I_{zz}} & \frac{2(I_{xx}-I_{yy})\omega_1}{I_{zz}} & 0 \end{bmatrix}.$$

Up to this stage the filters for translation and rotation are wholly uncoupled. It is with the measurement function, however, where they can no longer be treated totally separate. The measurement function is identical to the one used for the combined filter of section §5.6.2.

We define $k_1, \ldots, k_9$ and $Z_p$ exactly as in section §5.6.2. The Jacobian of the first measurement function is:

$$\boldsymbol{H}_t = \begin{bmatrix} \frac{-f}{Z_p} & 0 \\ 0 & \frac{-f}{Z_p} \\ f\frac{X+k_9\xi+2k_1\psi+2k_4\zeta}{(Z_p)^2} & f\frac{Y+2k_2\xi+k_8\psi+2k_6\zeta}{(Z_p)^2} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}^T.$$

The second measurement function's Jacobian looks as follows:

$$\boldsymbol{H}_r(1,:) = \begin{bmatrix} \frac{f(-2(q_1\xi+q_2\psi+q_3\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)-2(-q_3\xi+q_4\psi+q_1\zeta)(X+k_9\xi+2k_1\psi+2k_4\zeta))}{(Z_p)^2} \\ \frac{f(2(q_2\xi-q_1\psi+q_4\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_4\xi+q_3\psi-q_2\zeta)(X+k_9\xi+2k_1\psi+2k_4\zeta))}{(Z_p)^2} \\ \frac{f(2(q_3\xi-q_4\psi-q_1\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_1\xi+q_2\psi+q_3\zeta)(X+k_9\xi+2k_1\psi+2k_4\zeta))}{(Z_p)^2} \\ \frac{f(-2(q_4\xi+q_3\psi-q_2\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_2\xi-q_1\psi+q_4\zeta)(X+k_9\xi+2k_1\psi+2k_4\zeta))}{(Z_p)^2} \\ 0 \\ 0 \\ 0 \end{bmatrix}^T,$$

$$\boldsymbol{H}_r(2,:) = \begin{bmatrix} \frac{f(-2(q_2\xi+q_1\psi+q_4\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)-2(-q_3\xi+q_4\psi+q_1\zeta)(2k_2\xi+Y+k_8\psi+2k_5\zeta))}{(Z_p)^2} \\ \frac{f(-2(q_1\xi+q_2\psi+q_3\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_4\xi+q_3\psi-q_2\zeta)(2k_2\xi+Y+k_8\psi+2k_6\zeta))}{(Z_p)^2} \\ \frac{f(2(q_4\xi+q_3\psi-q_2\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_1\xi+q_2\psi+q_3\zeta)(2k_2\xi+Y+k_8\psi+2k_6\zeta))}{(Z_p)^2} \\ \frac{f(2(q_3\xi-q_4\psi-q_1\zeta)(2k_3\xi+2k_5\psi+Z+k_7\zeta)+2(q_2\xi-q_1\psi+q_4\zeta)(2k_2\xi+Y+k_8\psi+2k_6\zeta))}{(Z_p)^2} \\ 0 \\ 0 \\ 0 \end{bmatrix}^T.$$

Note that both measurement Jacobians refer to the same measurement, and that both Jacobians contain terms contained in both state vectors. We resolve this in the following manner:

1. Use the most recent estimate of the rotation to compute the measurement Jacobian for the translation's Kalman Filter. The rotation's state vector is thus treated as a constant during

this step.

2. Use the updated estimate of the translation to compute the measurement Jacobian for the rotation's Kalman Filter. The translation's state vector is thus treated as a constant during this step.

   We can now implement either EKF or UKF filters as described in sections §4.2 and §4.3.

### 5.6.4 Separate Simultaneous vs. Concatenated measurements

At each measurement instant (when an image of the target becomes available) we are presented with a number of 2D feature points.[9] Calibration algorithms (chapter 3) use this set of points to compute a position and pose as "measurement". We have, however, discussed approaches (section §5.6) in which the 2D points are directly interpreted as measurements.

If a Kalman filter is used to process a set of $n$ 2D points (measurements) at time $t$, we first have to propagate the state and covariance estimates from time $t-1$. This yields the prior estimates at time $t$. We now have two logical options:

- Update the estimate sequentially with each of the $n$ measurements – finally obtaining the posterior estimate at time $t$. Or...
- Concatenate the $n$ measurements to form a single super-measurement. Use this measurement to update the estimate in a single step, to yield the posterior estimate at time $t$.

Both methods process all the available data. Intuition does not give a clear answer to which method will work best, although the following argument favours the latter:

If the dimension of the state vector is larger than the dimension of the measurement vector, we are effectively updating more variables than we are measuring. Although the Kalman filter's equations model the dependencies between variables, we can intuitively expect some problems. By concatenating the measurements, we are reducing the ratio of state dimension to measurement dimension, which should benefit performance.

However, Kelly [30] claims that in situations where the errors in individual measurements are uncorrelated[10], it can be shown that processing them one at a time gives the same result as processing them as one large block. That is, the measurement matrix can be reduced to individual rows or any logical group of sub matrices, and presented to the filter as such. Thus, the software complication involved in restructuring the matrices to accommodate presence or absence of measurements can be completely avoided.

---

[9] The number of visible points will typically fluctuate with time

[10] which is not necessarily the case here

### 5.6.5 Concatenating a set of measurements

We will now give an example showing how a set of 2D feature points can be concatenated to form a single measurement, and how the Kalman filter is adapted accordingly. We use the Single EKF and UKF, as discussed in section §5.6.2, to illustrate our example.

Note that only the equations dealing with the measurements are affected. We will therefore see a change in the measurement function $\boldsymbol{h}(\boldsymbol{x})$ from equation (5.6.1), as well as in the measurement Jacobian matrix $\boldsymbol{H}$ (in the case of the EKF).

The measurement function is redefined to generate all the visible feature points as a single vector with length $2N$, where $N$ is the number of visible feature points.

$$\boldsymbol{h}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{h}_1(\boldsymbol{x}) \\ \boldsymbol{h}_2(\boldsymbol{x}) \\ \vdots \\ \boldsymbol{h}_N(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ x_2 \\ y_2 \\ \vdots \\ x_N \\ y_N \end{bmatrix} = \begin{bmatrix} -f\frac{X+\xi_1 k_9+2\psi_1 k_1+2\zeta_1 k_4}{Z_1} \\ -f\frac{Y+2\xi_1 k_2+\psi_1 k_8+2\zeta_1 k_6}{Z_1} \\ -f\frac{X+\xi_2 k_9+2\psi_2 k_1+2\zeta_2 k_4}{Z_2} \\ -f\frac{Y+2\xi_2 k_2+\psi_2 k_8+2\zeta_2 k_6}{Z_2} \\ \vdots \\ -f\frac{X+\xi_N k_9+2\psi_N k_1+2\zeta_N k_4}{Z_N} \\ -f\frac{Y+2\xi_N k_2+\psi_N k_8+2\zeta_N k_6}{Z_N} \end{bmatrix}. \tag{5.6.2}$$

We see from equation (5.6.2) that the concatenated measurement function is exactly that – the concatenation of the same computation, with only the coordinates of the corresponding 3D OCS point updated each time. The subscripts for the variables $\xi$, $\psi$, $\zeta$ and $Z$ indicate for which 3D OCS point they are evaluated.

If we define

$$\boldsymbol{H}_i = \frac{\partial}{\partial \boldsymbol{x}} \boldsymbol{h}_i(\boldsymbol{x})$$

we can write the measurement Jacobian for the EKF as follows:

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{H}_1 \\ \boldsymbol{H}_2 \\ \vdots \\ \boldsymbol{H}_N \end{bmatrix}.$$

The structure of each of the submatrices $\boldsymbol{H}_i$ is identical to $\boldsymbol{H}$ in section §5.6.2, with the appropriate values for $\xi$, $\psi$, $\zeta$ and $Z$ (according to the subscript index $i$). Note that in this case the subscript $i$ refers to the index of the 3D OCS point, and not to the time step.

We will show in chapters 6 and 7 that concatenating measurements, as explained above, provides considerable advantages in accuracy and stability of the tracking algorithm.

# Chapter 6

# Tracking a Simulated Target

The easiest way in which we can create an image sequence with known properties, corresponding to 3D movement with a known ground truth, is by simulating. An imaginary satellite was therefore created for simulating sets of observations to test our algorithms on. A sketch of the imaginary satellite with marker positions is shown in figure 6.1. One marker point was "placed" on each corner of the body, one halfway along the length of the boom and the last one on the boom's tip.

The physical dimensions of the satellite were chosen to represent generally realistic values for a small satellite. The satellite was chosen with a mass of 50 kg, a diagonal inertia matrix with diagonal $[50 \;\; 50 \;\; 20]\text{kg} \cdot \text{m}^2$ (the lower value is for the axis co-axial with the boom).

The camera viewing the satellite was simulated having an effective focal length of -1000 units (refer to section §2.1). This implies that, if viewed head-on from a distance of 30 m, one of the satellite's $1.5 \times 1.5$m body panels would occupy a square $50 \times 50$ pixels. If we assume a sensor of $1000 \times 1000$ pixels, it implies that the camera has a field of view of about 53.1°, which is actually very close to the field of view of a 36mm (expressed as 35mm film equivalent) lens. This exactly the field of view found on most consumer digital cameras (at wide angle).

The satellite was placed in a circular orbit with an orbital period of 90 minutes. This corresponds to a height of roughly 859 km above sea level, which is also typical for small imaging satellites.

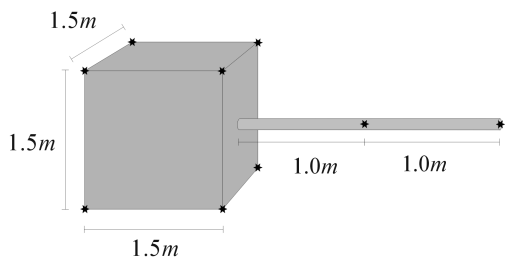



**Figure 6.1:** The simulated satellite (marker points indicated)

We assume that both the observing satellite and the target are in near-identical circular orbits, and that Hill's equations (see section §2.3) are therefore valid for describing their relative motion. We assume that the camera's coordinate system (CCS) is aligned with the conventional orbital coordinate system (figure 2.6) in the sense that the CCS's Z-axis (figure 2.2) corresponds the the orbital X-axis, and the CCS's X-axis corresponds to the orbital negative Z axis. The Y axes are identical between the two coordinate systems.

We chose the initial displacement of the target satellite relative to the observer (camera) to be $[X,\ Y,\ Z] = [0.37,\ -1.40,\ 30.14]$m. The relative initial velocity was chosen so that the Hills equations predict periodic movement, in this case $[v_x,\ v_y,\ v_z] = [0.0015,\ 0.0023,\ -0.0009]$m/s. The initial rotation of the target was chosen at random, and corresponds to the Euler axis and angle rotation of $33.9^\circ$ around the unit vector $[-0.202\ -0.254\ \ 0.946]$. Note that all these vectors are expressed in the CCS. The angular velocity was initially chosen as $-0.017$deg/s, purely around the body $X$-axis. The simulated data set (soon to be estimated from the simulated visual observations) is shown in figures 6.2, 6.3, 6.4 and 6.5. It is clear that the quaternion elements, as described in section §D.4, follow smooth trajectories during the maneuver; this is precisely why we chose this (less intuitive) way of expressive rotations.

We decided on observing the target every 30 seconds (of the simulated orbit). The simulated satellite has 10 available markers placed on its body (figure 6.1). We simulated occlusion of the markers by randomly removing two of the eight body-mounted visible markers (excluding those of the satellite's boom) every time a image was "taken". Two-dimensional noise with a uniform distribution of 0.5 pixels was added to each of the marker points' 2D positions in the "image". Real-world spatial image noise from a digital sensor is most likely caused by digitization. Broida et al. [1] notes that digitization noise is represented more accurately by a uniform rather than Gaussian distribution, even though the Kalman Filter assumes noise to be of a Gaussian nature.

The translational dynamics for the satellite was simulated as obeying Hill's orbital equations, which results in periodic and sinusoidal variations in the $X$, $Y$ and $Z$ axes. As stated earlier we chose the initial conditions so that the relative orbits will not diverge with time. During the satellite's orbit the rotation was chosen to correspond to a hypothetical sequence of orientation demands. We let the satellite experience sinusoidal angular velocity changes about it's body X-axis, while the angular rate about the $X$ and $Y$ axes were kept constant, with short episodes of linear acceleration followed by linear deceleration. Although hypothetical, this corresponds well with the demands on, say, an imaging satellite. The movement sequence was identical for each tested algorithm, so that the algorithms could be compared fairly.

Each simulation was run for 500 time steps. This corresponds to 15000 seconds, or in other words, 2.78 orbits.

At first the algorithms described in chapter 5 were tested without giving the Kalman Filters knowledge about Hill's equations in their system models. The filters therefore expect general inertial rigid body dynamics by means of their system model. The deviations observed from

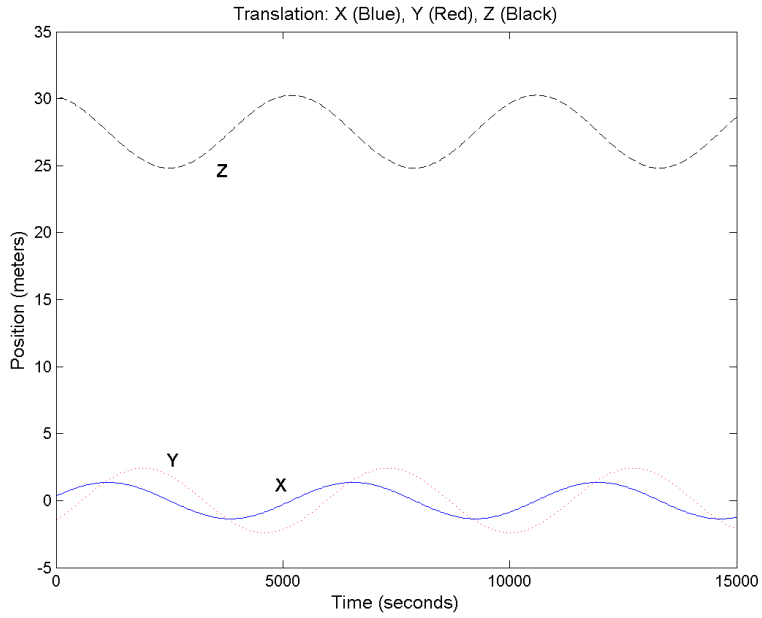



**Figure 6.2:** The simulated $X$,$Y$ and $Z$ trajectories of the target satellite according to Hill's equations (expressed in the CCS)

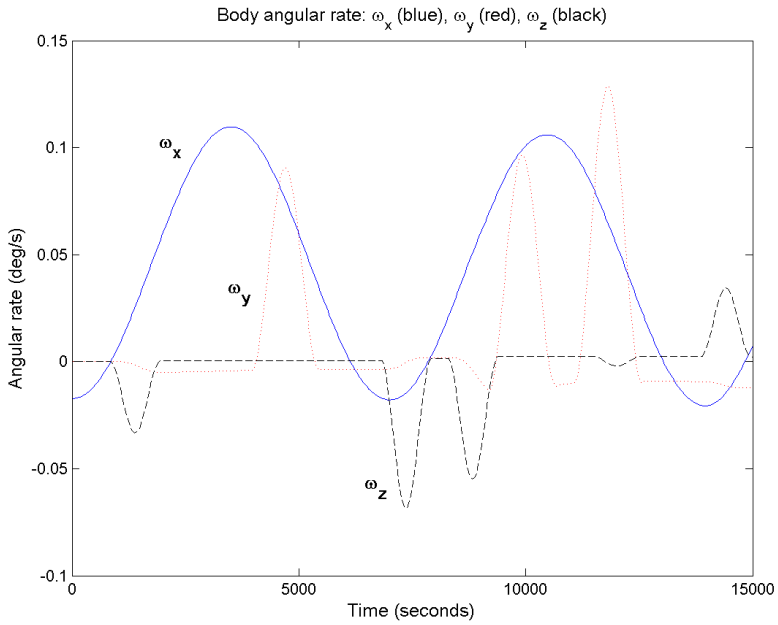



**Figure 6.3:** The simulated body angular rates of the target satellite, over the simulated interval

the filter's predicted trajectory is therefore attributed to "system noise" After testing each of the methods we ran the tests again – this time with Hill's equations built into the system model. Since no prior knowledge of the target's rotation sequence was given to the Kalman Filter, it

still needs to process all deviations from the inertial model as state noise, or rather uncertainty.

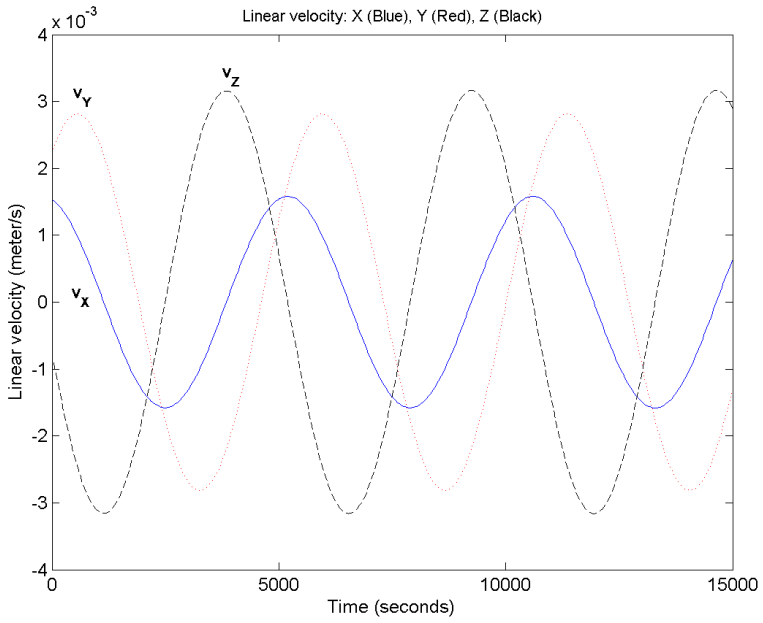



**Figure 6.4:** The $X$,$Y$ and $Z$ velocities corresponding to Hill's equations and figure 6.2

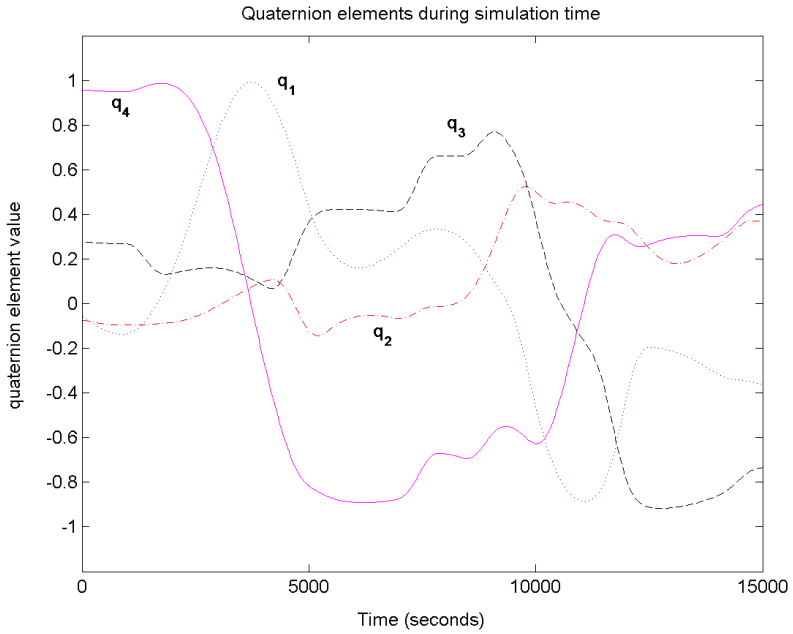



**Figure 6.5:** The quaternion elements corresponding to the angular rate demands of figure 6.3

For the sake of brevity not all the results are shown graphically. Some representative tracking results are shown in full, and all results are afterwards summarised in tabular form.

## 6.1 How the tracking accuracy is computed

To compare each of the tracking algorithms we needed a sensible way of measuring the tracking error. The error in 3D position and linear velocity, at a given instant, can easily be computed as the vector difference between the true and estimated values. We graph the ground truth, together with the estimate, for the $X$, $Y$ and $Z$ translation components. The error is afterwards expressed as the RMS difference between the ground truth and estimate

$$\begin{aligned} e_{trans} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left\| [X\ Y\ Z]_i - [\hat{X}\ \hat{Y}\ \hat{Z}]_i \right\|^2} \\ e_{vel} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left\| [v_X\ v_Y\ v_Z]_i - [\hat{v}_X\ \hat{v}_Y\ \hat{v}_Z]_i \right\|^2} \end{aligned}$$

For representing rotation errors the situation is a little different. Some authors [1, 3, 4] prefer to also measure the rotation error as an absolute vector difference error. While measuring the state vector error directly, a quaternion error vector cannot be directly interpreted in physical terms; the size of this difference vector is not even linear in the angle by which the two rotations differ. Refer to section §D.4 for an explanation of the quaternion attitude representation. We choose to measure rotational error in terms of the smallest angle needed to rotate from the estimated rotation to the ground truth's value. This can quite easily be computed from the quaternion representation. Given the ground truth $\boldsymbol{q}$ and estimate $\hat{\boldsymbol{q}}$ (both as quaternions), we can compute the "error" quaternion which rotates $\boldsymbol{q}$ to $\hat{\boldsymbol{q}}$ as $\boldsymbol{q}_{error} = \text{conj}(\boldsymbol{q}) \otimes \hat{\boldsymbol{q}}$. The magnitude of this error (as an angle) can now be easily computed as $\theta = 2 \arccos q_{e4}$, where $q_{e4}$ is the fourth (scalar) element of $\boldsymbol{q}_{error}$. The RMS error for rotation is thus expressed as

$$e_{rot} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \theta_i^2}$$

The state vector expresses angular velocity as rates around the satellite's body axes. We compute the error in the same way as with the linear velocity – as a vector difference.

$$e_{rotvel} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} \left\| [\omega_1\ \omega_2\ \omega_3]_i - [\hat{\omega}_1\ \hat{\omega}_2\ \hat{\omega}_3]_i \right\|^2}$$

In each case the filter might take a little while to converge from its initialisation values to a stable estimate. The error of the estimates was computed only after reasonable convergence was obtained. In practice this meant computing the error over the last 375 frames, which corresponds to three quarters of the total number of frames. Each algorithm was run ten times on the same simulated sequence (with the noise being computed randomly with every trial) and the average RMS error values computed from these results.

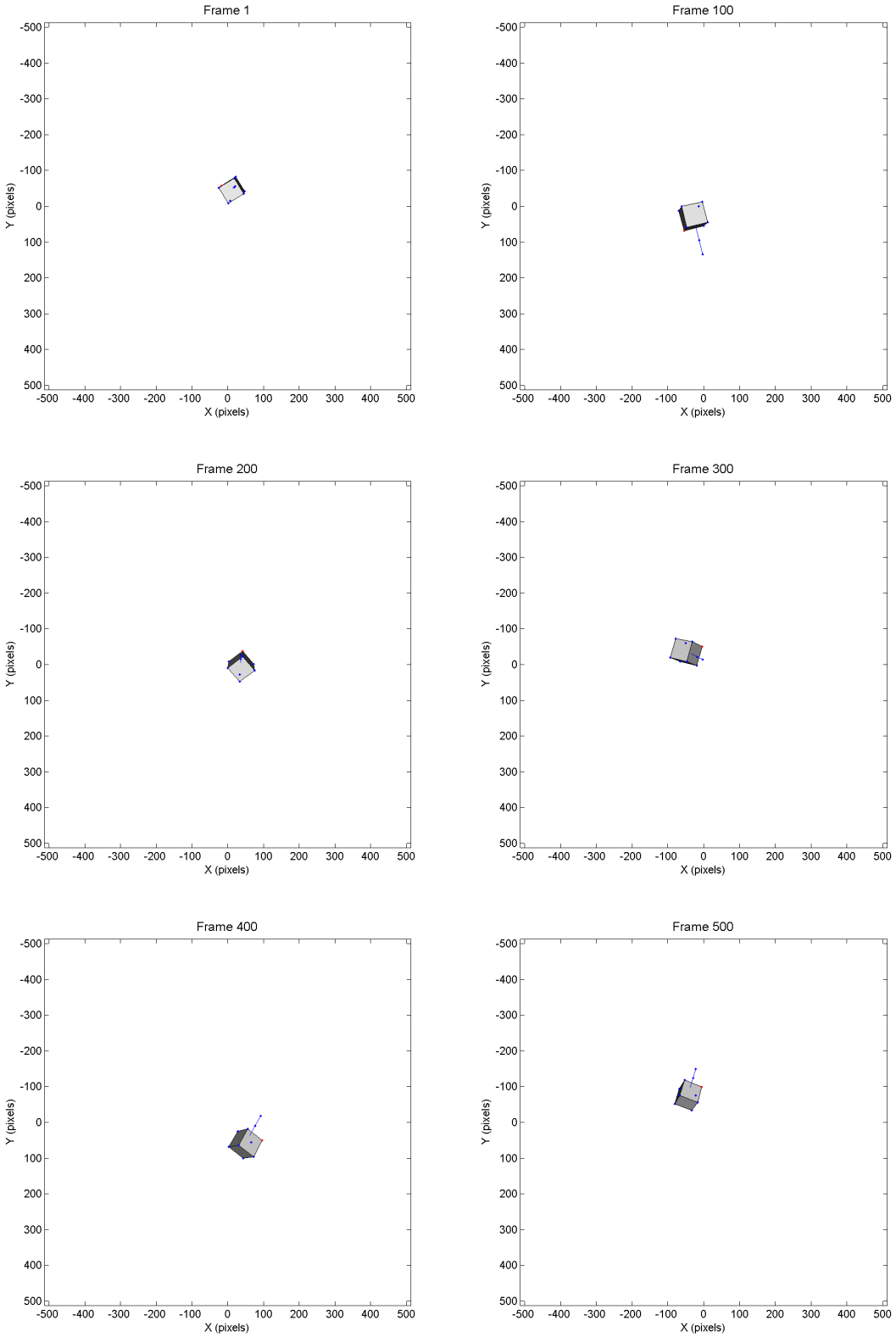



**Figure 6.6:** Every hundredth frame of the simulated image sequence (all features shown)

## 6.2 Explanation of the different approaches

As explained earlier in sections §5.5 and §5.6, there are two fundamentally different ways in which we can try to track the target. The first approach relies on computing a translational position and rotation from the feature points of a each separate image frame. The output generated by this so-called calibration algorithm is then filtered by a combination of LKF(s) for the linear and possibly uncoupled[1] translational dynamics, and an EKF or UKF for the nonlinear rotational dynamics. The second approach sidesteps the need for a calibration algorithm by processing the 2D image points directly as measurements. The increased nonlinearity of this model requires a larger and more complex EKF or UKF in which no part of the state vector is decoupled.

For each of the two basic approaches we have the choice of using either an EKF or UKF. We are specifically interested in the performance of the EKF relative to the UKF, in the light that the UKF is a recent invention, and has become quite popular in recent publications such as [4, 5, 23, 24] since being introduced by Julier and Uhlmann [21] in 1996.

For each of the combinations of the above we have the option of testing the different conceptual approaches discussed in sections §5.6.2, §5.6.3 and §5.6.4 (such as whether or not to concatenate 2D feature matches into a single measurement).

The large number of possible combinations to test has a prohibitive effect on the inclusion of graphs of their performance. We will try to show a few representative graphs, and summarise the complete set of results in tables 6.1, 6.2 and 6.3. A more complete set of graphical results and setup information can be found in appendix E.

### 6.2.1 Tracking using a calibration algorithm

The first approach was to use a calibration algorithm (in this case the direct method of section §3.2) to compute an estimate of the satellite's full state vector, and then to filter these results by means of three uncoupled LKF filters for the translation, and an EKF or UKF for rotation (refer to section §5.5). Representative results for "measurements" of the translation and rotation, by the calibration algorithm, are shown in figure 6.7.

### 6.2.2 Using purely an EKF or UKF with points treated separately

Due to the extremely little information contained in a single 2D image feature point, this method was not found to be stable. It can be considered generally bad practice to update a state vector of high dimension by separate measurements of low dimension, specifically when the measurements are related to the state vector by a complex nonlinear relation.

[1] when we are not using Hill's equations in the system model

### 6.2.3 Using purely an EKF or UKF with points treated together

By still using a single EKF, but now concatenating all the 2D features into a single measurement, we solve the dimensional discrepancy between the state vector and the measurement vector. This approach does in fact work excellently well, as shown in figure 6.8.

### 6.2.4 Split EKF with point features treated together

Of the methods looked at so far it is clear that the single EKF filter of section §5.6.2, which treats all the feature points of a single frame as a single measurement, is by far the most effective. A related idea mentioned in section §5.6.3 is to split the EKF into two smaller EKF filters which separately estimate the translation and rotation. This implementation cuts the size of the covariance matrices from $13 \times 13$ to $6 \times 6$ and $7 \times 7$.

### 6.2.5 Single EKF with point features treated together - reconstructing missing points

The last test run on this data set consists of replacing the features which are not visible by their most recent Kalman Filter estimates. As explained earlier we marked two of the feature points "invisible" (occluded) in every image frame. We can now create fake observations for these occluded feature points by generating "observations" of their estimated positions. This carries the risk of inflating the estimation error with positive feedback, but has the advantage of generating a larger set of measurements which might have a positive effect on the stability of the estimator.

## 6.3 Inertial model with features matched correctly

Our simulation is of the target satellite of figure 6.1 being observed from the observer satellite (which has the camera mounted to its body). We stipulated that the orbit has a period of 90 minutes, and gave the initial separation. The simulation is built upon Hill's orbital equations of section §2.3. We will first, for simplicity, not model Hill's equations in our Kalman filter's system model. We will assume only inertial rigid body dynamics as discussed in chapter 5. The sinusoidal translational oscillations shown in figure 6.2 (due to Hill's orbital dynamics), as well as the rotational dynamics shown in figures 6.5 and 6.3, are therefore treated as uncertainty in the state transition (i.e. system noise). Two of the ten features are randomly occluded in every image frame, but all the remaining features are correctly matched to their corresponding 3D positions.

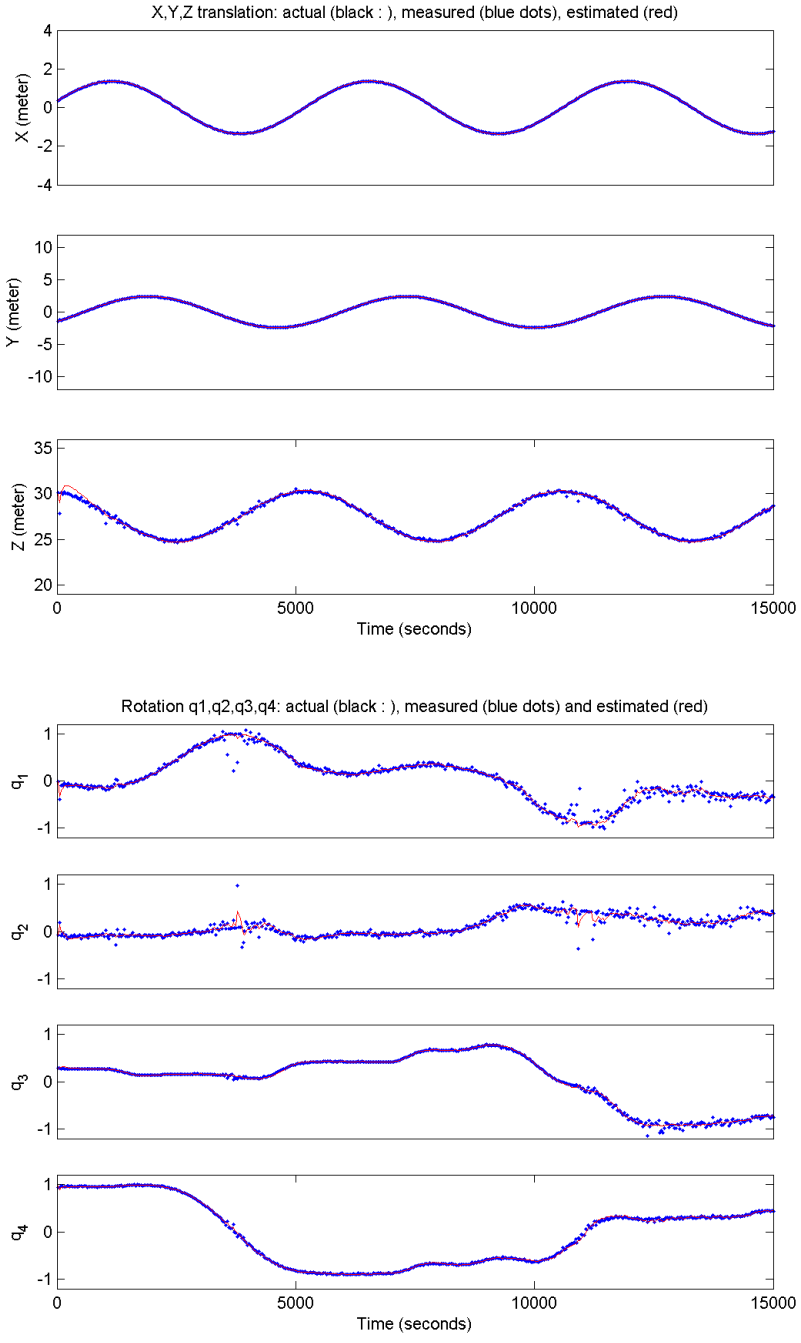



**Figure 6.7:** Simulated: Calibration algorithm and EKF - translation and rotation

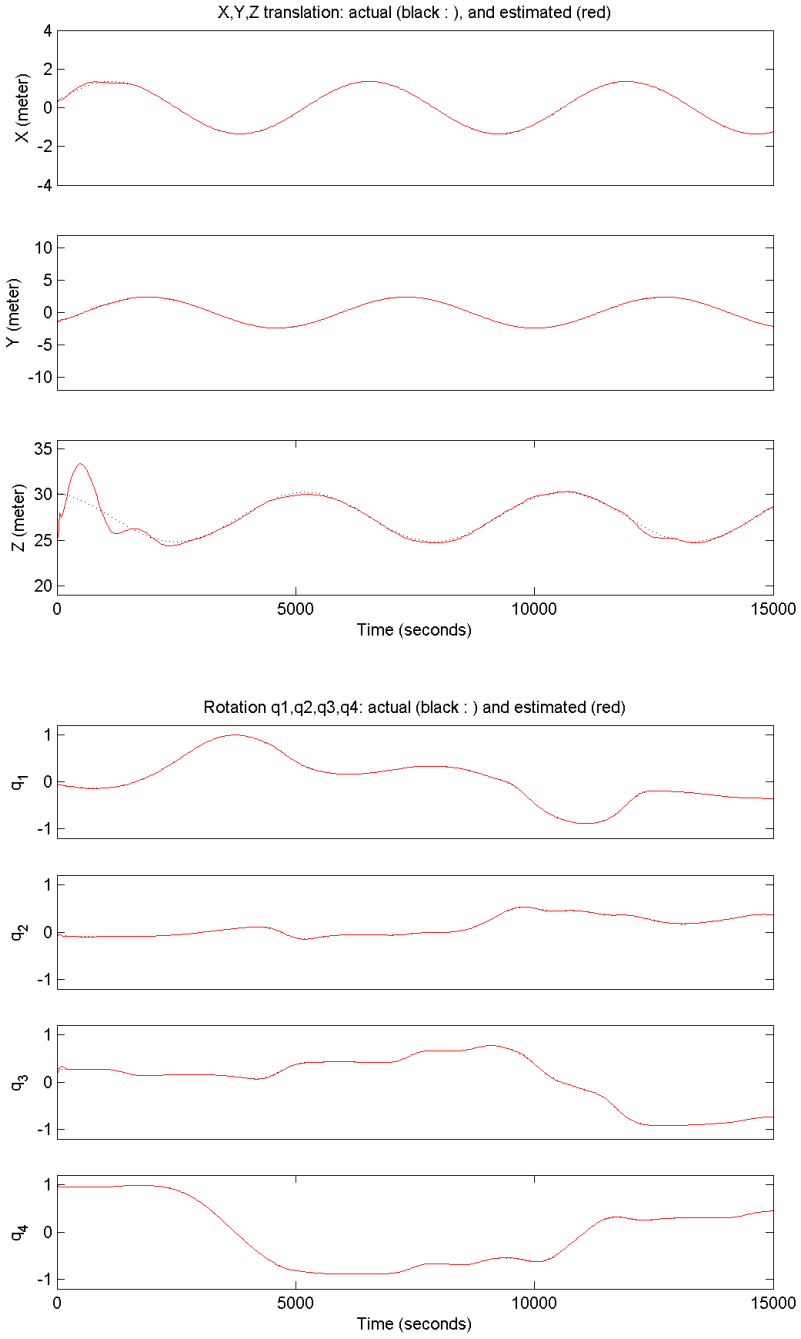



**Figure 6.8:** Simulated: Single UKF with features treated together - translation and rotation

## 6.4 Inertial model with errors in feature matching

Up to this point we assumed that each of the 2D feature points in the image can be successfully identified and matched with its 3D counterpart on the target's body. We will now simulate erroneously matched points (in addition to occlusion), to see whether the tracking algorithms

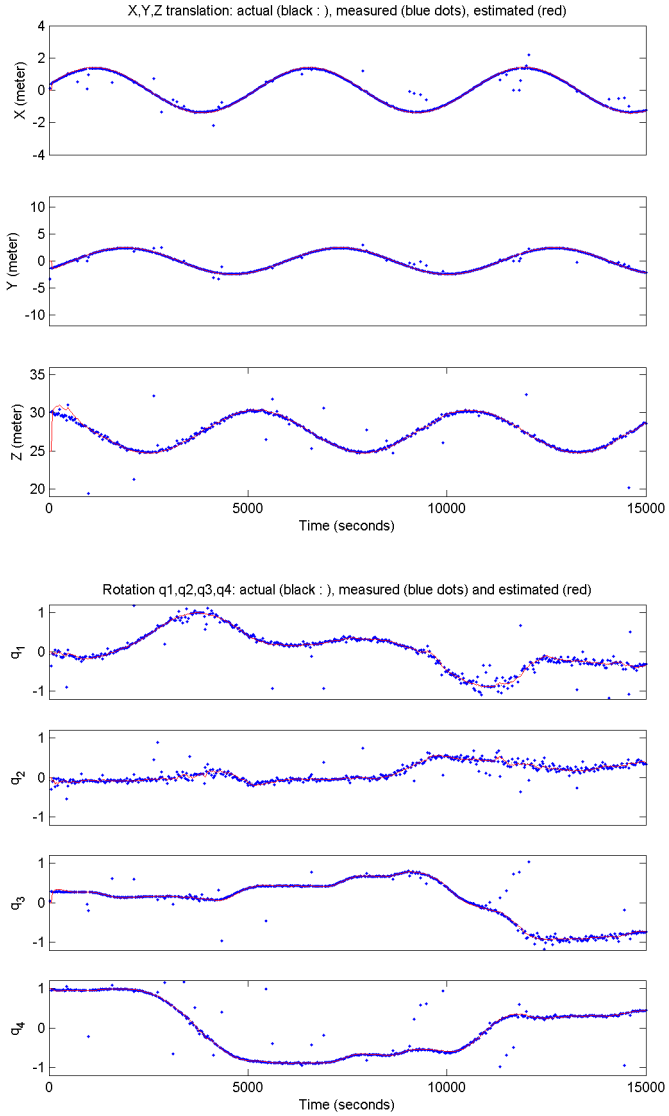



**Figure 6.9:** Simulated: Calibration algorithm and EKF with faulty feature matches - translation and rotation

are able to cope with this additional instability. A single erroneous match[2] was designed to occur with a 10% probability at every image frame. Thus, on average, the tracking algorithm had to deal with a false match every tenth frame.

As we can see in figure 6.9 the faulty matches produce many spurious "calibration" results, but this can be compensated for by ignoring measurements which deviate strongly from the state estimate (as was done in this case).

[2]meaning that a 2D image point is matched with another (randomly selected) 3D feature position on the target's body.

## 6.5 Orbital model with features matched correctly

In the previous two sets of simulated tracking runs we used only an inertial rigid body model to represent the target satellite's motion, although the target actually obeyed Hill's equations. We now extend the tracking algorithms' system model to include Hill's equations, as described in chapter 5. This should no doubt increase the accuracy of the linear motion prediction, and therefore of the translational estimates. The results, shown in figure 6.10, confirm this.

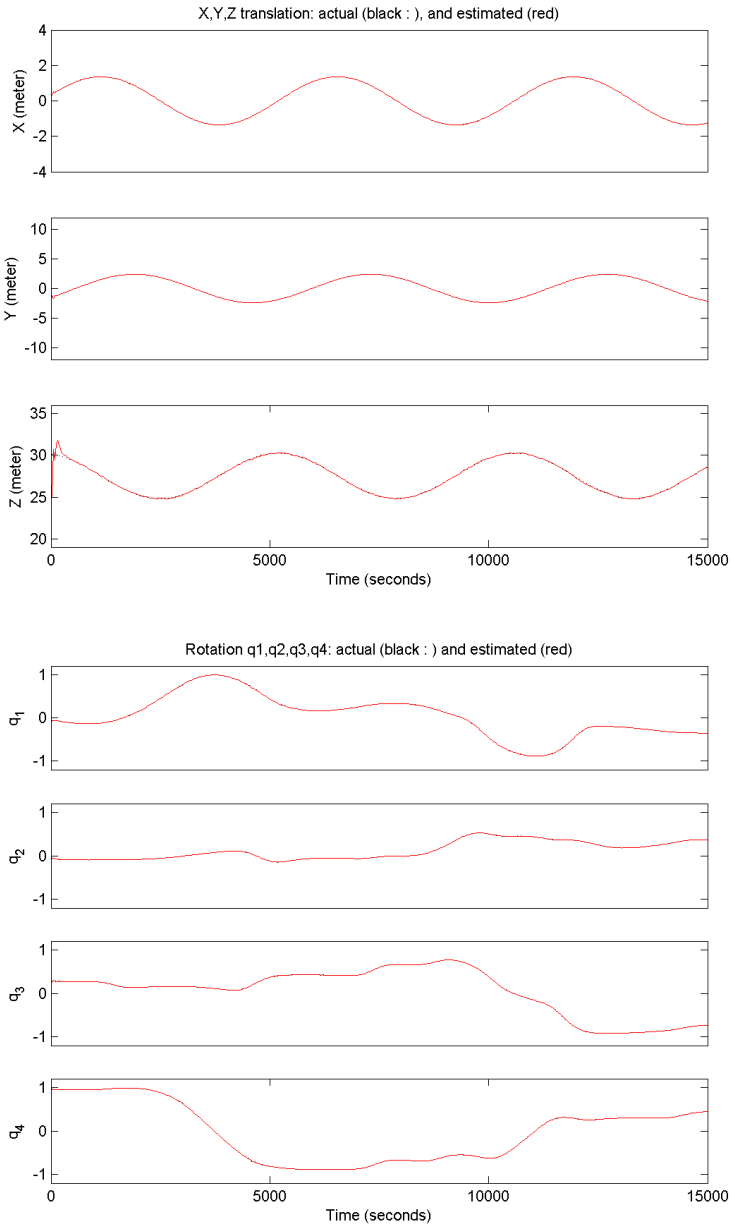



**Figure 6.10:** Simulated: Single EKF incorporating Hill's equations, with features treated together - translation and rotation

## 6.6 Results and Summary

The average tracking accuracy[3] for the different approaches are summarised in tables 6.1, 6.2 and 6.3. The last column shows the average time, $\tau$, the simulation took to complete on the author's computer. The system specifications are given in section §F.3.

| Method used | RMS error | | | | time |
|---|---|---|---|---|---|
| | translation (m) | velocity (m/s) | rotation (deg) | angular rate (deg/s) | $\tau$ (sec) |
| Cal. & LKF+EKF | 0.0908 | 0.0007 | 5.46 | 0.0374 | 3.56 |
| Cal. & LKF+UKF | 0.0893 | 0.0007 | 6.50 | 0.0403 | 17.3 |
| EKF (points separate) | - | - | - | - | - |
| UKF (points separate) | - | - | - | - | - |
| EKF (points combined) | 0.0384 | 0.0004 | 0.403 | 0.0105 | 2.81 |
| UKF (points combined) | 0.168 | 0.0008 | 0.975 | 0.0095 | 23.6 |
| split EKF (points combined) | 0.666 | 0.0015 | 1.46 | 0.0135 | 4.16 |
| split UKF (points combined) | 0.556 | 0.0023 | 1.35 | 0.0118 | 28.1 |
| EKF (est. points combined) | 0.212 | 0.0019 | 1.32 | 0.0163 | 2.63 |
| UKF (est. points combined) | 0.198 | 0.0010 | 1.43 | 0.0127 | 25.8 |

**Table 6.1:** Simulated Satellite: Tracking results; Inertial model with correct feature matching

From figure 6.7 and table 6.1 we can deduce that the calibration algorithm provides decent translation estimates for this simulation, although the rotation estimate is more noisy. The UKF algorithms are consistently slower by a wide margin (on occasion by a factor of almost ten), without offering noticeably better accuracy. Treating 2D measurement points as separate measurements did not provide stable estimates and diverged under the test conditions. The noticeably most accurate – and fastest – method is to use a single EKF to process the points as a combined measurement.

It is interesting to look at what happens when the measurements are corrupted by faulty 2D-3D matches. An absence of visible features is arguably preferable to the wrong and confusing data provided by false matches. The possibility of incorrectly matching feature points is however a real possibility in real-world implementations, and might prove useful to investigate. Experimental results are shown in table 6.2. An asterisk (*) in the first column[4] indicates that measurements deviating strongly from the corresponding estimate were discarded.

We see that the calibration algorithms are extremely sensitive to false matches. The rotation estimate becomes practically useless when faulty matches are included for processing, even

[3] tracking error is computed according to section §6.1

[4] only for the methods using a calibration algorithm due to their sensitivity to false matches, and the fact that they directly "measure" the state vector

| Method used | RMS error | | | | time |
|---|---|---|---|---|---|
| | translation (m) | velocity (m/s) | rotation (deg) | angular rate (deg/s) | $\tau$ (sec) |
| Cal. & LKF+EKF | 1.92 | 0.0028 | 172 | 0.445 | 3.85 |
| Cal*. & LKF+EKF | 0.0964 | 0.0007 | 4.97 | 0.0342 | 3.74 |
| Cal. & LKF+UKF | 1.72 | 0.0029 | 156 | 0.183 | 15.7 |
| Cal*. & LKF+UKF | 0.0941 | 0.0007 | 5.74 | 0.0386 | 13.5 |
| EKF (points combined) | 0.792 | 0.0020 | 3.94 | 0.0316 | 2.67 |
| UKF (points combined) | 0.734 | 0.0038 | 2.70 | 0.0408 | 30.9 |
| EKF (est. points combined) | 0.732 | 0.0019 | 2.69 | 0.0245 | 2.37 |
| UKF (est. points combined) | 0.571 | 0.0028 | 2.49 | 0.0247 | 25.8 |

**Table 6.2:** Simulated Satellite: Tracking results; Inertial model with feature matching errors

though the translation estimate suffers less. Discarding suspect results (indicated by an asterisk) is highly successful. Even so, the most accurate rotation estimate is obtained by using a single EKF to process the combined feature points, even when taking faulty matches into account. An interesting result is that we can increase accuracy by replacing occluded feature points with their predicted measurements. This can be explained by the fact that these constructed measurements lessen the filter's sensitivity to the faulty matches.

| Method used | RMS error | | | | time |
|---|---|---|---|---|---|
| | translation (m) | velocity (m/s) | rotation (deg) | angular rate (deg/s) | $\tau$ (sec) |
| Cal. & LKF+EKF | 0.0136 | 9.76e-6 | 5.49 | 0.0375 | 3.30 |
| Cal. & LKF+UKF | 0.0159 | 9.73e-6 | 9.55 | 0.0484 | 16.9 |
| EKF (points combined) | 0.0366 | 0.0003 | 0.415 | 0.0105 | 3.93 |
| UKF (points combined) | 0.155 | 0.0007 | 0.961 | 0.0094 | 25.5 |

**Table 6.3:** Simulated Satellite: Tracking results; Hill's model with correct feature matching

Lastly, we look at the effect of incorporating Hill's equations into our system model. The result is, as expected, much better estimates for translation. The tracking error for the rotational elements remain the same, which is again what we would expect (since the rotation is independent from the translation). A surprising result is that the calibration algorithm's translational accuracy now outperform the (already good) single EKF. This can possibly be explained by the fact the the translation and rotation is more uncoupled when using a separate LKF and EKF (or UKF) as opposed to a single EKF (or UKF). The uncertainty of the rotation feeds through to the translation estimate in a single nonlinear filter. The calibration-based algorithms still provide inferior rotation estimates when compared to the alternative.

To summarise: The best approach is arguably to use a single EKF to process the feature points of each image frame as a single concatenated measurement. This method is fast, accurate and

also relatively robust. The UKF does not show any consistent advantage over the EKF, although it performs slightly better in certain instances. The calibration algorithms are not good at providing reliable rotation estimates from noisy images, and are especially vulnerable to faulty feature matches. In practice these algorithms might serve a purpose for initialization of the state vector of one of the methods using only a Kalman Filter. Incorporating Hill's equations into the system model has a predictably beneficial influence on the estimates, since the satellite was simulated to obey Hill's orbital model. Modeling the movement more accurately also reduces any bias in the unmodeled disturbances, thereby approximating the assumptions inherent in the Kalman Filter more closely. Tracking is acceptable even without modeling Hill's equations and using inertial rigid body dynamics instead. This is confirmation of a reasonable measure of robustness in the tracking algorithms.

Splitting the single Kalman Filter model (EKF or UKF) into two parts, or replacing invisible points with their estimates as "fake" measurements, generally decrease the performance of the filter. An exception is the case where false point matches occur. We see in table 6.2 that the fake measurements decrease the filter's sensitivity to the erroneous matches. It is also interesting that the UKF's most noticeable advantage over the EKF is for cases in which we take steps which introduces obvious modeling inaccuracies (for example forcing first the rotation and then translation estimate to be constant in the split KF, or by incorporating fake measurements to replace obscured features). This hints at the possibly greater flexibility of the UKF – something advocated by all the UKF's proponents.

# Chapter 7

# Tracking Actual Targets

In chapter 6 we looked at how the algorithms that were described in chapter 5 fared on a simulated image sequences. The results that were obtained looked promising, and the logical next step would be to test on real-world image sequences. When setting up an actual experiment one naturally runs into some practical constraints. Obtaining accurate ground truth information is not always easy. In the absence of reliable ground-truth data, some authors devised alternate ways of testing for accuracy. Broida et al. [1] projects the estimated 3D features back to a simulated 2D image, and measures the Euclidean distances between these "predicted" and observed image features. A similar approach is used by Salvi et al. [14]. Although ingenious the resulting error norm is not a direct measure of the accuracy in the parameters we are interested in tracking. We chose to set up controlled experiments in which the target could be moved in a easily measurable way. A second practical constraint is that realistic data sets for the intended application is not always possible to obtain. This is especially true in our case, where the intended application is over several tens of metres and in an orbital environment. The distances over which the practical experiments were set up are closer than would be found in a satellite constellation, but this is partially compensated for by using a smaller target and a sensor with lower resolution.

We chose to measure the estimation accuracy in the same way as was done in section §6.1 for the simulated movement sequence. A small addition is that the convergence time is shown in addition to the computational time in tables 7.1 and 7.2. As with the simulated targets we first look at results obtained from using calibration algorithms, and follow up by looking at results from treating the 2D feature points as measurements directly. We use an inertial physics model as in section §6.3.

## 7.1 A box on a linear rail

As a first practical proof-of-concept test, we tracked a cardboard target fitted with ten LED markers. Physically affixing clearly identifiable feature points is an approach also used by Broida [1]. The target was set up to move to and fro along a straight track, and could simultaneously be rotated in the vertical axis about its centre of mass.

The cardboard target consisted of a cardboard box with dimensions 22,7 cm $\times$ 31,3 cm $\times$ 26,4 cm, and is shown in figure 7.3. The target was placed at a distance so that the LED markers could clearly be distinguished, while staying in the field of view at all times.

The camera used for the image sequence is a regular Canon S45 consumer digital camera. The images were recorded as still frames at a pixel resolution of 1024x768, and at time intervals averaging 0.7 seconds apart. The low frame rate resulted in only 58 images in the sequence, though covering more than one complete movement cycle. The camera has an effective focal length[1] of $-1088$ units, which meant placing the target at a distance of approximately 2 m along the CCS $Z$-axis. After recording the images they were processed offline.

During the sequence there are always 7 to 9 visible LED feature points. The 2D image points were matched with the their original 3D counterparts by an heuristic method based on their relative geometry. For this purpose two additional LED markers were placed at different distances from one of the corner LEDs, as is visible in figure 7.3.

The rail-mounted target platform could be moved by means of a computer equipped with hardware interface card. The target was sinusoidally moved parallel to the CCS Z-axis, while being rotated in the CCS X-Z plane by means of a stepper motor. Rotational movement was somewhat erratic due to the stepper motor being too weak for the task at hand. The initial position of the target, relative to the camera, was measured as $[X, Y, Z]^T = [-0.48,\ 0.27,\ 2.00]^T$m (expressed in CCS coordinates). See figure 7.2 for a schematic top-down view of the camera and target. Note that the way in which the CCS axes are defined imply a negative X and positive Z coordinate (figure 2.2).

The ground truth is *only approximate*, since no accurate measuring equipment was available (translation is accurate to roughly 0.01m and rotation accurate to roughly 5°). Inaccuracy in the experimental setup did not guarantee that movement of the target was perfectly parallel to the CCS's Z-axis. The main purpose of this test was to verify that the proposed algorithms do, indeed, work for a real-world system.

The initial estimate for each of the tracking algorithms was chosen as position $[X, Y, Z]^T = [0,\ 0,\ 3]^T$m, velocity $[v_x, v_y, v_z]^T = [0,\ 0,\ 0]^T$m/s, quaternion rotation $[q_1, q_2, q_3, q_4]^T = [0,\ 0,\ 0,\ 1]^T$, and (body-referenced) angular velocity of $[\omega_x, \omega_y, \omega_z]^T = [0,\ 0,\ 0]^T$deg/s. The first data point in each estimation graph is shown after the first measurement is processed.

[1] see equation (2.1.1)

**Figure 7.1:** The LED-fitted target (box) mounted on the computer-controlled sliding rail

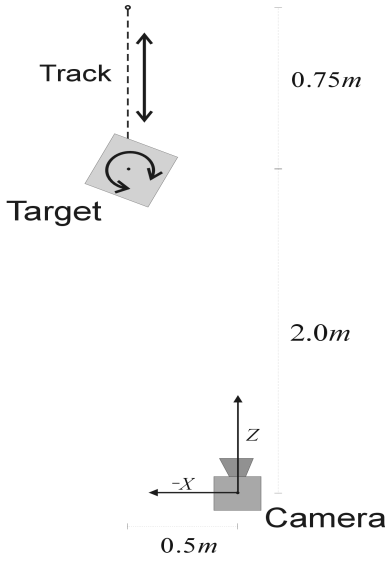



**Figure 7.2:** A schematic overhead view of the target and camera setup.

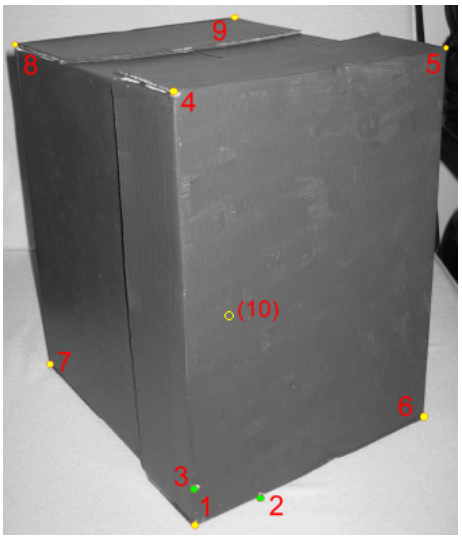



**Figure 7.3:** The improvised target (box) with LED markers indicated

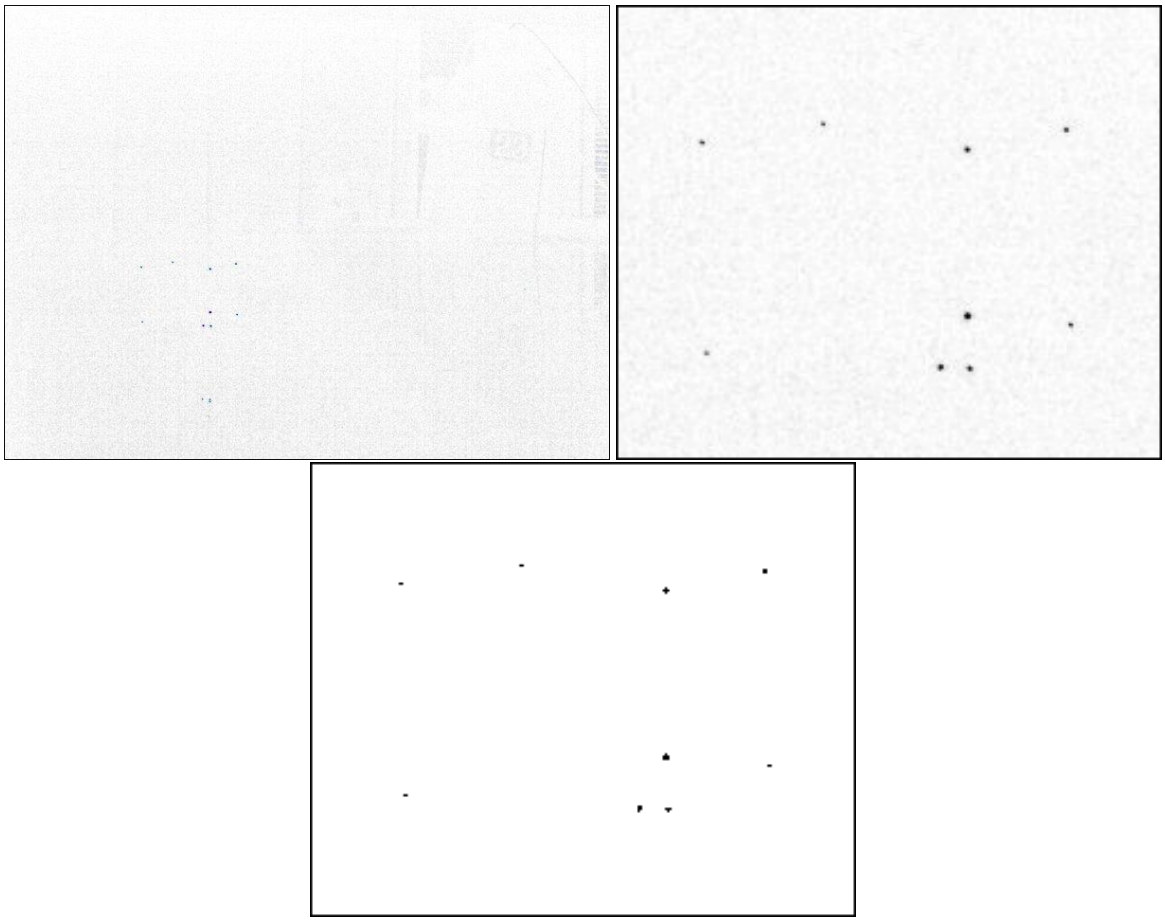


**Figure 7.4:** A single image frame, with enlarged detail showing the effect before and after thresholding

### 7.1.1 Selected results

The way in which the different algorithms were applied to the data is identical as for the simulated data set in section §6.2. Selected results are shown in figures 7.5 and 7.6.

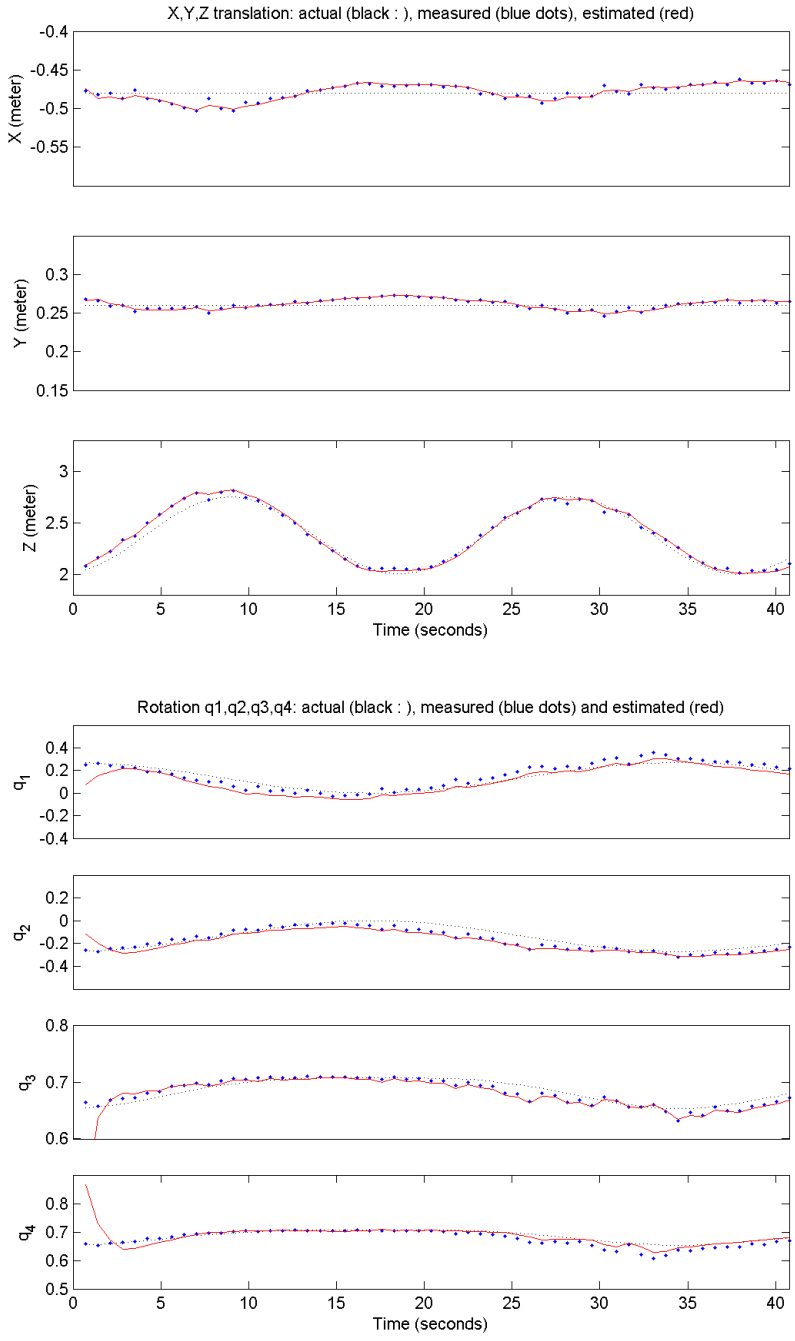



**Figure 7.5:** Box on a rail: Calibration algorithm and UKF - translation and rotation

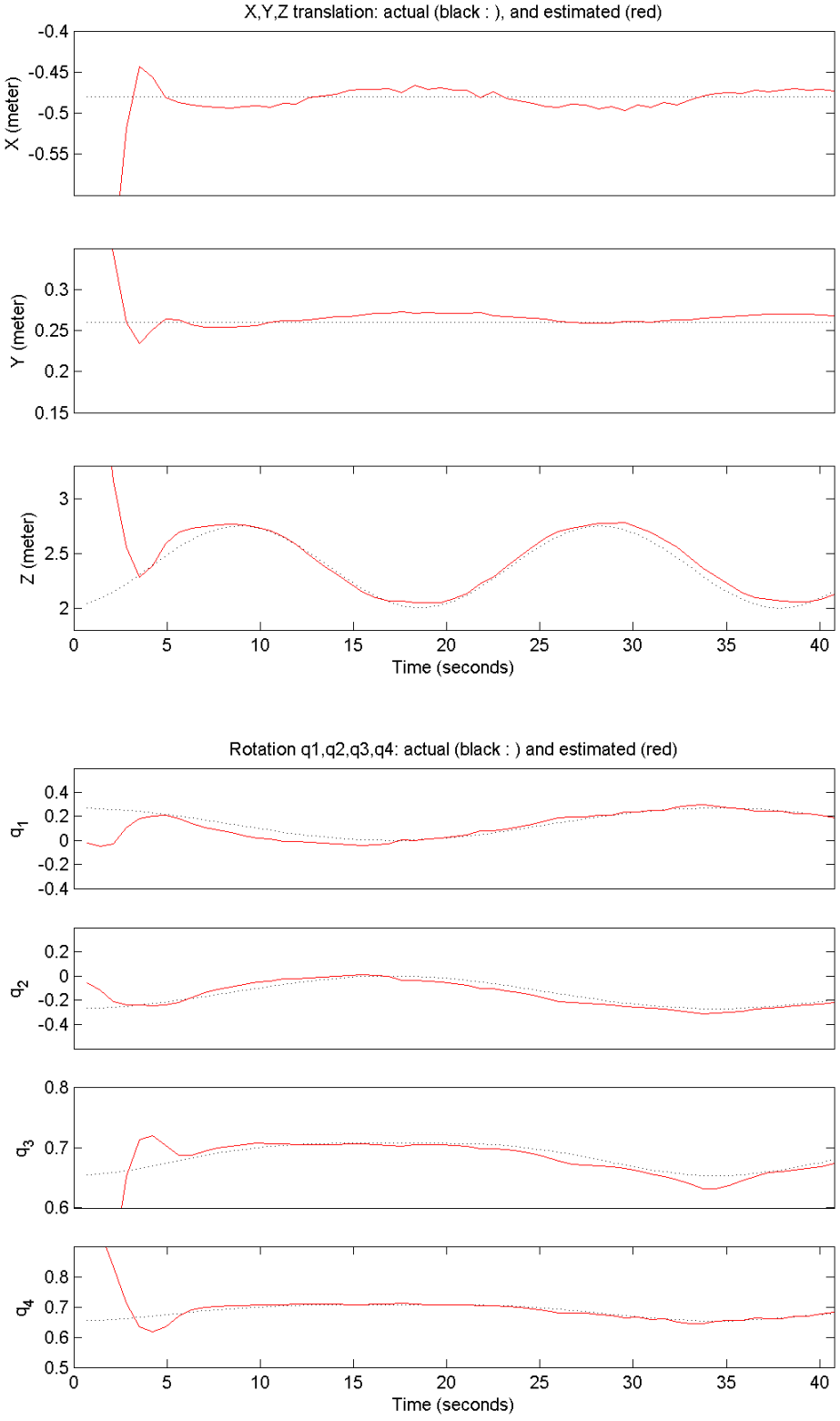



**Figure 7.6:** Box on a rail: Single EKF with concatenated measurements - translation and rotation

### 7.1.2 Summary and discussion of results

| Method used | RMS error | | | | time | c.time |
|---|---|---|---|---|---|---|
| | translation (m) | velocity (m/s) | rotation (deg) | angular rate (deg/s) | $\tau$ (sec) | $\varsigma$ (sec) |
| Cal. & LKF+EKF | 0.0317 | 0.0398 | 7.94 | 1.49 | 0.45 | 8 |
| Cal. & LKF+UKF | 0.0317 | 0.0398 | 8.44 | 1.55 | 1.55 | 10 |
| EKF (features separate) | 0.0379 | 0.0348 | 14.4 | 0.822 | 0.98 | 14 |
| UKF (features separate) | - | - | - | - | - | - |
| EKF (features concat.) | 0.0502 | 0.0329 | 4.81 | 1.17 | 0.32 | 10 |
| UKF (features concat.) | 0.0700 | 0.0358 | 4.66 | 0.927 | 3.22 | 10 |

**Table 7.1:** Box on a linear rail: Tracking results

The results obtained for this practical test echo the results of the simulated sequence shown in table 6.1. Once again the single nonlinear filter, which uses the feature points as a concatenated measurement, gives the best rotation estimate with the fastest execution time. Once again the performance of the UKF is generally worse then that of the EKF.

The value of $\tau$ shows the average time needed to process the measured data on the author's system. Just as with the simulated data of chapter 6, the UKF is much slower than the EKF. In table 7.1, the last column shows the time, $\varsigma$, that was required for each of the methods to converge to a stable estimate. The value of $\varsigma$ is approximate and should be seen as a ballpark figure, since some of the parameters converge at different rates. The methods using a calibration algorithm converge quicker due to the reduced complexity of the associated filter. The EKF also tends to converge quicker than the UKF.

For this experiment, the EKF treating single 2D feature points as separate measurements yielded reasonable results, which can be accessed in appendix E. Even though some success was reached, this method was slower to converge and yielded generally worse estimates. The fact that the UKF could not be set up to converge with this approach highlights its inferiority when compared to concatenating the visible feature points as a single measurement.

All the methods (except UKF with separate point measurements) were able to track the target quite accurately: i.e. the mean square error of the position is in the order of 2% of the full scale distance, and the better methods could compute the pose to within 5°. It needs to be stressed that there are systemic offsets between the ground truth and the estimated values, due to inaccuracies in the measurement of the camera's initial position, and especially orientation, relative to the target. The actual theoretical accuracy of these methods might be better judged from simulated data.

## 7.2 A robot-controlled target

We needed to set up an experiment in which the target could be controlled more smoothly, and with more degrees of freedom, as was the case with the rather crude setup of section §7.1. Furthermore we also needed accurate ground truth measurements to use for comparison with the estimated trajectory.

A cubical $30 \times 30 \times 30$ cm plastic target was constructed, and attached to an industrial robot arm, as shown in figure 7.7. The target was fitted with eight LED markers (one on each corner) – a similar setup to that of section §7.1. Different LED colours (two LEDs of each colour) were used to differentiate between features, for identification and matching.

The robot was programmed to move the target through a specific translation sequence, while simultaneously rotating it about the vertical ($Y$) axis. Mostly of the translational movement took place in the vertical plane, although the geometry of the arm also resulted in movement along the $Z$-axis.

A consumer-level Fuji FinePix S602Z digital camera was used to record the sequence as as 640x480 video at 30 frames per second. The total captured video length is 22 seconds exactly – a total of 660 frames. The footage was processed offline.

The ground-truth measurements for the robot's movement was available at a rate of 5 measurements per second. The robot's ground truth was obtained relative to its base, while the tracking algorithms provide the coordinates in the CCS. The orientation and position between the camera and the robot were measured (with measuring tape), and accounted for. Small inaccuracies in the camera's viewing angle, which later became apparent, are to blame for a small offset between "actual" and measured trajectories. This (rather small) offset is systemic and identical for each of the tracking algorithms.

The trajectory of the robot-controlled target is shown in figure 7.9. The target was moved in a closed and roughly rectangular path, while being rotated about its vertical axis in alternating directions. A black reference mark (star) in figure 7.9 gives an idea of the angles through which the target was rotated. A superposition of every tenth image frame is shown in figure 7.11, to give an idea of what it is that the camera saw. The number of visible marker points is shown as a function of the frame number in figure 7.10

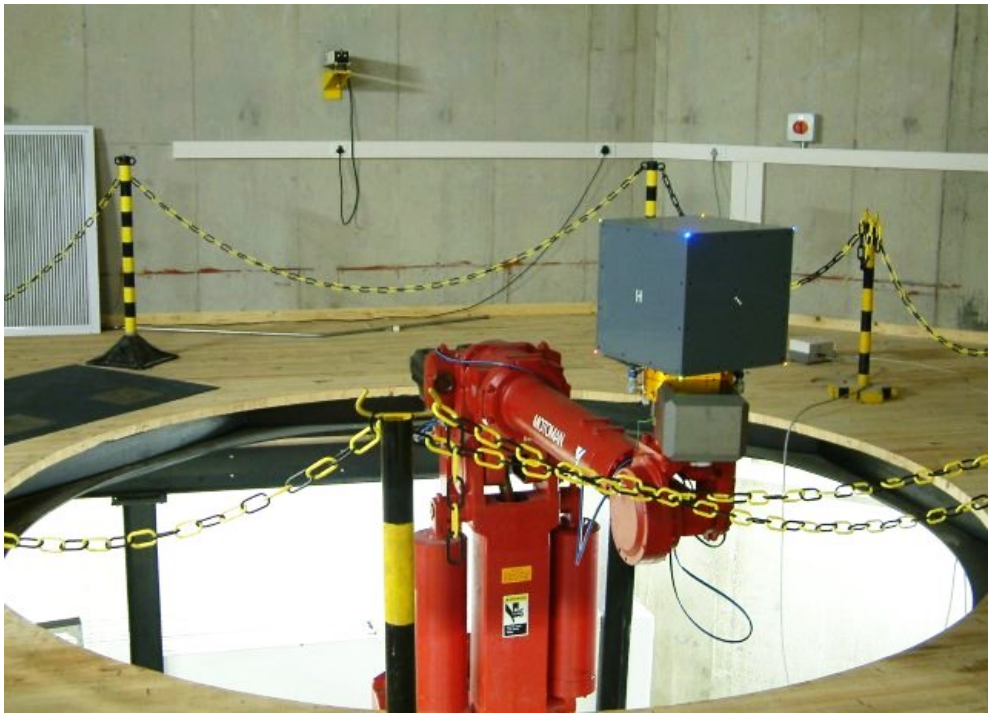


**Figure 7.7:** The LED-fitted (cubical) target mounted on the industrial robot arm

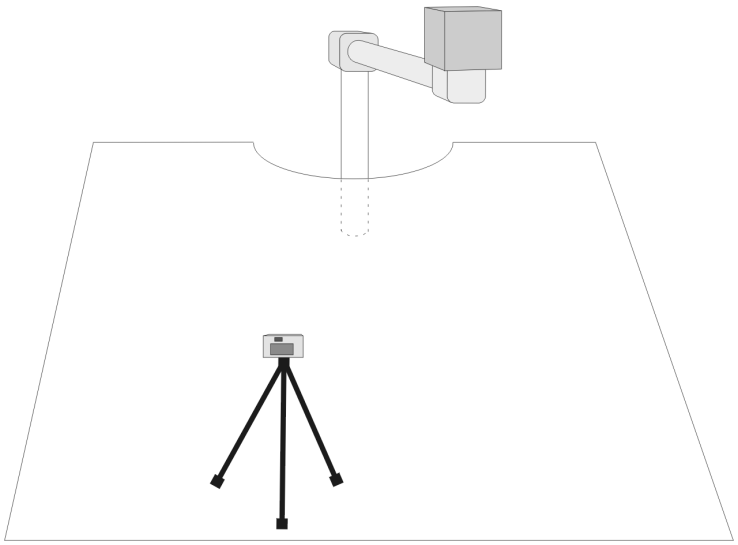


**Figure 7.8:** Robot-controlled target: The camera's position

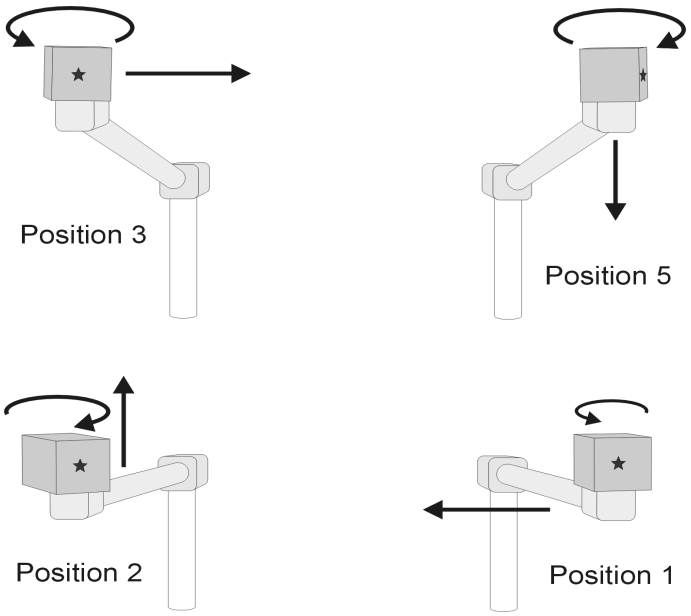



**Figure 7.9:** The movement sequence of the robot-controlled target

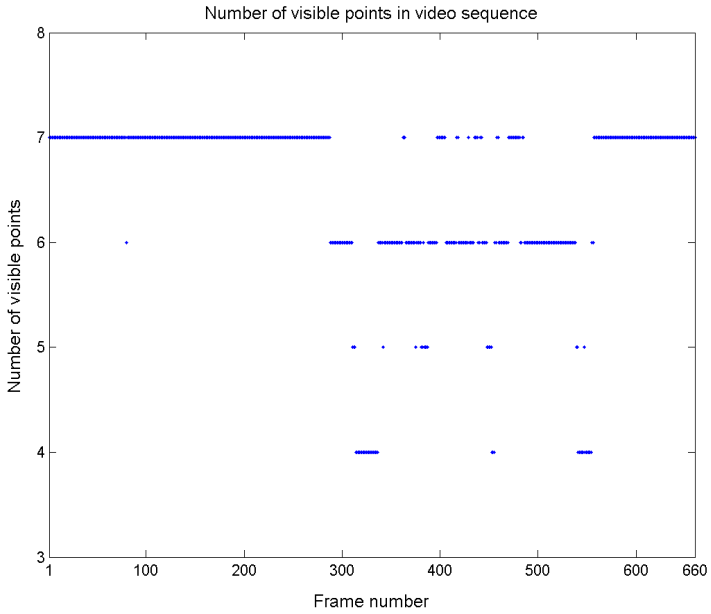



**Figure 7.10:** Robot-controlled target: The number of visible marker points in every frame

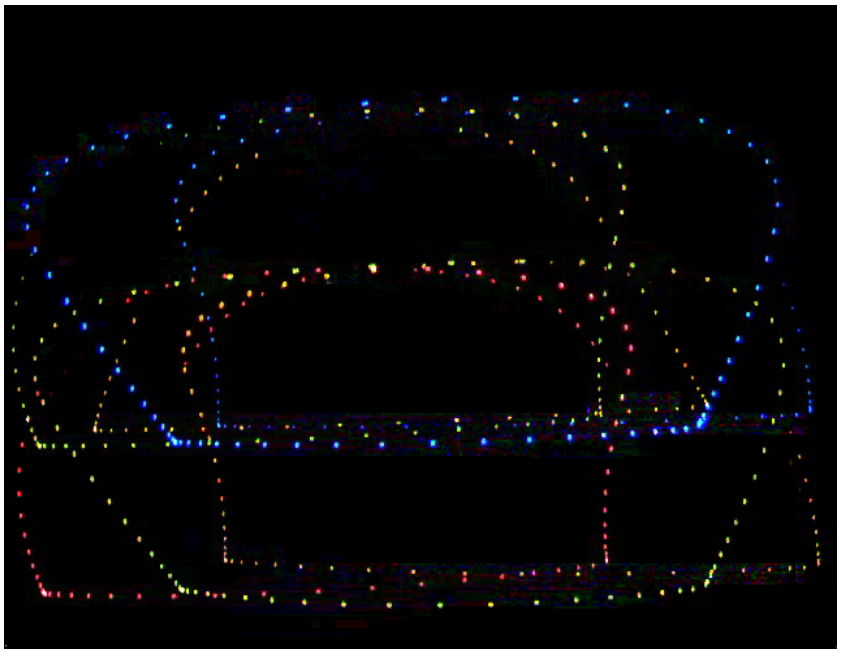


**Figure 7.11:** Robot-controlled target - A superposition of every tenth image frame

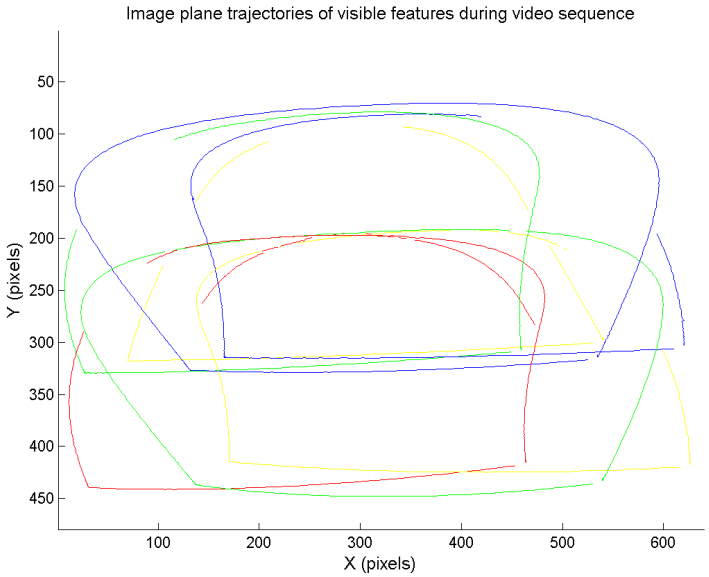



**Figure 7.12:** Robot-controlled target - Extracted image plane trajectories for visible feature points

### 7.2.1 Selected results

As shown in figure 7.10 there are several image frames for which fewer than six feature points are visible. These frames contain too little information for calibration algorithms to generate reliable estimates from. Since these frames are relatively few and scattered, we chose to ignore them as valid frames when using calibration algorithms. The dynamic system model of the Kalman Filter was used to update the state estimate, without adjustment, for these time steps. Using a nonlinear filter which does not rely on calibration algorithms, does not require this form of intervention.

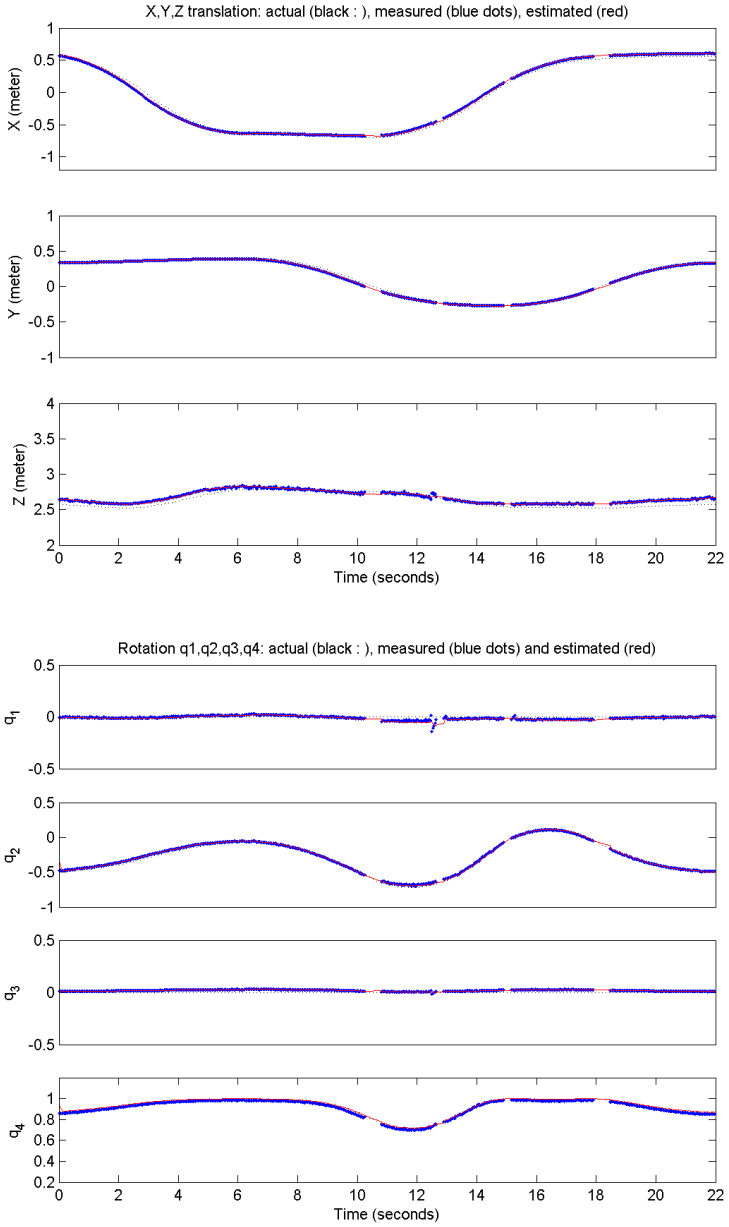



**Figure 7.13:** Robot-controlled target: Calibration algorithm and UKF - translation and rotation

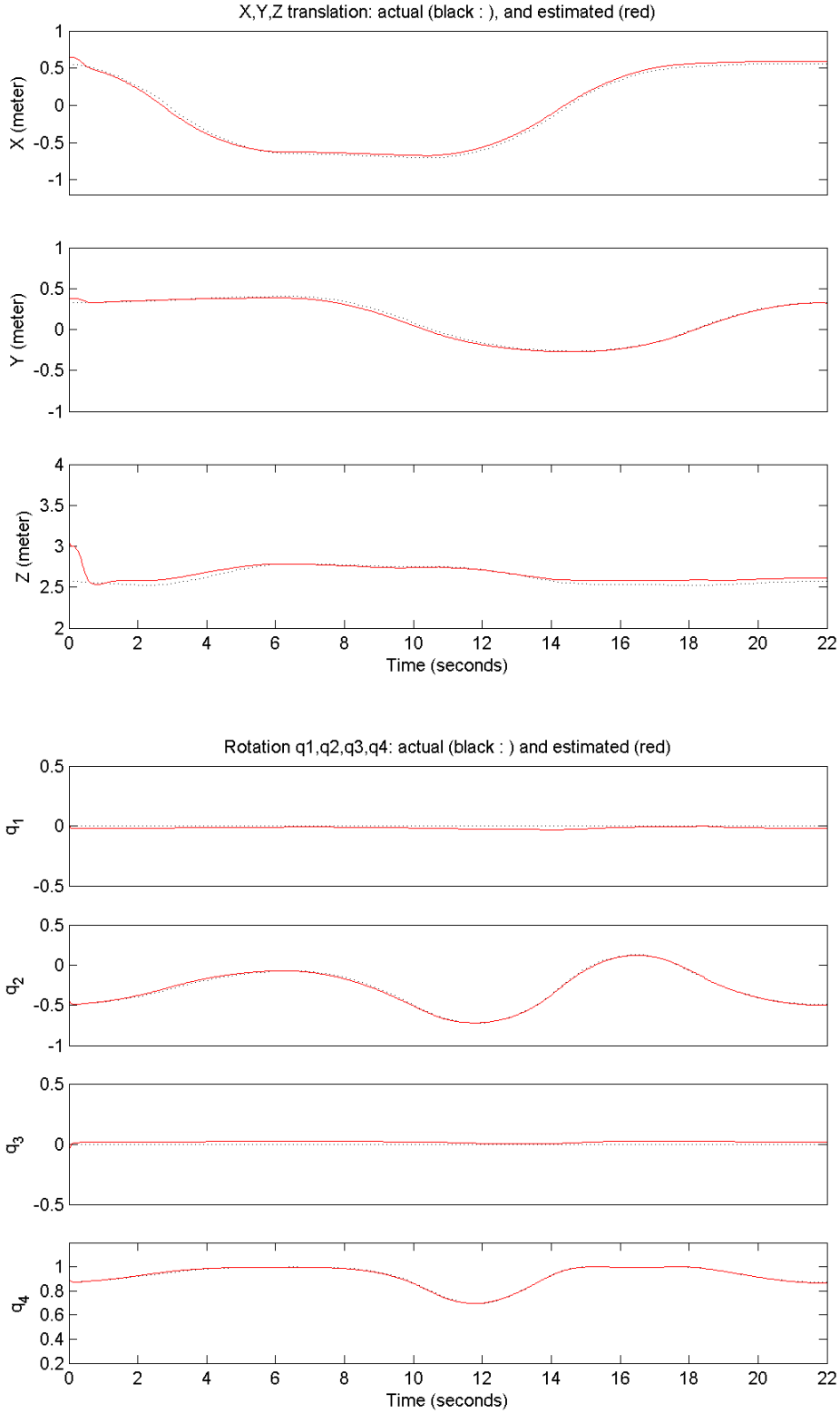



**Figure 7.14:** Robot-controlled target: Single EKF with concatenated measurements - translation and rotation

### 7.2.2 Summary and discussion of results

| Method used | RMS error | | | | time | c.time |
|---|---|---|---|---|---|---|
| | translation (m) | velocity (m/s) | rotation (deg) | angular rate (deg/s) | $\tau$ (sec) | $\varsigma$ (sec) |
| Cal. & LKF+EKF | 0.0649 | 0.0329 | 3.74 | 3.22 | 4.95 | 2 |
| Cal. & LKF+UKF | 0.0649 | 0.0329 | 4.30 | 8.30 | 17.5 | 2 |
| EKF (features separate) | 0.212 | 0.258 | 7.21 | 7.60 | 8.67 | 3 |
| UKF (features separate) | - | - | - | - | - | - |
| EKF (features combined) | 0.0570 | 0.0269 | 3.43 | 2.69 | 3.54 | 1.5 |
| UKF (features combined) | 0.0640 | 0.0723 | 6.02 | 9.94 | 46.6 | 2 |
| split EKF (feat. comb.) | 0.0572 | 0.0283 | 3.45 | 3.64 | 5.67 | 2 |
| split UKF (feat. comb.) | 0.0557 | 0.120 | 5.12 | 22.7 | 36.5 | 2 |

**Table 7.2:** Robot-controlled target: Tracking results

The definitions of $\tau$ and $\varsigma$ are identical to those in table 7.1.

The results of this second, and more accurate, real-world test once again support the results that were obtained in chapter 6 as well as section §7.1. Using a single EKF to process the feature points of each frame as a concatenated measurement, once again, emerges as the winning technique. Not only is it the fastest (in this implementation), but also generally the most accurate. It is once again clear that there is a performance loss when artificially splitting the EKF (or UKF) into two functionally separate filters. Treating the feature points as separate measurements does work with the EKF, but is once again not very accurate and prone to divergence. As was found with all the other results, the UKF in its implemented guise does not perform better than the EKF, and is significantly slower.

The accuracy obtained is generally quite good and encouraging. The mean square error of the best position estimate is less than 2% of the full scale distance, and the rotation estimate's mean square error less than $4^\circ$. These results have been obtained even though slight inaccuracies in camera positioning caused some small but noticeable offsets between the estimates and the ground truth.

# Chapter 8

# Summary and Conclusion

In this thesis we examined ways in which a target's 3D position and rotation could be determined by using a sequence of images from a single digital (video) camera. Three dimensional motion tracking is already widely used for motion capture in the movie industry, in the medical industry, and for robotic computer vision. Almost all existing commercial systems known to the author employ stereo vision, achieved by means of two (or typically more) cameras viewing the target from different angles. It appears as if 3D tracking of an object by means of monocular vision (except as part of the Structure from Motion problem) is not widely used or even widely discussed in the literature. Our application differs from the usual SfM problem in that we can assume the structure of the target to be known. We are only interested in obtaining the 3D position and rotation of the target. This simplification of the SfM problem enables the accurate determination of absolute position – something which most SfM solutions fail on delivering.

The specific motivation behind our research into 3D tracking by monocular vision, is for the application of close formation flight between satellites. For this application it is not feasible to have multiple cameras viewing the target. The stereoscopic effect of viewing a target at, say, fifty metres away, by means of cameras mounted one or two metres apart is very little. These limitations negate many of the advantages (such as simplicity) offered by stereo vision. It is furthermore not practical for satellites to be fitted with multiple cameras where one could suffice, due to strict weight and layout constraints.

The work in this thesis builds on work done on the structure from motion (SfM) problem by, amongst others, Broida [1] and Venter [4] . The SfM problem – determining (scaled) structure and (scaled) motion from a single camera's image sequence – is less tractable than motion estimation alone. We restricted the problem formulation to determining the 3D position and orientation of the target (along with their first derivatives), implying that the structure of the target had to be known beforehand. Structure recognition was achieved by placing markers on the object at predetermined positions, which is similar to the way in which features are identified when using stereoscopic vision for medical tomography, or motion capture in the film industry. This solution is practically feasible for a satellite application, since visual markers can easily be affixed to a satellite, and the visual space environment is largely clutter free.

The desired solution arrived at is conceptually similar to methods explored by Goddard [3], although the specific implementation and application differ.

The use of Kalman filters was chosen as the central theme around which each of the different methods were designed and tested. The Kalman filter is a very powerful tool for estimating visible and hidden states of a dynamic system, although it is only provably optimal for linear systems. The nonlinear extensions of the Kalman Filter (for example the EKF or UKF) are far more useful, but lack the guarantee of stability or of optimality. It seems that the Extended Kalman Filter is the algorithm of choice in SfM and motion tracking research, although a lot of attention is currently being drawn to the Unscented Kalman Filter in various applications. We were able to implement algorithms employing the EKF and/or UKF in a way that the desired tracking estimates converged quickly and thereafter followed the target stably. It was, however, found that the filters needed to be set up carefully, since stability and performance of the filter rely greatly on the choice of initial and state covariance matrices. The fact that real-world systems rarely closely resemble the statistical assumptions inherent to the Kalman Filter, coupled with the fact that linearisations are made to accommodate the nonlinear properties of the problem, mean that we can by no means assume that the filters perform optimally. There is therefore much room for further investigation into alternative ways of achieving similar results. The UKF is currently a hot topic of discussion, and many authors claim that it is superior to the EKF in every way. While having its advantages, we found no clear performance advantage to using the UKF in the algorithms explored in this thesis. Even taking the UKF's reduced speed and lack of accuracy improvement over the EKF into account, it still has a few things counting in its favour. The UKF does not require the mathematical derivation of any Jacobian matrices. A direct advantage of this is that is reduces the likelihood of (human) error in the modeling process. Many a researcher is wary of the implementation of these matrices for large and complex systems, since it may lead to unforeseen difficulties (as discussed by [18]). A second advantage is the UKF's ability to act with greater flexibility under highly nonlinear conditions. The specific problem of motion estimation does however not require these abilities to the extent that the UKF becomes superior.

Alternatively, as a conceptually different approach, it is possible to compute a 3D position and pose estimate from a single image, without utilising any information about the dynamic model or previous image frames. Some of these methods, called "calibration algorithms", were tested. Due to their rather unreliable estimates (especially of rotation) when few markers are visible or when significant image noise or false feature matches are present, it is still advantageous to use Kalman Filters on the output of these algorithms. The methods relying on calibration algorithms were found to perform worse, in general, than the methods utilising only the Kalman Filter. Since no previous estimate is needed for these algorithms to work, they are very useful for generating a first estimate for an otherwise sequential algorithm.

To conclude: We demonstrated by means of simulated and practical experiments that a rigid 3D object can be accurately tracked by means of a sequence of monocular camera images. Some methods fare significantly better than others, so that a clear recommendation can be made as

to which methods to pursue for future development. The use of an Extended Kalman Filter to process the 2D image feature points directly as concatenated measurements, was found to be the best-performing method in both speed and accuracy, and performs sufficiently well to have practical applicability. The filter's state vector can best be initialized by using estimates obtained from the so-called calibration algorithm.

## 8.1 Possible Future Work

While the conceptual model has been developed and successfully tested, much additional work remains in terms of analysis and especially implementation. The algorithms described in this thesis were coded and tested in the MATLAB environment. All image sequences were processed offline. A practical system will require images to be processed online, whereas the computational speed requirements would require the algorithms to be coded in a faster technical language like C++.

Robustness is a prime consideration when designing a system applicable to monitoring satellite formation flight. It would be desirable to examine this issue more thoroughly for further implementation. Improvements to robustness might also be attained by using unconstrained parameters for incremental rotations, instead of normalized quaternions, as mentioned in section §5.1.

The practical experiments in this thesis yielded promising results, but due to physical constraints were not conducted in an environment with similar dimensions as the intended application. It is assumed that close formation flight between satellites will be in the order of tens of metres rather than the distances for which the practical tests were set up. We were, however, able to simulate image sequences for more realistic distances. These simulations indicate that the algorithms will still be able to perform their function under the intended conditions.

The feature points which were used for the 2D to 3D reconstruction were preselected markers on the target object. After the markers were detected they had to be individually identified so that the 2D-3D matching would be correct. For our experiments we used markers of different colours, combined with heuristic methods of identifying each visible marker. An effective way of discriminating between markers based solely on their relative image-plane positions would simplify this task. There are effective algorithms available for automatic feature detection and image tracking (like the KLT algorithm) which could do away with placing preselected markers, given that the 3D location of the features could be determined in the target's coordinate frame. This would, however, most likely unnecessarily complicate the already difficult task of accurate tracking, since the user has little control over where the KLT algorithm decides on placing features. Furthermore, placing physical markers on the target should be practically feasible as far as satellites are concerned.

# Appendices

# Appendix A

# Derivation of the Discrete Linear Kalman Filter (LKF)

## A.1 General form of the Filter

A filter gives an estimate of a state vector at the current time, based on all measurements taken up to the present time. The Kalman Filter is a minimum mean square error (MMSE) estimator that works recursively.

Speaking in broad terms, at every time step, a Kalman filter needs to

- predict the mean and covariance of the state estimate, by using the appropriate time-propagation system model, and
- correct the predicted mean and covariance of the state estimate by incorporating (direct or indirect) measurements of the state (if available).

The following derivation follows a similar approach as the one provided in [19]. For a more thorough exposition we recommend [19, 16, 30].

## A.2 The system model

Consider a discrete-time *linear* dynamic system described by a vector difference equation[1] with additive white Gaussian noise. The noise models "unpredictable" disturbances, and inaccuracies in the model itself. The dynamic equation can be written as

$$\boldsymbol{x}_{k+1} = \boldsymbol{\Phi}_k \boldsymbol{x}_k + \boldsymbol{\Gamma}_k \boldsymbol{u}_k + \boldsymbol{v}_k$$

[1] A vector equation can be thought of as just a set of equations.

where $\boldsymbol{x}_k$ is the $n_x$-dimensional state vector, $\boldsymbol{u}_k$ is an $n_u$-dimensional known input vector (for example a control signal) and $\boldsymbol{v}_k$ is a zero-mean white Gaussian process noise with covariance $E[\boldsymbol{v}_k \boldsymbol{v}_k^T] = \boldsymbol{Q}_k$.
In cases where the noise is mostly due to unknown forcing impulses, the Gaussian noise can be fed through the forcing function. The additive noise $\boldsymbol{v}_k$ is then replaced by $\boldsymbol{\Gamma}_k \boldsymbol{v}_k$ with covariance $E[(\boldsymbol{\Gamma}_k \boldsymbol{v}_k)(\boldsymbol{\Gamma}_k \boldsymbol{v}_k)^T] = \boldsymbol{\Gamma}_k \boldsymbol{Q}_k \boldsymbol{\Gamma}_k^T$.

The measurement equation is

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{w}_k$$

where $\boldsymbol{z}_k$ is the $n_z$-dimensional measurement, and $\boldsymbol{w}_k$ is zero-mean white Gaussian measurement noise with covariance $E[\boldsymbol{w}_k \boldsymbol{w}_k^T] = \boldsymbol{R}_k$.

The matrices $\boldsymbol{\Phi}_k, \boldsymbol{\Gamma}_k, \boldsymbol{H}_k, \boldsymbol{Q}_k$ and $\boldsymbol{R}_k$ are assumed known and possibly time-varying. The initial state $\boldsymbol{x}_0$, in general unknown, is modeled as a Gaussian random variable with known mean and covariance. The two noise sequences and the initial state are assumed mutually independent.

Due to the linearity of the update and measurement equations, the Gaussian properties of the state and measurement's random variables are conserved – an instance of a Gauss-Markov process.

### A.2.1 About the transition matrix

Often, the differential equations of a system in their time-continuous form are known. However, a Kalman filter is generally implemented in discrete time in a computer cycling at a finite rate, and measurements taken at discrete intervals. In order to implement the Kalman filter, it suffices to know that the time continuous matrix differential equation can always be transformed as follows:

$$\dot{\boldsymbol{x}} = \boldsymbol{F}\boldsymbol{x} \qquad \Rightarrow \qquad \boldsymbol{x}_{k+1} = \boldsymbol{\Phi}_k \boldsymbol{x}_k.$$

The question is how easily this can be done in practice. When the matrix $\boldsymbol{F}$ is constant in time and the equation is linear (no elements of $\boldsymbol{x}$ occur inside $\boldsymbol{F}$ – always true for the LKF), then the matrix $\boldsymbol{\Phi}_k$ is a function only of $\boldsymbol{F}$ and the time step $\Delta t$, and is given by the matrix exponential

$$\boldsymbol{\Phi}_k = \mathrm{e}^{\boldsymbol{F}_k \Delta t} = \boldsymbol{I} + \boldsymbol{F}_k \Delta t + \frac{(\boldsymbol{F}_k \Delta t)^2}{2!} + \dots$$

[30]. When $\Delta t$ is much smaller than the dominant time constants in the series the following two-term approximation is sufficient:

$$\boldsymbol{\Phi}_k \approx \boldsymbol{I} + \boldsymbol{F}_k \Delta t$$

Slight inaccuracies in the system model can often be compensated for by the modeling of the system noise, and therefore approximations such as these are acceptable.

## A.3 The estimation algorithm

The estimation algorithm starts with an initial state estimate $\hat{\boldsymbol{x}}_0$ and an initial state covariance $\boldsymbol{P}_0$. These two parameters uniquely describe the estimate's Gaussian distribution. We therefore only need to know the conditional mean $\hat{\boldsymbol{x}}_k$ and covariance $\boldsymbol{P}_k$ at time step $k$, when we want to combine the time propagation of the state, and measurement $\boldsymbol{z}_k$, to a new Gaussian distribution at time step $k+1$. We also have the linear system model which propagates (in time) the Gaussian pdf to a new Gaussian pdf.

Note that the state estimate is computed as a conditional mean, which implies the state estimate covariance is the same as the estimation error covariance.

### A.3.1 Predicting the state

Now, assume that we are at time step $k$, and know that the estimated conditional mean of the state variable is $\hat{\boldsymbol{x}}_k$ with error covariance $\boldsymbol{P}_k$. The estimated state variable differs from the actual state by an unknown error quantity $\Delta\hat{\boldsymbol{x}}_k$,

$$\boldsymbol{x}_k = \hat{\boldsymbol{x}}_k + \Delta\hat{\boldsymbol{x}}_k.$$

Therefore, since $\hat{\boldsymbol{x}}_k$ is an unbiased estimate and $\boldsymbol{u}_k$ is known,

$$\begin{aligned} E[\boldsymbol{\Phi}_k\hat{\boldsymbol{x}}_k + \boldsymbol{\Gamma}_k\boldsymbol{u}_k] &= E[\boldsymbol{\Phi}_k(\boldsymbol{x}_k - \Delta\hat{\boldsymbol{x}}_k) + \boldsymbol{\Gamma}_k\boldsymbol{u}_k] \\ &= E[\boldsymbol{\Phi}_k\boldsymbol{x}_k] - E[\boldsymbol{\Phi}_k\Delta\hat{\boldsymbol{x}}_k] + \boldsymbol{\Gamma}_k\boldsymbol{u}_k \\ &= E[\boldsymbol{\Phi}_k\boldsymbol{x}_k] - 0 + \boldsymbol{\Gamma}_k\boldsymbol{u}_k \\ &= E[\boldsymbol{\Phi}_k\boldsymbol{x}_k] + \boldsymbol{\Gamma}_k\boldsymbol{u}_k + E[\boldsymbol{v}_k] \\ &= E[\boldsymbol{\Phi}_k\boldsymbol{x}_k + \boldsymbol{\Gamma}_k\boldsymbol{u}_k + \boldsymbol{v}_k] \\ &= E[\boldsymbol{x}_{k+1}]. \end{aligned}$$

From this we see that the unbiased state estimate can be propagated in time by the following equation

$$\bar{\boldsymbol{x}}_{k+1} = \boldsymbol{\Phi}_k\hat{\boldsymbol{x}}_k + \boldsymbol{\Gamma}_k\boldsymbol{u}_k.$$

### A.3.2 Predicting the state (error) covariance

$$\begin{aligned}
\bar{\boldsymbol{P}}_{k+1} &= E[(\boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}_{k+1})(\boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}_{k+1})^T] \\
&= E[((\boldsymbol{\Phi}_k \boldsymbol{x}_k + \boldsymbol{\Gamma}_k \boldsymbol{u}_k + \boldsymbol{v}_k) - (\boldsymbol{\Phi}_k \hat{\boldsymbol{x}}_k + \boldsymbol{\Gamma}_k \boldsymbol{u}_k))\,((\boldsymbol{\Phi}_k \boldsymbol{x}_k + \boldsymbol{\Gamma}_k \boldsymbol{u}_k + \boldsymbol{v}_k) - (\boldsymbol{\Phi}_k \hat{\boldsymbol{x}}_k + \boldsymbol{\Gamma}_k \boldsymbol{u}_k))^T] \\
&= E[(\boldsymbol{\Phi}_k(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k) + \boldsymbol{v}_k)\,(\boldsymbol{\Phi}_k(\boldsymbol{x}_k - \hat{\boldsymbol{x}}_k) + \boldsymbol{v}_k)^T] \\
&= E[\boldsymbol{\Phi}_k(\Delta\hat{\boldsymbol{x}}_k)(\Delta\hat{\boldsymbol{x}}_k)^T\boldsymbol{\Phi}_k^T + \boldsymbol{\Phi}_k\Delta\hat{\boldsymbol{x}}_k\boldsymbol{v}_k^T + \boldsymbol{v}_k\Delta\hat{\boldsymbol{x}}_k^T\boldsymbol{\Phi}_k^T + \boldsymbol{v}_k\boldsymbol{v}_k^T] \\
&= \boldsymbol{\Phi}_k E[(\Delta\hat{\boldsymbol{x}}_k)(\Delta\hat{\boldsymbol{x}}_k)^T]\boldsymbol{\Phi}_k^T + \boldsymbol{\Phi}_k\boldsymbol{0} + \boldsymbol{0}\boldsymbol{\Phi}_k^T + E[\boldsymbol{v}_k\boldsymbol{v}_k^T] \\
&= \boldsymbol{\Phi}_k \boldsymbol{P}_k \boldsymbol{\Phi}_k^T + \boldsymbol{Q}_k
\end{aligned}$$

If the additive noise is fed through the forcing function, as in section §A.2, we will replace $\boldsymbol{Q}_k$ with $\boldsymbol{\Gamma}_k \boldsymbol{Q}_k \boldsymbol{\Gamma}_k^T$ in the covariance update equation above.

### A.3.3 Predicting the measurement

We now have the prior estimate for the state vector at time step $k$, namely $\bar{\boldsymbol{x}}_k$, with covariance $\bar{\boldsymbol{P}}_k$. Since we assume a Gaussian distribution and an unbiased system model, the MMSE estimate of the predicted measurement is the measurement function applied to the predicted state.

$$\begin{aligned}
E[\boldsymbol{z}_k] &= E[\boldsymbol{H}_k \boldsymbol{x}_k + \boldsymbol{w}_k] \\
&= E[\boldsymbol{H}_k \boldsymbol{x}_k] + E[\boldsymbol{w}_k] \\
&= E[\boldsymbol{H}_k \boldsymbol{x}_k] + \boldsymbol{0} \\
&= \boldsymbol{H}_k E[\boldsymbol{x}_k] \\
&= \boldsymbol{H}_k \bar{\boldsymbol{x}}_k.
\end{aligned}$$

Therefore

$$\bar{\boldsymbol{z}}_k = \boldsymbol{H}_k \bar{\boldsymbol{x}}_k.$$

### A.3.4 Predicting the measurement covariance

The difference between the actual and predicted measurements is called the innovation, and therefore the measurement covariance is usually referred to as the *innovation covariance*. The

innovation covariance can be computed as follows

$$\begin{aligned}
\boldsymbol{S}_k &= E[(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k)(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k)^T] \\
&= E[(\boldsymbol{H}_k\boldsymbol{x}_k + \boldsymbol{w}_k - \boldsymbol{H}_k\bar{\boldsymbol{x}}_k)(\boldsymbol{H}_k\boldsymbol{x}_k + \boldsymbol{w}_k - \boldsymbol{H}_k\bar{\boldsymbol{x}}_k)^T] \\
&= E[(\boldsymbol{H}_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \boldsymbol{w}_k)\,(\boldsymbol{H}_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \boldsymbol{w}_k)^T] \\
&= E[\boldsymbol{H}_k\Delta\bar{\boldsymbol{x}}_k\Delta\bar{\boldsymbol{x}}_k^T\boldsymbol{H}_k^T + \boldsymbol{H}_k\Delta\bar{\boldsymbol{x}}_k\boldsymbol{w}_k^T + \boldsymbol{w}_k\Delta\bar{\boldsymbol{x}}_k^T\boldsymbol{H}_k^T + \boldsymbol{w}_k\boldsymbol{w}_k^T] \\
&= \boldsymbol{H}_k E[\Delta\bar{\boldsymbol{x}}_k\Delta\bar{\boldsymbol{x}}_k^T]\boldsymbol{H}_k^T + \boldsymbol{H}_k\boldsymbol{0} + \boldsymbol{0}\boldsymbol{H}_k^T + E[\boldsymbol{w}_k\boldsymbol{w}_k^T] \\
&= \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T + \boldsymbol{R}_k,
\end{aligned}$$

with $\Delta\bar{\boldsymbol{x}}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k$.

### A.3.5 Predicting the cross covariance

The cross covariance between the predicted state and the predicted measurement can be computed as follows

$$\begin{aligned}
\boldsymbol{C}_k &= E[(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k)^T] \\
&= E[(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)(\boldsymbol{H}_k\boldsymbol{x}_k + \boldsymbol{w}_k - \boldsymbol{H}_k\bar{\boldsymbol{x}}_k)^T] \\
&= E[(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)(\boldsymbol{H}_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \boldsymbol{w}_k)^T] \\
&= E[(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)^T\boldsymbol{H}_k^T + (\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)\boldsymbol{w}_k^T] \\
&= \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T + \boldsymbol{0} \\
&= \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T.
\end{aligned}$$

### A.3.6 Incorporating a measurement

The incorporation of measurements into the state estimate and covariance estimate relies on the following theorem:

> Given jointly Gaussian vectors $\boldsymbol{x}$ and $\boldsymbol{z}$, the conditional probability $p(\boldsymbol{x}|\boldsymbol{z})$ is also Gaussian,
> with mean $E[\boldsymbol{x}|\boldsymbol{z}] = E[\boldsymbol{x}] + \boldsymbol{P}_{xz}\boldsymbol{P}_{zz}^{-1}(\boldsymbol{z} - E[\boldsymbol{z}])$
> and covariance $\boldsymbol{P}_{xx|z} = \boldsymbol{P}_{xx} - \boldsymbol{P}_{xz}\boldsymbol{P}_{zz}^{-1}\boldsymbol{P}_{zx}$.

The proof of this theorem is not given here since it is rather long and tedious, but can be found in [19] on pages 52 through 54.

Note that we use the following naming convention in this thesis: $\bar{\boldsymbol{x}} \equiv E[\boldsymbol{x}]$, $\hat{\boldsymbol{x}} \equiv E[\boldsymbol{x}|\boldsymbol{z}]$, $\bar{\boldsymbol{P}} \equiv \boldsymbol{P}_{xx}$, $\boldsymbol{P} \equiv \boldsymbol{P}_{xx|z}$, $\boldsymbol{S} \equiv \boldsymbol{P}_{zz}$ and $\boldsymbol{C} \equiv \boldsymbol{P}_{xz}$. For each of the variables used in the rest of this thesis a subscript indicates the time to which it applies.

The theorem stated above immediately gives us the remaining equations of the Kalman Filter. To update the state estimate when a measurement becomes available:

$$\begin{aligned}\hat{\boldsymbol{x}}_k &= \bar{\boldsymbol{x}}_k + \boldsymbol{K}_k(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k) \\ &= \bar{\boldsymbol{x}}_k + \boldsymbol{C}_k\boldsymbol{S}_k^{-1}(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k)\end{aligned}$$

Hence

$$\boldsymbol{K}_k \equiv \boldsymbol{C}_k\boldsymbol{S}_k^{-1} = \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T(\boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}.$$

To update the state covariance matrix:

$$\begin{aligned}\boldsymbol{P}_k &= \bar{\boldsymbol{P}}_k - \boldsymbol{C}_k\boldsymbol{S}_k^{-1}\boldsymbol{C}_k^T \\ &= \bar{\boldsymbol{P}}_k - (\bar{\boldsymbol{P}}_k\boldsymbol{H}_t^T)(\boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}(\bar{\boldsymbol{P}}_k\boldsymbol{H}_t^T)^T \\ &= \bar{\boldsymbol{P}}_k - \boldsymbol{K}_k\boldsymbol{H}_k\bar{\boldsymbol{P}}_k^T \\ &= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\bar{\boldsymbol{P}}_k,\end{aligned}$$

which is the standard form used in the literature [16, 30, 19]. An alternative and symmetric equation form can be derived which, although equivalent, should be numerically advantageous [17].

$$\begin{aligned}\boldsymbol{P}_k &= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\bar{\boldsymbol{P}}_k \\ &= \bar{\boldsymbol{P}}_k - \boldsymbol{K}_k(\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T)^T \\ &= \bar{\boldsymbol{P}}_k - \boldsymbol{K}_k(\boldsymbol{C}_k)^T \\ &= \bar{\boldsymbol{P}}_k - \boldsymbol{K}_k(\boldsymbol{K}_k\boldsymbol{S}_k)^T \\ &= \bar{\boldsymbol{P}}_k - \boldsymbol{K}_k\boldsymbol{S}_k^T\boldsymbol{K}_k^T.\end{aligned}$$

## A.4 Optimality of the Kalman Filter

Given the Kalman update equation $\hat{\boldsymbol{x}}_k = \bar{\boldsymbol{x}}_k + \boldsymbol{K}_k(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k)$, we show that the choice of $\boldsymbol{K}_k = \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T(\boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T + \boldsymbol{R}_k)^{-1}$ minimizes the variances of the elements of the state error vector $\Delta\boldsymbol{x}_k = \hat{\boldsymbol{x}}_k - \boldsymbol{x}_k$.

The variances of the state error vector elements are found on the diagonal of the state covariance matrix $\boldsymbol{P}_k$. We must therefore minimize the trace of the matrix. Note that the off-diagonal covariances between different elements of the state vector are not minimized.

Now, we can (once again) rewrite the measurement update equation for the state covariance,

before taking partial derivatives. Some steps are omitted for brevity, but can be found in [19].

$$\begin{aligned}
\boldsymbol{P}_k &= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\bar{\boldsymbol{P}}_k \\
&= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\bar{\boldsymbol{P}}_k + \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T\boldsymbol{K}_k^T - \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T\boldsymbol{K}_k^T \\
&= \quad \vdots \\
&= (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)\bar{\boldsymbol{P}}_k(\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k)^T + \boldsymbol{K}_k\boldsymbol{R}_k\boldsymbol{K}_k^T \\
&= \quad \vdots \\
&= \bar{\boldsymbol{P}}_k - \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T\boldsymbol{K}_k^T - \boldsymbol{K}_k\boldsymbol{H}_k\bar{\boldsymbol{P}}_k + \boldsymbol{K}_k(\boldsymbol{R}_k + \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T)\boldsymbol{K}_k^T.
\end{aligned}$$

The derivative of a scalar $s$ with respect to a matrix $\boldsymbol{A}$ is the matrix

$$\left[\frac{\partial s}{\partial \boldsymbol{A}}\right]_{ij} = \left[\frac{\partial s}{\partial \boldsymbol{A}_{ij}}\right].$$

Furthermore:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{A}}(\mathrm{Tr}(\boldsymbol{B})) &= \boldsymbol{0} \qquad \text{if } \boldsymbol{B} \text{ does not depend on } \boldsymbol{A} \\
\frac{\partial}{\partial \boldsymbol{A}}(\mathrm{Tr}(\boldsymbol{A}\boldsymbol{B}^{\boldsymbol{T}})) &= \boldsymbol{B} \\
\frac{\partial}{\partial \boldsymbol{A}}(\mathrm{Tr}(\boldsymbol{B}\boldsymbol{A}^{\boldsymbol{T}})) &= \boldsymbol{B} \\
\frac{\partial}{\partial \boldsymbol{A}}(\mathrm{Tr}(\boldsymbol{A}\boldsymbol{B}\boldsymbol{A}^{\boldsymbol{T}})) &= \boldsymbol{A}(\boldsymbol{B} + \boldsymbol{B}^T).
\end{aligned}$$

Taking the derivative of the trace of $\boldsymbol{P}_k$ with relation to $\boldsymbol{K}_k$:

$$\begin{aligned}
\frac{\partial}{\partial \boldsymbol{K}_k}(\mathrm{Tr}(\boldsymbol{P}_k)) &= \boldsymbol{0} - \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T - (\boldsymbol{H}_k\bar{\boldsymbol{P}}_k)^T + \boldsymbol{K}_k\left((\boldsymbol{R}_k + \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T) + (\boldsymbol{R}_k + \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T)^T\right) \\
&= -\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T - \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T + \boldsymbol{K}_k\left((\boldsymbol{R}_k + \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T) + (\boldsymbol{R}_k + \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T)\right) \\
&= 2\left(\boldsymbol{K}_k(\boldsymbol{R}_k + \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T) - \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T\right) \\
&= \boldsymbol{0} \qquad \text{for minimum of maximum} \\
\Rightarrow\ \boldsymbol{K}_k &= \bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T(\boldsymbol{R}_k + \boldsymbol{H}_k\bar{\boldsymbol{P}}_k\boldsymbol{H}_k^T)^{-1}.
\end{aligned}$$

This is exactly the result we set out to prove. (Taking the second order derivative confirms that this is indeed a minimum and not a maximum).

## A.5 State Vector Augmentation

Although the Kalman Filter assumes Gaussian zero-mean white noise, we can model unknown system parameters and non-white noise sources, by passing white noise through a system with linear dynamics. The parameters of this system (for example the noise bias) are added as additional states and estimated together with the actual states. This is especially useful in systems where a significant bias may be present.

## A.6 Observability

Situations may arise in which there are not enough measurements to bound the state error covariance. These are referred to as observability problems.

Observability problems might be caused by trying to estimate states which are not expressed in the measurements being taken, or by having measurements of much lower dimension than the state being estimated. These problems may be addressed by reducing the number of state variables–i.e. removing states of little relevance from the model, or by adding additional sensors.

# Appendix B

# Derivation of the Discrete Extended Kalman Filter (EKF)

Since very few real-world problems can be accurately modeled by linear equations, the Kalman filter was quickly extended to be used for general nonlinear problems. A framework similar to the one for linear systems is desirable for a nonlinear system. Such an estimator can be obtained by a series expansion of the nonlinear dynamics and of the nonlinear measurement equations.

The following derivation follows a similar approach as the one provided in [19]. For a more thorough exposition we recommend [19, 16, 30].

## B.1 The system model

Consider a discrete-time *nonlinear* dynamic system described by a vector equation with additive white Gaussian noise. We assume, for simplicity, no control.

$$\boldsymbol{x}_{k+1} = f_k(\boldsymbol{x}_k) + \boldsymbol{v}_k$$

where $\boldsymbol{x}_k$ is the $n_x$-dimensional state vector and $\boldsymbol{v}_k$ is zero-mean white Gaussian process noise with covariance $E[\boldsymbol{v}_k \boldsymbol{v}_k^T] = \boldsymbol{Q}_k$.

The measurement equation is

$$\boldsymbol{z}_k = h_k(\boldsymbol{x}_k) + \boldsymbol{w}_k$$

where $\boldsymbol{z}_k$ is the $n_z$-dimensional measurement, and $\boldsymbol{w}_k$ is zero-mean white Gaussian measurement noise with covariance $E[\boldsymbol{w}_k \boldsymbol{w}_k^T] = \boldsymbol{R}_k$.

The initial state $\boldsymbol{x}_0$, generally unknown to us, is modeled as a Gaussian random variable with known mean and covariance $\boldsymbol{P}_0$.

### B.1.1 About the transition function

Often, the (possibly nonlinear) differential equations of a system are only known in their time-continuous form. An EKF is generally implemented in discrete time, and measurements taken at discrete intervals. We therefore have a similar situation with the EKF as we had in section §A.2.1 with the LKF.

By using an integration method of our choice, the state equation can be written in terms of the state's time derivative as follows:

$$\dot{\boldsymbol{x}} = g(\boldsymbol{x}(t), t) \qquad \Rightarrow \qquad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} g(\boldsymbol{x}(t), t)\mathrm{d}t + \boldsymbol{v}(t) \qquad \text{with } \boldsymbol{x}(t_k) = \boldsymbol{x}_k.$$

Both the discrete and continuous formulations will therefore lead to similar state prediction schemes (the latter by defining an equivalent but discrete update function in terms of an integration method).

In order to propagate the state error covariance (actually MSE) matrix (see section §B.2.2) we will need to look at the Taylor series expansion of the state update equation. We cannot directly use the series expansion as in section §A.2.1 since $g(\boldsymbol{x})$ is not linear in the elements of $\boldsymbol{x}$.

We can expand the state update equation as a Taylor series

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \dot{\boldsymbol{x}}\Delta t + \mathrm{HOT} + \boldsymbol{v}_k \\ &\approx \boldsymbol{x}_k + g(\boldsymbol{x}_k)\Delta t + \boldsymbol{v}_k \end{aligned}$$

with HOT referring to *higher order terms*. Now, given $\hat{\boldsymbol{x}}_k$, the "best" estimate at time $t_{k+1}$ using the above approximation, is

$$\bar{\boldsymbol{x}}_{k+1} = \hat{\boldsymbol{x}}_k + g(\hat{\boldsymbol{x}}_k)\Delta t.$$

The error in this prediction is

$$\begin{aligned} \Delta\bar{\boldsymbol{x}}_{k+1} &= \boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}_{k+1} \\ &\approx \boldsymbol{x}_k - \hat{\boldsymbol{x}}_k + \Delta t\left[g(\boldsymbol{x}_k) - g(\hat{\boldsymbol{x}}_k)\right] + \boldsymbol{v}_k \\ &\approx \Delta\hat{\boldsymbol{x}}_k + \Delta t\boldsymbol{G}_k\Delta\hat{\boldsymbol{x}}_k + \boldsymbol{v}_k \\ &= \left(\boldsymbol{I} + \Delta t\boldsymbol{G}_k\right)\Delta\hat{\boldsymbol{x}}_k + \boldsymbol{v}_k \end{aligned}$$

where

$$\boldsymbol{G}_k = \left.\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_k}.$$

The error covariance propagation can therefore be written in terms of the Jacobian matrix $\boldsymbol{G}_k$,

and calculated by

$$\begin{aligned} E[\Delta\hat{\boldsymbol{x}}_{k+1}\Delta\hat{\boldsymbol{x}}_{k+1}^T] &= E\left[\left[(\boldsymbol{I}+\Delta t\boldsymbol{G}_k)\Delta\hat{\boldsymbol{x}}_k+\boldsymbol{v}_k\right]\left[(\boldsymbol{I}+\Delta t\boldsymbol{G}_k)\Delta\hat{\boldsymbol{x}}_k+\boldsymbol{v}_k\right]^T\right] \\ &= E\left[\left[\boldsymbol{\Phi}_k\Delta\hat{\boldsymbol{x}}_k+\boldsymbol{v}_k\right]\left[\boldsymbol{\Phi}\Delta\hat{\boldsymbol{x}}_k+\boldsymbol{v}_k\right]^T\right] \\ &= E\left[\boldsymbol{\Phi}_k\Delta\hat{\boldsymbol{x}}_k\Delta\hat{\boldsymbol{x}}_k^T\boldsymbol{\Phi}^T+\boldsymbol{\Phi}_k\Delta\hat{\boldsymbol{x}}_k\boldsymbol{v}_k^T+\boldsymbol{v}_k\Delta\hat{\boldsymbol{x}}_k^T\boldsymbol{\Phi}^T+\boldsymbol{v}_k\boldsymbol{v}_k^T\right] \\ &= \boldsymbol{\Phi}_k E[\Delta\hat{\boldsymbol{x}}\Delta\hat{\boldsymbol{x}}^T]\boldsymbol{\Phi}_k^T+\boldsymbol{0}+\boldsymbol{0}+E[\boldsymbol{v}_k\boldsymbol{v}_k^T] \\ &= \boldsymbol{\Phi}_k\boldsymbol{P}_k\boldsymbol{\Phi}_k^T+\boldsymbol{Q}_k. \end{aligned}$$

The discrete covariance propagation matrix is thus approximated by

$$\boldsymbol{\Phi}_k = \boldsymbol{I}+\boldsymbol{G}_k\Delta t.$$

## B.2 The estimation algorithm

The estimation algorithm will take a very similar form to the linear Kalman filter. Note however that, since linearisations of the state and measurement equations will be used, the filter will no longer be strictly optimal. Therefore the estimation error will be *approximately* zero-mean. The covariance matrix $\boldsymbol{P}_k$ could more accurately be called the mean square error (MSE) matrix, since $\hat{\boldsymbol{x}}_k$ is no longer strictly the conditional mean.

### B.2.1 Predicting the state

The nonlinear system equation is expanded as a Taylor series about the latest estimate $\hat{\boldsymbol{x}}_k$. Only terms up to first order are considered for a first-order EKF, up to second order for a second-order EKF, and so forth. The vector Taylor series expansion of the state update equation is

$$\boldsymbol{x}_{k+1} = f_k(\hat{\boldsymbol{x}}_k)+\boldsymbol{F}_k\Delta\hat{\boldsymbol{x}}_k+\frac{1}{2}\sum_{i=1}^{n_x}\left[e_i\Delta\hat{\boldsymbol{x}}_k^T\boldsymbol{\Xi}_k\Delta\hat{\boldsymbol{x}}_k\right]+\text{HOT}+\boldsymbol{v}_k,$$

where $\Delta\hat{\boldsymbol{x}}_k = \boldsymbol{x}_k-\hat{\boldsymbol{x}}_k$ and $e_i$ is the $n_x$-dimensional vector with $i$th component unity and the rest zero. The Taylor series expansion can be truncated as follows:

$$\boldsymbol{x}_{k+1} \approx f_k(\hat{\boldsymbol{x}}_k)+\boldsymbol{F}_k\Delta\hat{\boldsymbol{x}}_k+\boldsymbol{v}_k,$$

Where

$$\Delta\hat{\boldsymbol{x}}_k = \boldsymbol{x}_k-\hat{\boldsymbol{x}}_k, \qquad \boldsymbol{F}_k = \left.\frac{\partial f_k(\boldsymbol{x})}{\partial\boldsymbol{x}}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_k} \quad \text{and} \quad \boldsymbol{\Xi}_k = \left.\frac{\partial^2 f_k(\boldsymbol{x})}{\partial\boldsymbol{x}^2}\right|_{\boldsymbol{x}=\hat{\boldsymbol{x}}_k}.$$

We will use this first-order approximation to derive the first-order EKF.

Note that both the Jacobian $\boldsymbol{F}_k$ (and the Hessian $\boldsymbol{\Xi}_k$) are deterministic quantities. Only $\boldsymbol{x}_k$ and $\boldsymbol{v}_x$ are random variables. These properties are important when considering conditional expec-

tations.

We can now write the predicted state

$$\begin{aligned}
\bar{\boldsymbol{x}}_{k+1} &\approx E[f_k(\hat{\boldsymbol{x}}_k) + \boldsymbol{F}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{v}_k] \\
&= E[f_k(\hat{\boldsymbol{x}}_k)] + E[\boldsymbol{F}_k \Delta\hat{\boldsymbol{x}}_k] + E[\boldsymbol{v}_k] \\
&= f_k(\hat{\boldsymbol{x}}_k) + \boldsymbol{F}_k E[(\Delta\hat{\boldsymbol{x}}_k)] + \boldsymbol{0} \\
&\approx f_k(\hat{\boldsymbol{x}}_k)
\end{aligned}$$

Note that the prediction is no longer perfectly unbiased as with the LKF, but only *approximately* unbiased. This is due to the fact that the EKF is no longer an optimal filter but a linearisation of an essentially nonlinear problem.

### B.2.2 Predicting the state (error) covariance

$$\begin{aligned}
\bar{\boldsymbol{P}}_{k+1} &= E[(\boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}_{k+1})(\boldsymbol{x}_{k+1} - \bar{\boldsymbol{x}}_{k+1})^T] \\
&\approx E[(f_k(\hat{\boldsymbol{x}}_k) + \boldsymbol{F}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{v}_k - f_k(\hat{\boldsymbol{x}}_k))\,(f_k(\hat{\boldsymbol{x}}_k) + \boldsymbol{F}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{v}_k - f_k(\hat{\boldsymbol{x}}_k))^T] \\
&= E[(\boldsymbol{F}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{v}_k)\,(\boldsymbol{F}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{v}_k)^T] \\
&= \boldsymbol{F}_k E[\Delta\hat{\boldsymbol{x}}_k \Delta\hat{\boldsymbol{x}}_k^T] \boldsymbol{F}_k^T + \boldsymbol{F}_k E[\Delta\hat{\boldsymbol{x}}_k \boldsymbol{v}_k^T] + E[\boldsymbol{v}_k \Delta\hat{\boldsymbol{x}}_k^T] \boldsymbol{F}_k^T + E[\boldsymbol{v}_k \boldsymbol{v}_k^T] \\
&= \boldsymbol{F}_k \boldsymbol{P}_k \boldsymbol{F}_k^T + \boldsymbol{F}_k \boldsymbol{0} + \boldsymbol{0} \boldsymbol{F}_k^T + \boldsymbol{Q}_k \\
&= \boldsymbol{F}_k \boldsymbol{P}_k \boldsymbol{F}_k^T + \boldsymbol{Q}_k
\end{aligned}$$

Note that this is almost identical to the state covariance update of the LKF, except for that the role of the transition matrix $\boldsymbol{\Phi}_k$ has been replaced by the system Jacobian matrix $\boldsymbol{F}_k$. This initially sounds a bit strange, since left-multiplying the state by the Jacobian matrix does not approximate the time propagation of the state vector, as with the LKF. Rather, the left-multiplying the state error by the Jacobian approximates the time propagation of the state error, and that is exactly what we are doing when we propagate the state (error) covariance.

Strictly speaking, $\boldsymbol{P}_{k+1}$ is a mean square error (MSE) matrix rather than a covariance matrix, since $\hat{\boldsymbol{x}}_{k+1}$ is no longer the exact conditional mean but an approximation.

### B.2.3 Predicting the measurement

Like with the LKF, we now have the prior estimate for the state vector at time step $k$, namely $\bar{\boldsymbol{x}}_k$, with covariance $\bar{\boldsymbol{P}}_k$. We are now dealing with a nonlinear measurement function $h_k(x)$, and

similarly to the state prediction we use a Taylor series expansion.

$$\begin{aligned} \boldsymbol{z}_k &= h_k(\bar{\boldsymbol{x}}_k) + \boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \text{HOT} + \boldsymbol{w}_k \\ &\approx h_k(\bar{\boldsymbol{x}}_k) + \boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{w}_k \end{aligned}$$

Where

$$\Delta\hat{\boldsymbol{x}}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k, \qquad \boldsymbol{H}_k = \left.\frac{\partial h_k(\boldsymbol{x})}{\partial \boldsymbol{x}}\right|_{\boldsymbol{x}=\bar{\boldsymbol{x}}_k}.$$

The expansion is only shown up to its linear terms, but we will only use this first-order approximation to derive the first-order EKF.

$$\begin{aligned} \bar{\boldsymbol{z}}_k = E[\boldsymbol{z}_k] &\approx E[h_k(\bar{\boldsymbol{x}}_k) + \boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{w}_k] \\ &= E[h_k(\bar{\boldsymbol{x}}_k)] + E[\boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k] + E[\boldsymbol{w}_k] \\ &= h_k(\bar{\boldsymbol{x}}_k) + \boldsymbol{H}_k E[\Delta\hat{\boldsymbol{x}}_k] + \boldsymbol{0} \\ &\approx h_k(\bar{\boldsymbol{x}}_k). \end{aligned}$$

### B.2.4 Predicting the measurement covariance

The innovation of the EKF measurement covariance now can be computed as follows

$$\begin{aligned} \boldsymbol{S}_k &= E[(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k)(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k)^T] \\ &\approx E[(h_k(\bar{\boldsymbol{x}}_k) + \boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{w}_k - h_k(\bar{\boldsymbol{x}}_k))\,(h_k(\bar{\boldsymbol{x}}_k) + \boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{w}_k - h_k(\bar{\boldsymbol{x}}_k))^T] \\ &= E[(\boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{w}_k)\,(\boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{w}_k)^T] \\ &= \boldsymbol{H}_k E[\Delta\hat{\boldsymbol{x}}_k \Delta\hat{\boldsymbol{x}}_k^T]\boldsymbol{H}_k^T + \boldsymbol{H}_k E[\Delta\hat{\boldsymbol{x}}_k \boldsymbol{w}_k^T] + E[\boldsymbol{w}_k \Delta\hat{\boldsymbol{x}}_k^T]\boldsymbol{H}_k^T + E[\boldsymbol{w}_k \boldsymbol{w}_k^T] \\ &= \boldsymbol{H}_k \bar{\boldsymbol{P}}_k \boldsymbol{H}_k^T + \boldsymbol{H}_k \boldsymbol{0} + \boldsymbol{0}\boldsymbol{H}_k^T + \boldsymbol{R}_k \\ &= \boldsymbol{H}_k \bar{\boldsymbol{P}}_k \boldsymbol{H}_k^T + \boldsymbol{R}_k. \end{aligned}$$

### B.2.5 Predicting the cross covariance

The cross covariance between the predicted state and the predicted measurement is analogous to the LKF

$$\begin{aligned} \boldsymbol{C}_k &= E[(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k)(\boldsymbol{z}_k - \bar{\boldsymbol{z}}_k)^T] \\ &= E[\Delta\hat{\boldsymbol{x}}_k (h_k(\bar{\boldsymbol{x}}_k) + \boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{w}_k - h_k(\bar{\boldsymbol{x}}_k))^T] \\ &= E[\Delta\hat{\boldsymbol{x}}_k (\boldsymbol{H}_k \Delta\hat{\boldsymbol{x}}_k + \boldsymbol{w}_k)^T] \\ &= E[\Delta\hat{\boldsymbol{x}}_k \Delta\hat{\boldsymbol{x}}_k^T \boldsymbol{H}_k^T + \Delta\hat{\boldsymbol{x}}_k \boldsymbol{w}_k^T] \\ &= \bar{\boldsymbol{P}}_k \boldsymbol{H}_k^T + \boldsymbol{0} \\ &= \bar{\boldsymbol{P}}_k \boldsymbol{H}_k^T. \end{aligned}$$

### B.2.6 Incorporating measurements

The incorporation of measurements by the EKF is identical to the way in which it is handled by the LKF.

### B.2.7 A word about the stability of the EKF

The use of the series expansion in the state prediction and/or in the measurement prediction can possibly introduce errors that violate some basic assumptions about the prediction errors. In general, a nonlinear transformation will introduce a bias and the covariance calculation based on a series expansion will not always be accurate. Also, the fact that these expansions rely on Jacobians (and Hessians in a second order filter) that are evaluated at the *estimated* or *predicted* states, can cause errors. Such conditions may cause the EKF to diverge. The EKF does, however, perform well if the initial error and noises are not too large, and the system not too severely nonlinear.

# Appendix C

# Homogeneous coordinates

A well-written introduction to projective geometries, including homogeneous coordinates, can be found in [31].

To summarize:
Homogeneous coordinates present us with a scale-invariant way of representing vectors of dimension $n$, by expressing them as vectors with dimension $n+1$, but retaining $n$ degrees of freedom.

For example: to represent an Euclidean vector $[X,\ Y]^T$ in homogeneous coordinates we simply add third coordinate, 1, at the end. Overall scaling is unimportant, so that

$$\begin{aligned} [X,\ Y]^T &\Rightarrow [X,\ Y,\ 1]^T \\ &\equiv [\alpha X,\ \alpha Y,\ \alpha]^T, \quad \alpha \neq 0. \end{aligned}$$

The name "*homogeneous coordinates*" have its origin in the fact that the new vector is scale invariant. Homogeneous coordinates are very useful for working with abstract concepts in projective geometry, such as points and lines at infinity [31]. For our application we can express the non-linear perspective projection (section §2.1) as a linear equation in homogeneous coordinates. This is an advantage to the implementation of algorithms, since we can now use linear algebra for these equations.

# Appendix D

# Attitude representation

A good survey of different attitude representations, with specific applications to satellite dynamics can be found in [9].

Let's consider a rigid body, with an orthogonal right-handed triad $\hat{\boldsymbol{u}}$, $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{w}}$ of unit vectors fixed to the body (the Object Coordinate System). The basic problem is to specify the orientation of this triad, and hence the rigid body, relative to some reference coordinate frame (in our case the Camera Coordinate System).

## D.1 Direction Cosine Matrix (DCM)

It is clear that specifying the three unit vectors $\hat{\boldsymbol{u}}$, $\hat{\boldsymbol{v}}$ and $\hat{\boldsymbol{w}}$ along the axes of the CCS will completely describe the orientation of the OCS relative to the CCS. This requires 9 elements, which can be written as the elements of a $3 \times 3$ matrix $\boldsymbol{A}$, called the attitude matrix.

$$\boldsymbol{A} = \begin{bmatrix} \hat{\boldsymbol{u}}_x & \hat{\boldsymbol{v}}_x & \hat{\boldsymbol{w}}_x \\ \hat{\boldsymbol{u}}_y & \hat{\boldsymbol{v}}_y & \hat{\boldsymbol{w}}_y \\ \hat{\boldsymbol{u}}_z & \hat{\boldsymbol{v}}_z & \hat{\boldsymbol{w}}_z \end{bmatrix}$$

Each of the elements is the cosine of the angle between a body unit vector and one of the reference axes, and therefore $\boldsymbol{A}$ is usually called the *Direction Cosine Matrix* (DCM). The elements are not all independent. Since each row is a unit vector, the rows are linearly independent. In fact the DCM has only three degrees of freedom, and is a real orthonormal matrix. The DCM therefore has the following properties:

$$\begin{aligned} \boldsymbol{x}_{ccs} &= \boldsymbol{A}\boldsymbol{x}_{ocs} \quad , \quad \text{with } \boldsymbol{x}_{ccs} \text{ and } \boldsymbol{x}_{ocs} \text{ 3D position vectors} \\ \boldsymbol{A}\boldsymbol{A}^T &= \boldsymbol{I} \\ \det(\boldsymbol{A}) &= 1 \\ \text{Eigenvalues of } \boldsymbol{A} &\Rightarrow 1,\ \cos(\theta) + i\sin(\theta),\ \cos(\theta) - i\sin(\theta). \end{aligned}$$

Two successive DCM rotations $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are combined as follows:

$$\boldsymbol{A}_{total} = \boldsymbol{A}_2\boldsymbol{A}_1.$$

(Note the order, since the vector being rotated is multiplied from the right)

## D.2 Euler Angles

The idea of Euler angles is to split the complete rotation of a cartesian coordinate system into three simpler rotations about the axes of this system. The feasibility of this approach is guaranteed by Euler's displacement theorem. Euler angles are not as convenient for numerical computations as some of the other parametrizations, but their geometrical significance is more apparent, and are often used for computer input/output.

Unfortunately, their definition is not unique and in the literature there are as many different conventions as authors. These conventions depend on the axes about which the rotations are carried out, and the order (since rotations are not commutative). There are in fact 24 possible definitions for Euler Angle rotations about a perpendicular right-handed coordinate frame – which makes it a messy subject.[1] Also note the difference in whether the rotated target is expressed in terms of the external frame, or whether the external frame is expressed in terms of the rotated body frame. *Here we express the rotated target in terms of the external frame.*

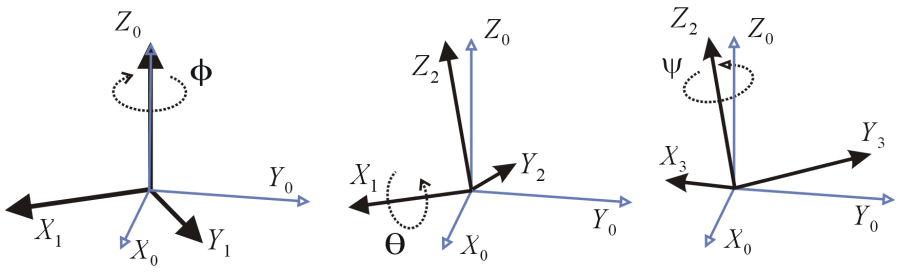



**Figure D.1:** General convention 3-1-3 Euler Angles with $(\phi, \theta, \psi) = (-60°, 30°, 45°)$

The so-called "general convention", illustrated in figure figure D.1, rotates about successive body-fixed axes. For example: The first rotation $\phi$ is about the $Z_0$-axis (parallel to the $Z$-axis), the second angle $\theta$ about the *new* $X$-axis (denoted as $X_1$) and finally the third angle $\psi$ about the *new* $Z$-axis ($Z_2$). This produces an *identical rotation* as rotating about the original fixed axes by the same angles but in *reverse order* – see below.

[1] The engineering and robotics communities usually use 3-1-3 Euler Angles. Other terminology (e.g., Fick angles, Helmholtz angles, roll-pitch-yaw) are used for other conventions. Computer graphics papers tend to confuse the matter by either defining Euler Angles as XYZ, or by using the term more loosely, to describe any 3-angle parameterization

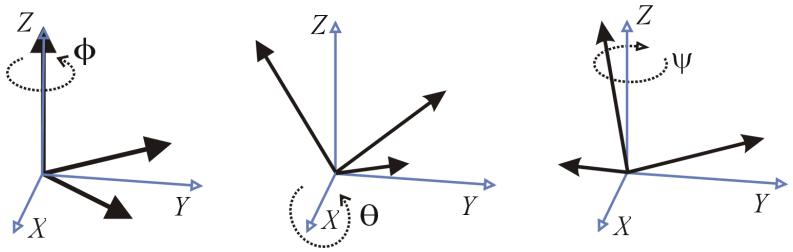



**Figure D.2:** x-convention 3-1-3 Euler Angles with $(\phi, \theta, \psi) = (45°, 30°, -60°)$

The so-called "x-convention," illustrated in figure figure D.2, uses the original fixed reference axes as the axes of rotation (not the body axes). The first rotation $\phi$ is about the $Z$-axis, then a second rotation $\theta$ about the $X$-axis and finally a third rotation $\psi$ about the $Z$-axis (again). $\theta$ is usually restricted to $-90° \leq \theta < 90°$.

The "general" convention is the most common definition according to [32], especially in the aeronautical literature. Some authors claim that the x-convention is more popular [33], although we disagree. However, the x-convention is more intuitive in the mathematical sense, when observing a rotating target from an external reference frame.

Combining two successive rotations, each represented by a set of Euler angles, is not straightforward. It is usual to convert to DCM notation or quaternions, calculate the product, and then convert back to Euler angles. If this is done many times in succession the rounding errors of all these conversions can build up, and is therefore not a very appropriate method for updating global rotations.

## D.3 Euler Axis/Angle (rotation vector)

One of the properties of the DCM is that one of its eigenvalues is unity (see section §D.1). This means that there exists a unit vector $\hat{\boldsymbol{e}}$ which is unchanged by being rotated by the DCM $\boldsymbol{A}$. The only axis with this property is the axis of rotation.

The existence of $\hat{\boldsymbol{e}}$ (as well as the Euler angle representation of section §D.2) demonstrate Euler's displacement theorem [33], which states:
*"The general displacement of a rigid body (or coordinate frame) with one point fixed, is a rotation about some axis. Furthermore, a rotation may be described in any basis using three angles."*

Therefore we can represent a rotation by a three-dimensional unit vector – the axis $\hat{\boldsymbol{e}}$, and an angle $\theta$ (see figure D.3). There are only three independent parameters, since $\|\hat{\boldsymbol{e}}\| = 1$.

The *rotation vector* parametrization is written as a 3-vector with the angle being the magnitude of the vector, and the axis a unit vector in the given direction

$$\boldsymbol{a}_\theta = \theta \hat{\boldsymbol{e}}.$$

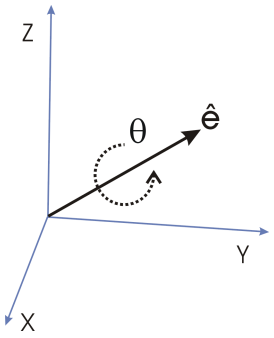



**Figure D.3:** Interpretation of an Euler Axis and Angle rotation

## D.4 Quaternions

Quaternions (Euler Symmetric Parameters) have proven very useful in representing rotations. In terms of the Euler axis $\hat{\boldsymbol{e}} = [e_x \; e_y \; e_z]^T$ and angle $\theta$ they are expressed as follow [2]

$$\begin{aligned} q_1 &= e_x \sin(\theta/2) \\ q_2 &= e_y \sin(\theta/2) \\ q_3 &= e_z \sin(\theta/2) \\ q_4 &= \cos(\theta/2) \end{aligned}$$

Inspection shows that the quaternion parametrization obeys the following constraint:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1.$$

The last term (in our definition) is often called the scalar term, which has its origin in quaternions when understood as the mathematical extension of the complex numbers, written as

$$a + bi + cj + dk.$$

$\{a, b, c, d\} \in \mathbb{R}$, and $\{i, j, k\}$ are the hyperimaginary numbers satisfying

$$\begin{aligned} i^2 = j^2 = k^2 = -1 \\ ij = -ji = k \\ jk = -kj = i \\ ki = -ik = j \end{aligned}$$

[2] We use the same definition as used in [9, 10, 27]. Other authors sometimes use a different definition with the "scalar" term as the first quaternion element, with the other elements shifted down one position [34, 28]

Quaternion multiplication is performed in the same manner as multiplication of complex numbers, except that the order of elements must be taken into account, since multiplication is not commutative. In matrix notation we can write quaternion multiplication as

$$\tilde{\boldsymbol{q}} \otimes \boldsymbol{q} = \begin{bmatrix} q_4 & q_3 & -q_2 & q_1 \\ -q_3 & q_4 & q_1 & q_2 \\ q_2 & -q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix} \begin{bmatrix} \tilde{q}_1 \\ \tilde{q}_2 \\ \tilde{q}_3 \\ \tilde{q}_4 \end{bmatrix}.$$

Combining two consecutive quaternion rotations is therefore just as simple as using DCMs. Two successive DCM rotations $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ are combined as follows:

$$\boldsymbol{A}_3 = \boldsymbol{A}_2 \boldsymbol{A}_1.$$

We can do the same in quaternion parameters:[3]

$$\boldsymbol{q}_3 = \boldsymbol{q}_1 \otimes \boldsymbol{q}_2.$$

Quaternions are a very popular parametrization due to the following properties:

- More compact than the DCM representation and less susceptible to roundoff errors
- The quaternion elements vary continuously over the unit sphere in $\mathbb{R}^4$, (denoted by $S^3$) as the attitude changes, avoiding discontinuous jumps (inherent to three-dimensional parametrizations)
- Expression of the DCM in terms of quaternion parameters involves no trigonometric functions
- Simplicity of combining two individual rotations by means of a quaternion product

Like DCM's, quaternions must sometimes be re-normalized due to rounding errors, to make sure that they correspond to valid rotations. The cost of re-normalizing a quaternion, however, is much less than for normalizing a $3 \times 3$ DCM.

Coutsias and Romero discuss the application of quaternion algebra to rigid body dynamics in [34].

## D.5 Other Parametrizations

Some other parametrizations used in the literature [9, 27], include the Gibbs Vector (also called Rodriguez parameters), and Modified Rodriguez Parameters (MRPs).

[3] Please note the inverse order of quaternion multiplication when compared to DCM multiplication.

The *Gibbs vector* can be expressed in terms of Euler axis and angle (section §D.3) as follows:

$$\boldsymbol{g} = \hat{\boldsymbol{e}} \tan(\theta/2).$$

The Gibbs vector is undefined for 180° rotations, which is undesirable for global attitude representation.

*Modified Rodriguez parameters* (MRPs) can be expressed in terms of Euler axis and angle (section §D.3) by:

$$\boldsymbol{p} = \hat{\boldsymbol{e}} \tan(\theta/4).$$

This parametrization shares many characteristics with the rotation vector parametrization (section §D.3), including the need for discrete jumps.

These alternative parametrizations have been used in modified versions of attitude estimation filters [27], but their use is limited and will not be further discussed here.

## D.6 Conversion Formulas

### D.6.1 DCM ⇒ Euler angles

The Euler angles $(\phi, \theta, \psi)$ can be extracted from the DCM $\boldsymbol{A}$ by inspecting the combined DCM in analytical form.

Using the the x-convention (section §D.2), the 3-1-3 Euler angles $\phi$, $\theta$ and $\psi$ (around the $Z$,$X$ and again the $Z$-axis) can be obtained as follows[4]:

$$\begin{aligned} \phi &= \arctan(A_{31}, A_{32}), \\ \theta &= \arccos(A_{33}), \\ \psi &= -\arctan(A_{13}, A_{23}). \end{aligned}$$

### D.6.2 DCM ⇒ Euler Axis/Angle

If the Euler angle $\theta$ is not a multiple of 180°, the Euler axis $\hat{\boldsymbol{e}} = [e_1\ e_2\ e_3]^T$ and $\theta$ can be computed from the elements of the DCM $\boldsymbol{A}$ as follow:

$$\begin{aligned} e_1 &= (A_{23} - A_{32})/(2\sin\theta), \\ e_2 &= (A_{31} - A_{13})/(2\sin\theta), \\ e_3 &= (A_{12} - A_{21})/(2\sin\theta), \\ \theta &= \arccos\left((A_{11} + A_{22} + A_{33} - 1)/2\right). \end{aligned}$$

Alternatively, the following method can be used:
Eigen-decomposition of the DCM yields the eigenvalues 1, and $\cos\theta \pm i\sin\theta$. The Euler axis is the eigenvector corresponding to the eigenvalue 1, and the Euler angle can be computed from the remaining eigenvalues.

### D.6.3 DCM ⇒ Quaternion

When computing a quaternion from the DCM parametrization there is a sign abiguity, since $\boldsymbol{q}$ and $-\boldsymbol{q}$ represents the same rotation. One way of computing the quaternion $\boldsymbol{q} = [q_1\ q_2\ q_3\ q_4]^T$

[4] $\arctan(a, b)$ is equivalent to $\arctan(a/b)$ while also taking into account which quadrant the point $(a, b)$ is in.

from the DCM $\boldsymbol{A}$ as follows:

$$\begin{aligned} q_4 &= \pm\frac{1}{2}\sqrt{1 + A_{11} + A_{22} + A_{33}}, \\ q_1 &= \frac{1}{4q_4}(A_{23} - A_{32}), \\ q_2 &= \frac{1}{4q_4}(A_{31} - A_{13}), \\ q_3 &= \frac{1}{4q_4}(A_{12} - A_{21}). \end{aligned}$$

There are three other mathematically equivalent ways to compute $\boldsymbol{q}$. Numerical inaccuracy can be reduced by avoiding situations in which the denominator (in this case $q_4$) is close to zero. [5] One of the other three methods looks as follows:

$$\begin{aligned} q_1 &= \pm\frac{1}{2}\sqrt{1 + A_{11} - A_{22} - A_{33}}, \\ q_2 &= \frac{1}{4q_1}(A_{12} + A_{21}), \\ q_3 &= \frac{1}{4q_1}(A_{13} + A_{31}), \\ q_4 &= \frac{1}{4q_1}(A_{23} - A_{32}). \end{aligned}$$

### D.6.4 Euler Angles ⇒ DCM

We will consider the Euler Angle x-convention (section §D.2) for the following algorithm.

The DCM of the complete rotation, $\boldsymbol{A}$, is the matrix product of the three matrices for the individual rotations. The first rotation's matrix is on the right and the last on the left[6] (since we left-multiply the orientation vector with $\boldsymbol{A}$).

$$\boldsymbol{A} = \boldsymbol{A}_3\boldsymbol{A}_2\boldsymbol{A}_1.$$

The axes of the rotation depend on the specific order chosen for the application. For rotations

[5] For small angles $q_4 \approx 1$.

[6] If we use the general convention instead of the x-convention, the order of multiplication is reversed

about the $X$, $Y$ and $Z$ axes with angles $\phi$, $\theta$ and $\psi$, the individual matrices are as follows:

$$\begin{aligned}
\boldsymbol{A}_X &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}, \\
\boldsymbol{A}_Y &= \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \\
\boldsymbol{A}_Z &= \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}.
\end{aligned}$$

### D.6.5 Euler Angles ⇒ Quaternion

We will consider the x-convention 3-1-3 Euler Angles (section §D.2) for the following algorithm.

We can compute the quaternion $\boldsymbol{q} = [q_1\ q_2\ q_3\ q_4]^T$ from the Euler angles $(\phi, \theta, \psi)$ as follows:

$$\begin{aligned}
q_1 &= -\cos\left((\phi-\psi)/2\right)\sin(\theta/2), \\
q_2 &= \sin\left((\phi-\psi)/2\right)\sin(\theta/2), \\
q_3 &= -\sin\left((\phi+\psi)/2\right)\cos(\theta/2), \\
q_4 &= \cos\left((\phi+\psi)/2\right)\cos(\theta/2).
\end{aligned}$$

### D.6.6 Euler Axis/Angle ⇒ DCM

The DCM corresponding to an Euler axis $\hat{\boldsymbol{e}} = [e_1\ e_2\ e_3]^T$ and angle $\theta$ can be computed as follows:

$$\boldsymbol{A} = \boldsymbol{I}_3\cos\theta + (1-\cos\theta)\hat{\boldsymbol{e}}\hat{\boldsymbol{e}}^T - \boldsymbol{\mathcal{E}}\sin\theta$$

with $\boldsymbol{I}_3$ the $3\times 3$ identity matrix, and

$$\boldsymbol{\mathcal{E}} = \begin{bmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{bmatrix}.$$

### D.6.7 Euler Axis/Angle ⇒ Quaternion

Given the Euler axis $\hat{\boldsymbol{e}}$ and angle $\theta$, the quaternion $\boldsymbol{q}$ can be computed by

$$\begin{aligned} q_1 &= \hat{e}_1 \sin(\theta/2), \\ q_2 &= \hat{e}_2 \sin(\theta/2), \\ q_3 &= \hat{e}_3 \sin(\theta/2), \\ q_4 &= \cos(\theta/2). \end{aligned}$$

### D.6.8 Quaternion ⇒ DCM

The DCM corresponding to the quaternion $\boldsymbol{q} = [q_1\ q_2\ q_3\ q_4]^T$ can be computed as follows:

$$\boldsymbol{A} = (q_4^2 - \check{\boldsymbol{q}}^T\check{\boldsymbol{q}})\boldsymbol{I}_3 + 2\check{\boldsymbol{q}}\check{\boldsymbol{q}}^T - 2q_4\boldsymbol{\mathcal{Q}}$$

with $\boldsymbol{I}_3$ the $3 \times 3$ identity matrix, and

$$\check{\boldsymbol{q}} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix}, \qquad \boldsymbol{\mathcal{Q}} = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix}.$$

### D.6.9 Quaternion ⇒ Euler Angles

Given the quaternion $\boldsymbol{q} = [q_1\ q_2\ q_3\ q_4]^T$, the x-convention 3-1-3 Euler angles $(\phi, \theta, \psi)$ can be computed by

$$\begin{aligned} \phi &= \arctan((q_1q_3 + q_2q_4), (q_2q_3 - q_1q_4)), \\ \theta &= \arccos(-q_q^2 - q_2^2 + q_3^2 + q_4^2), \\ \psi &= -\arctan((q_1q_3 - q_2q_4), (q_2q_3 + q_1q_4)). \end{aligned}$$

### D.6.10 Quaternion ⇒ Euler Axis/Angle

Given the quaternion $\boldsymbol{q} = [q_1\ q_2\ q_3\ q_4]^T$, define $\check{\boldsymbol{q}} = [q_1\ q_2\ q_3]^T$. Then the Euler axis $\hat{\boldsymbol{e}}$ and angle $\theta$ can be computed by

$$\begin{aligned} \hat{\boldsymbol{e}} &= \frac{\check{\boldsymbol{q}}}{\|\check{\boldsymbol{q}}\|}, \\ \theta &= 2\arccos(q_4). \end{aligned}$$

# Appendix E

# Additional Results

This appendix contains some additional graphs pertaining to chapters 6 and 7, as well as some information concerning initialization matrices. Please note that even this is not a complete list of results. Due to space considerations many of the results are still included only in electronic format on the accompanying CD, as described by appendix F.

## E.1 Simulated Data

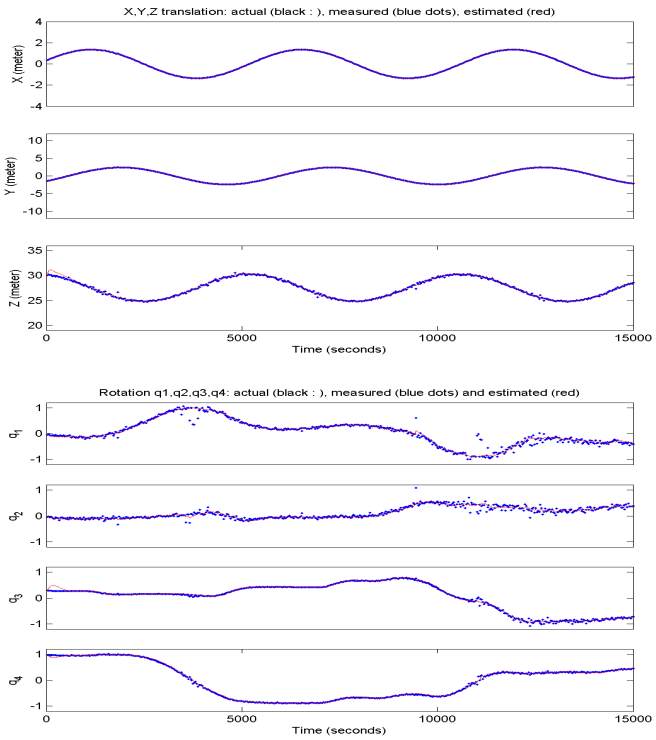



**Figure E.1:** Simulated: Calibration and UKF – translation and rotation

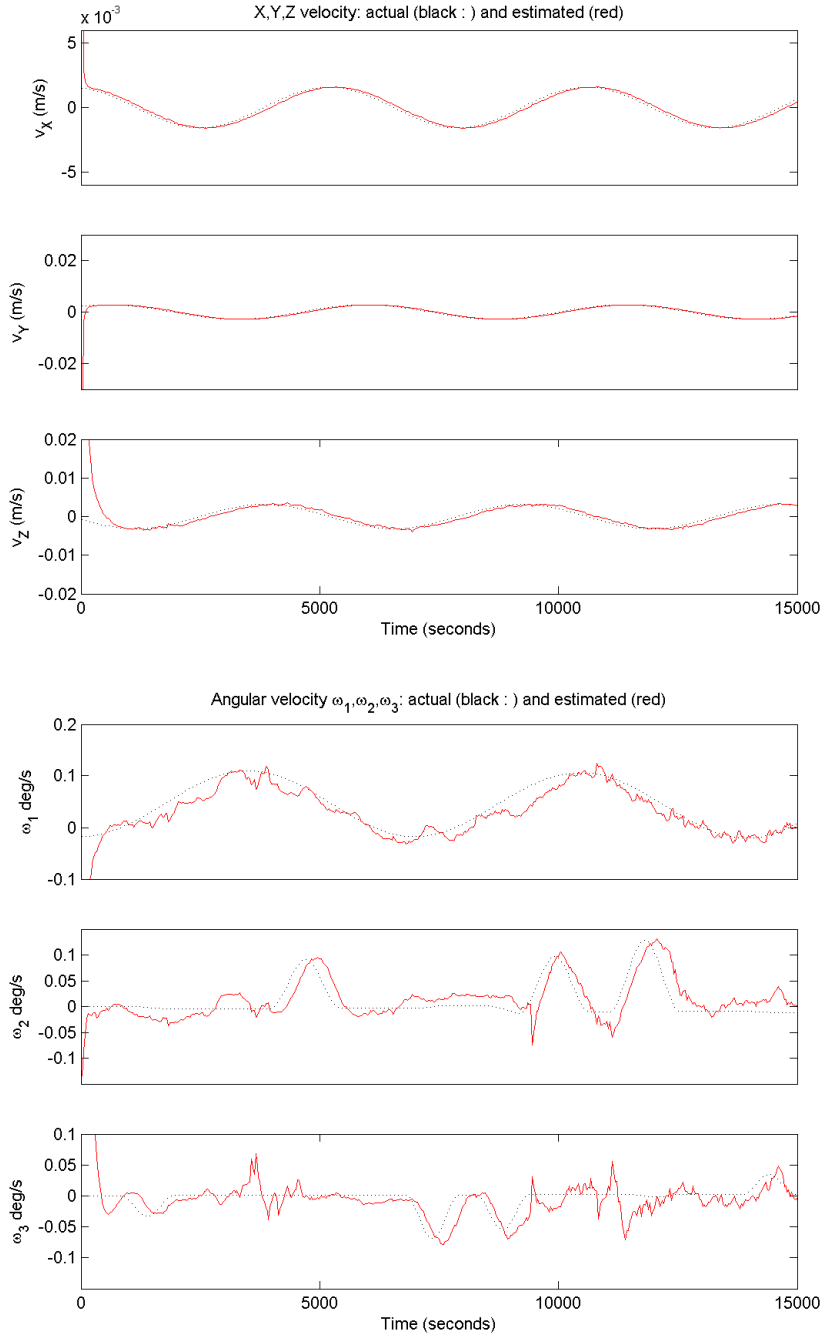



**Figure E.2:** Simulated: Calibration and UKF – linear and angular velocities

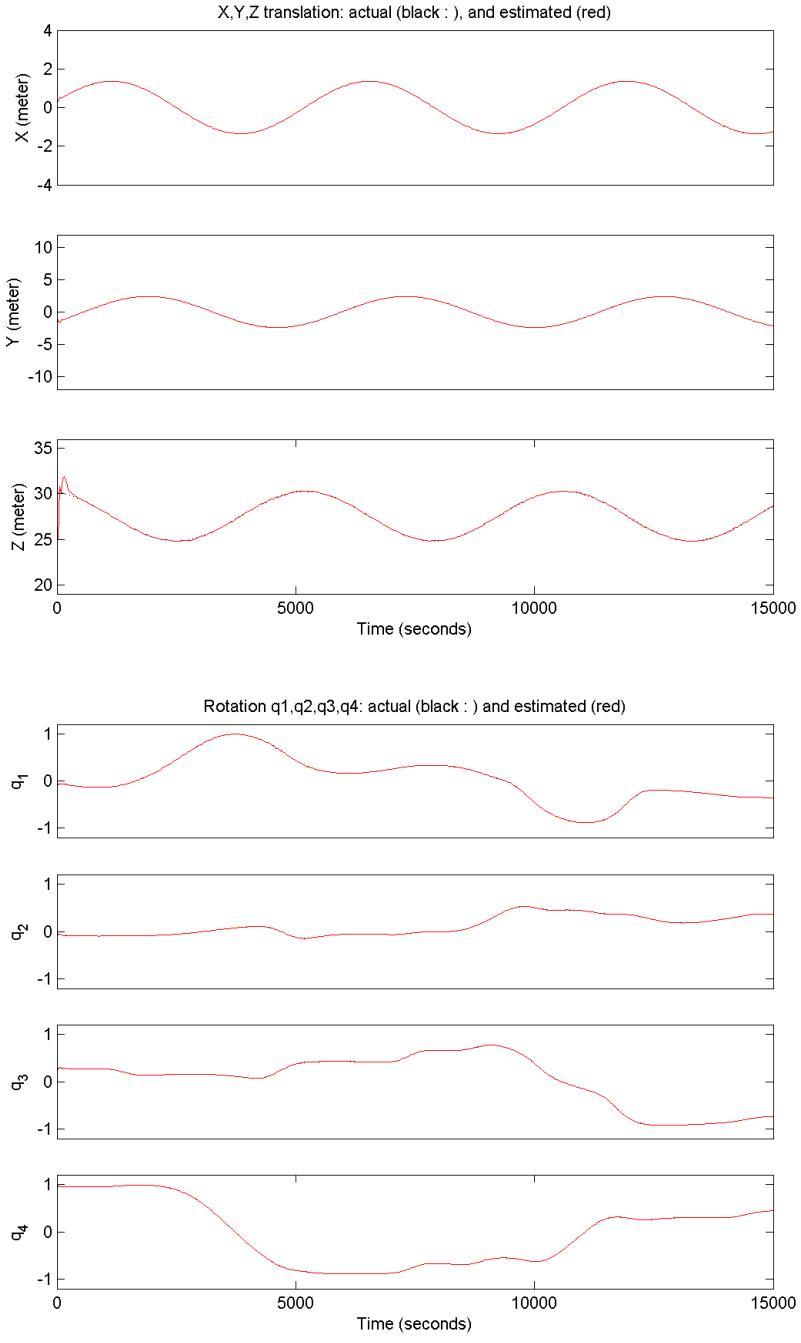



**Figure E.3:** Simulated: Single EKF with concatenated features – translation and rotation

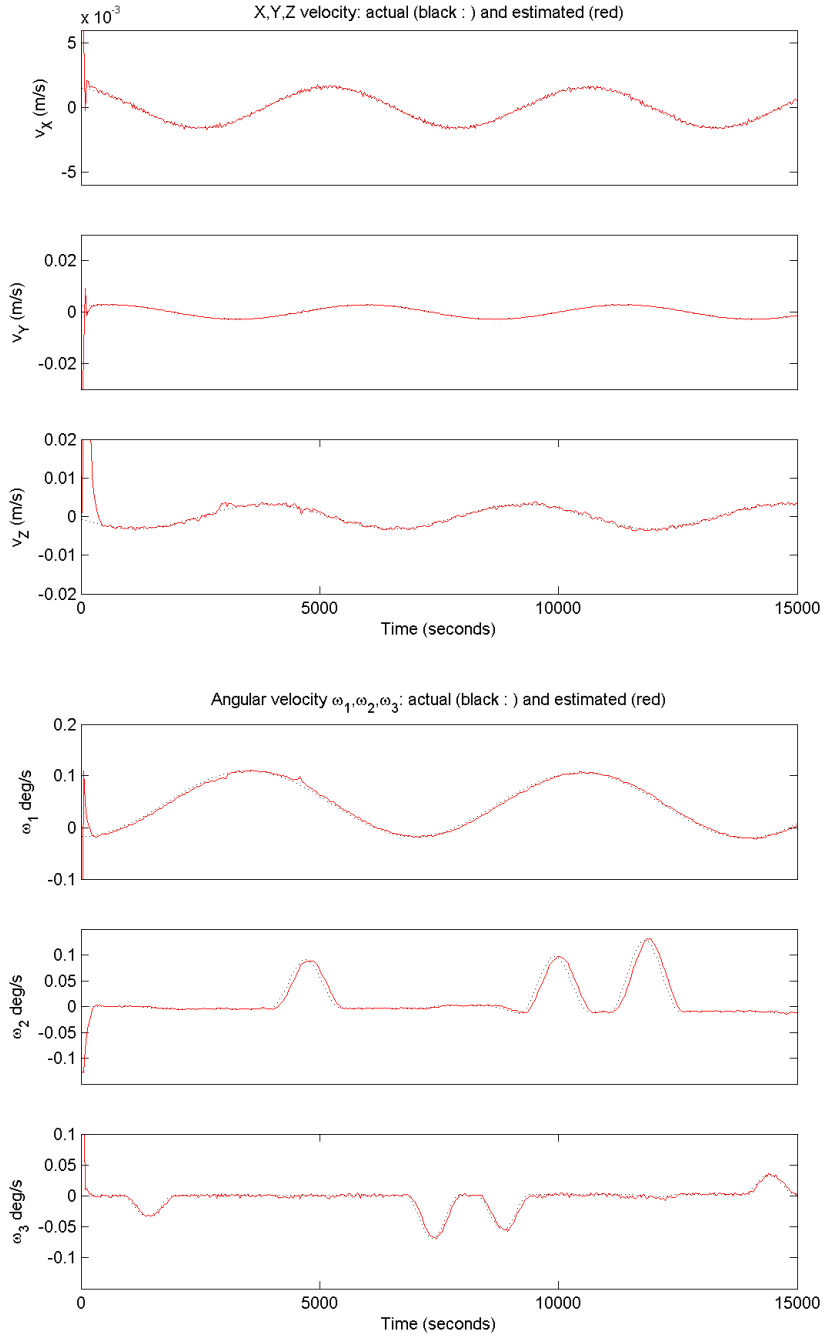



**Figure E.4:** Simulated: Single EKF with concatenated features – linear and angular velocities

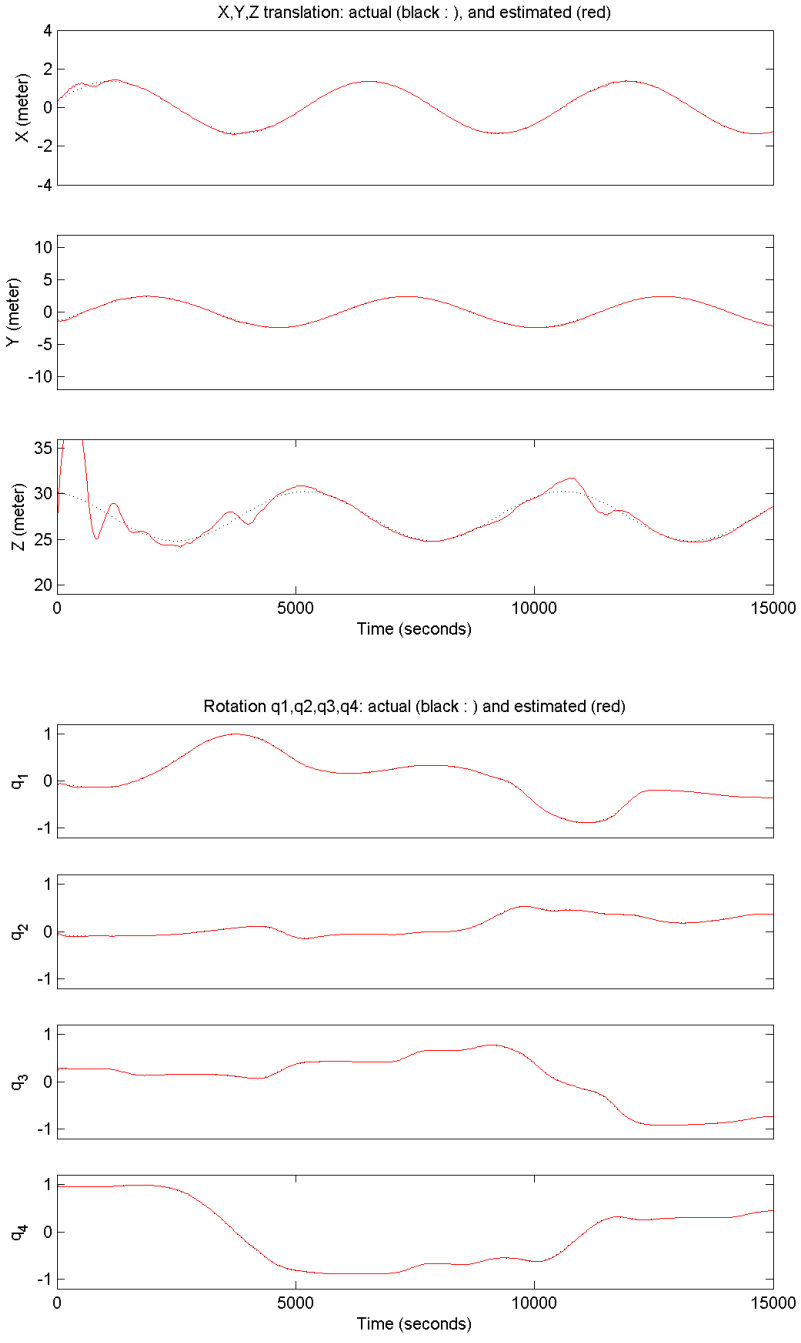



**Figure E.5:** Simulated: Split UKF with concatenated features – translation and rotation

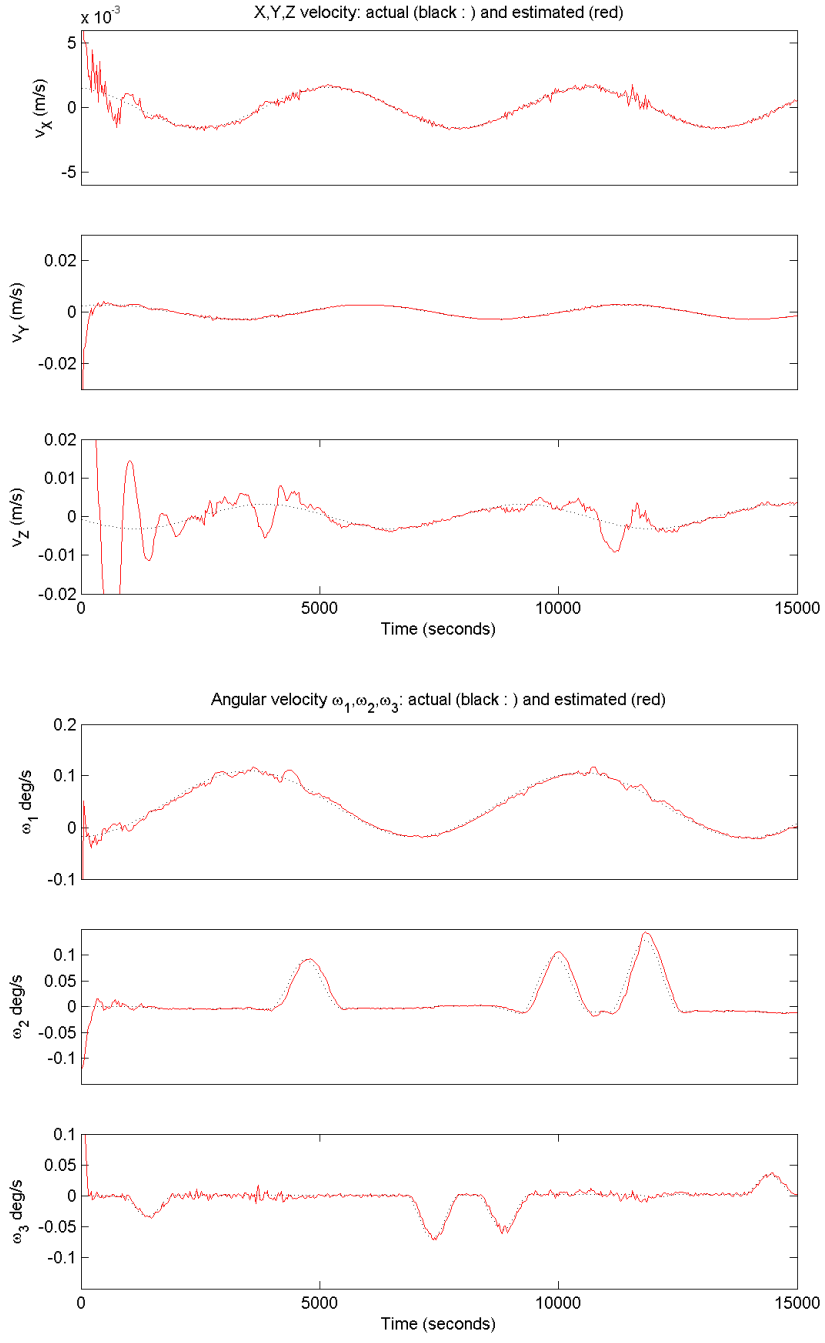



**Figure E.6:** Simulated: Split UKF with concatenated features – linear and angular velocities

## E.2 Real-World Data

Interesting to note is figures E.9 and E.10, where a single EKF is used to treat each 2D feature point as a separate measurement. The depth ($Z$) estimate suffers the most from this approach, as single feature points contain almost no depth information.

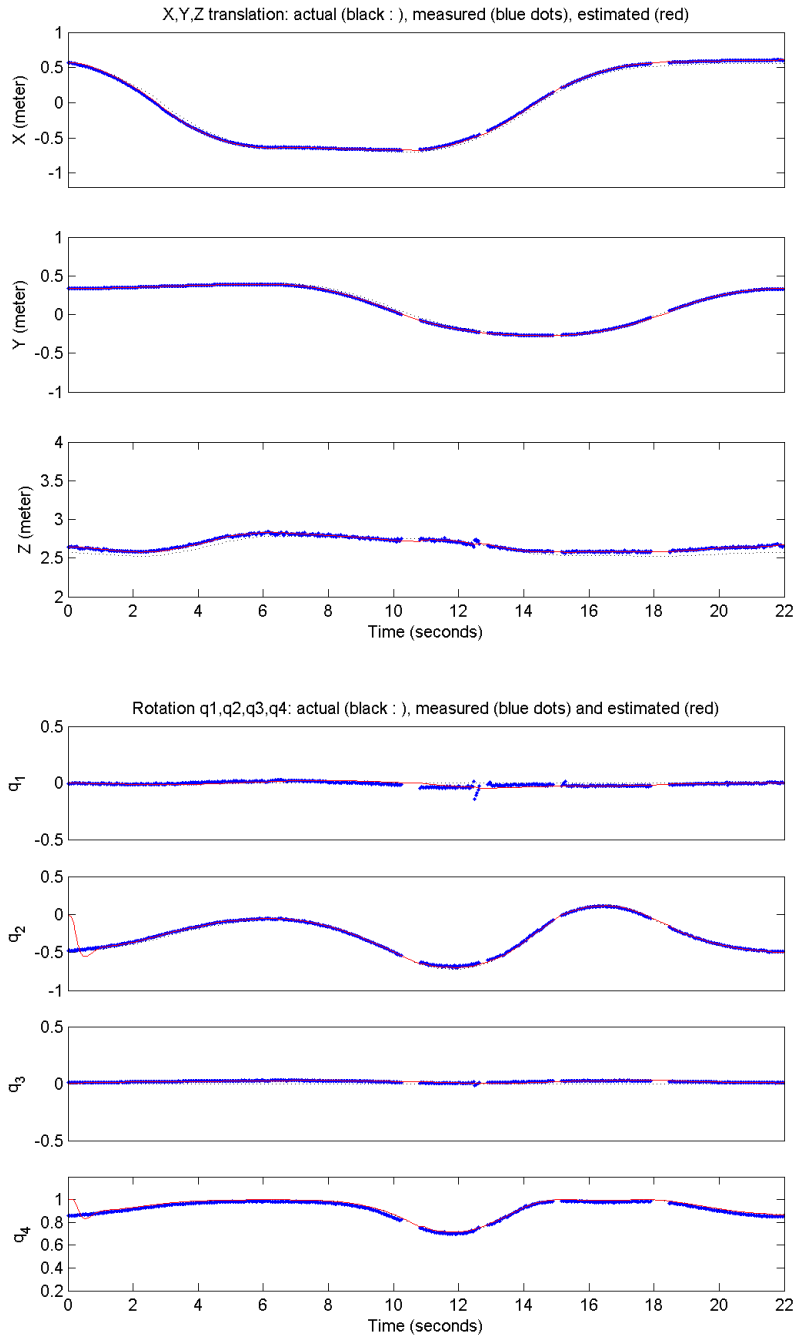



**Figure E.7:** Robot-controlled: Calibration and EKF - translation and rotation

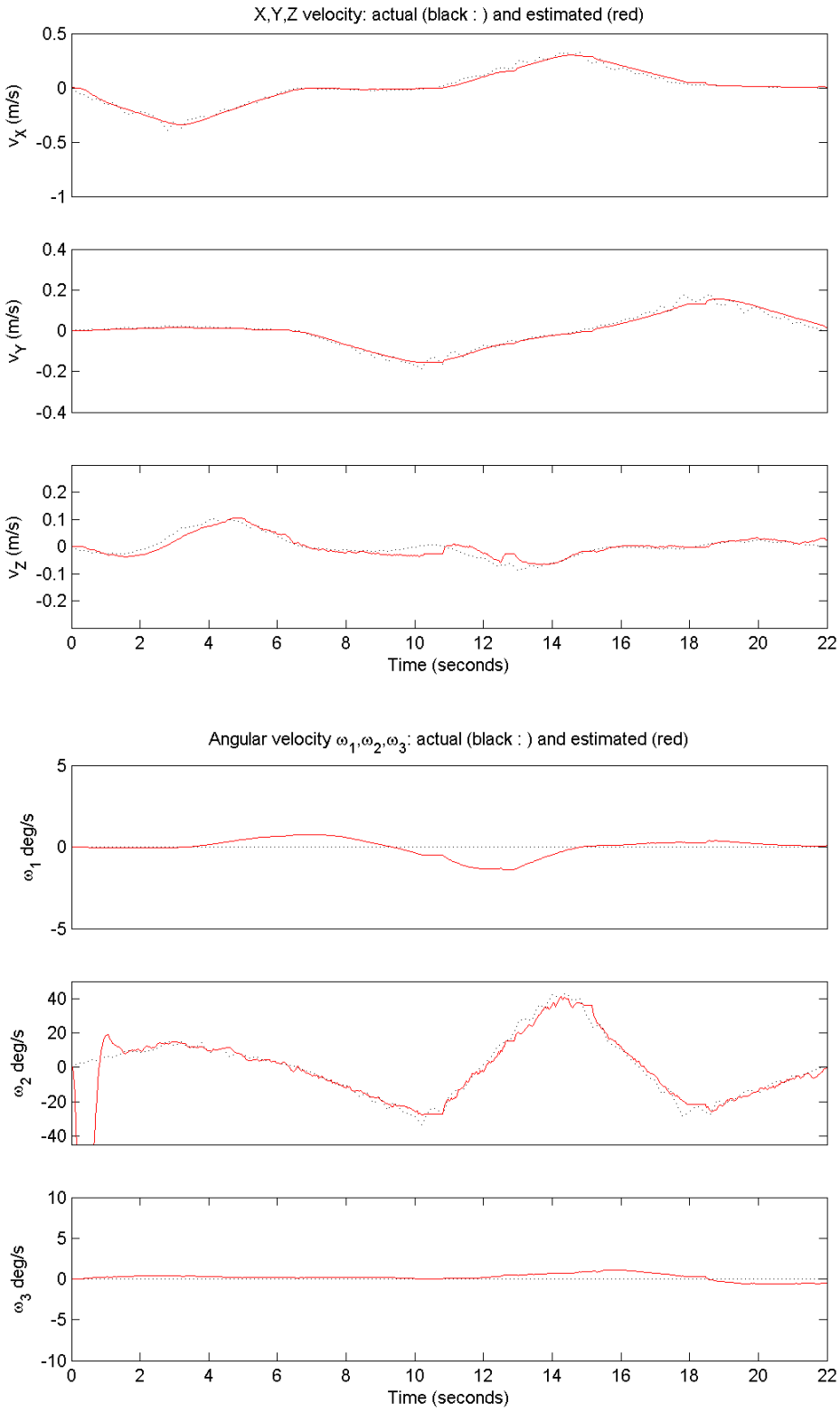



**Figure E.8:** Robot-controlled: Calibration and EKF - linear and angular velocities

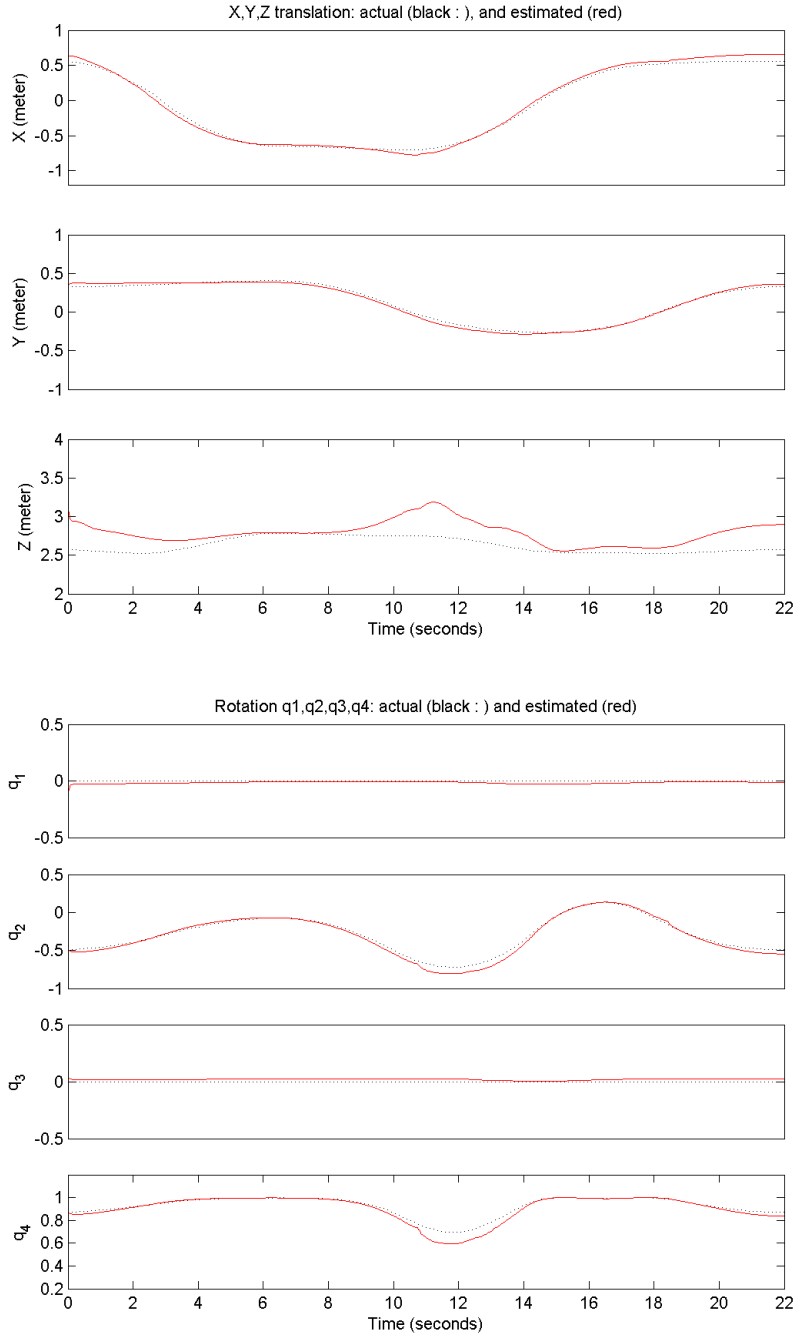



**Figure E.9:** Robot-controlled: Single EKF with separate features – translation and rotation

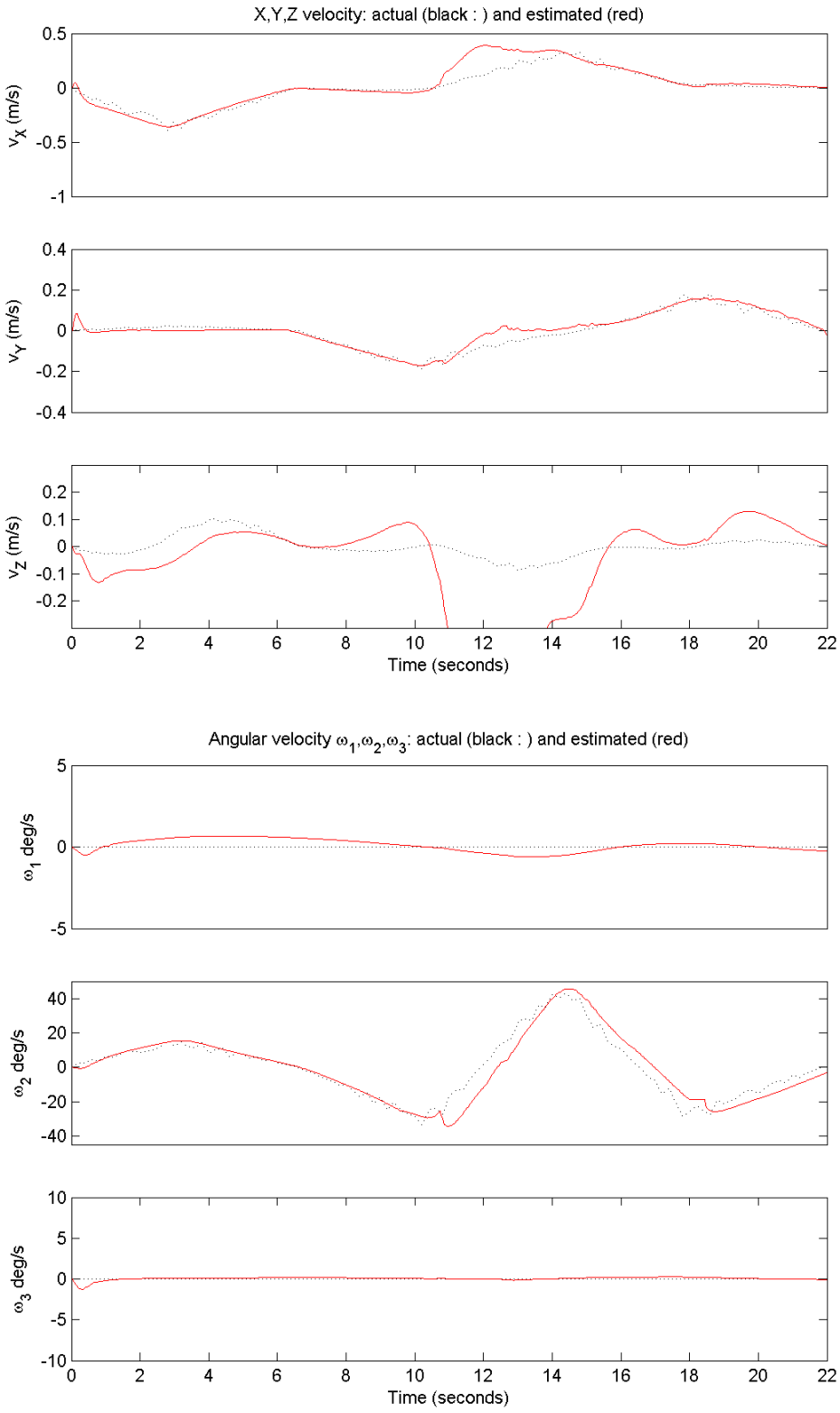



**Figure E.10:** Robot-controlled: Single EKF with separate features – linear and angular velocities

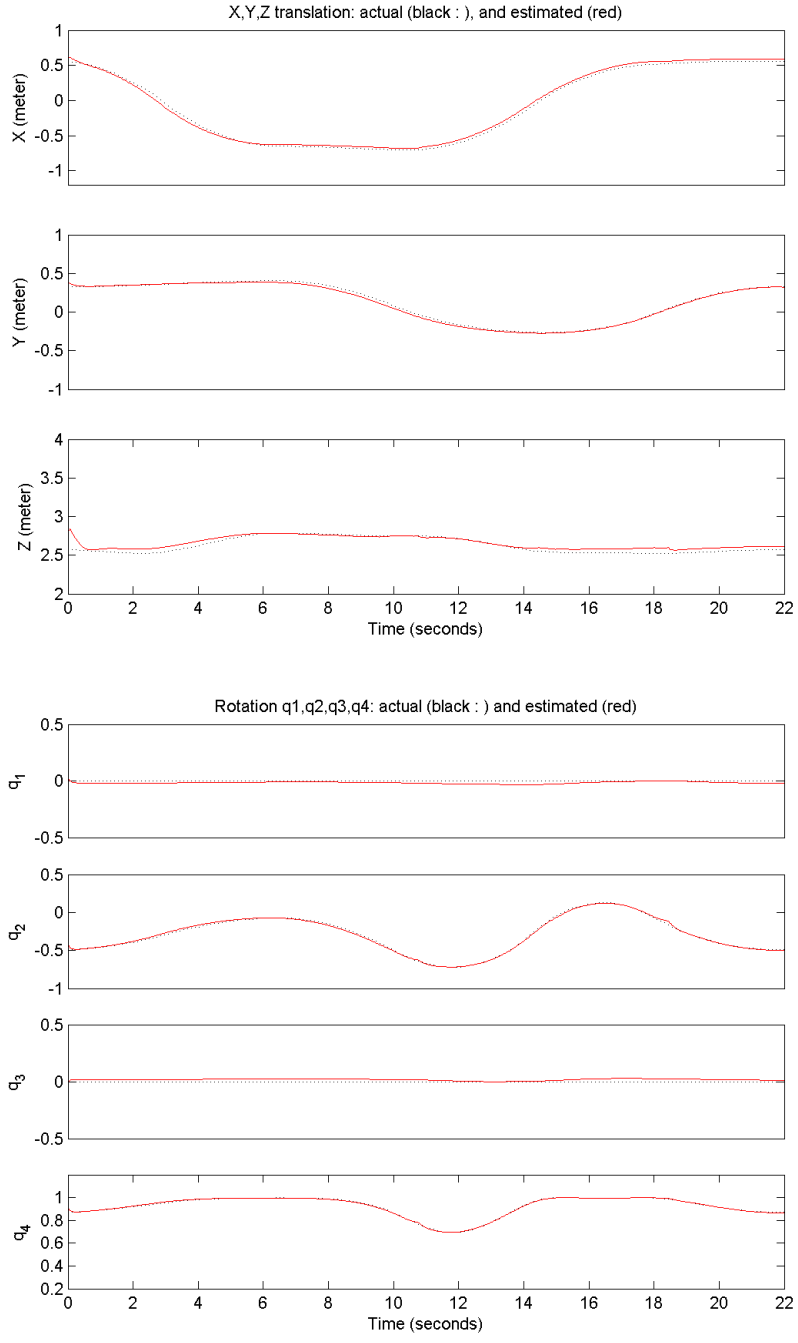



**Figure E.11:** Robot-controlled: Split EKF with concatenated features – translation and rotation

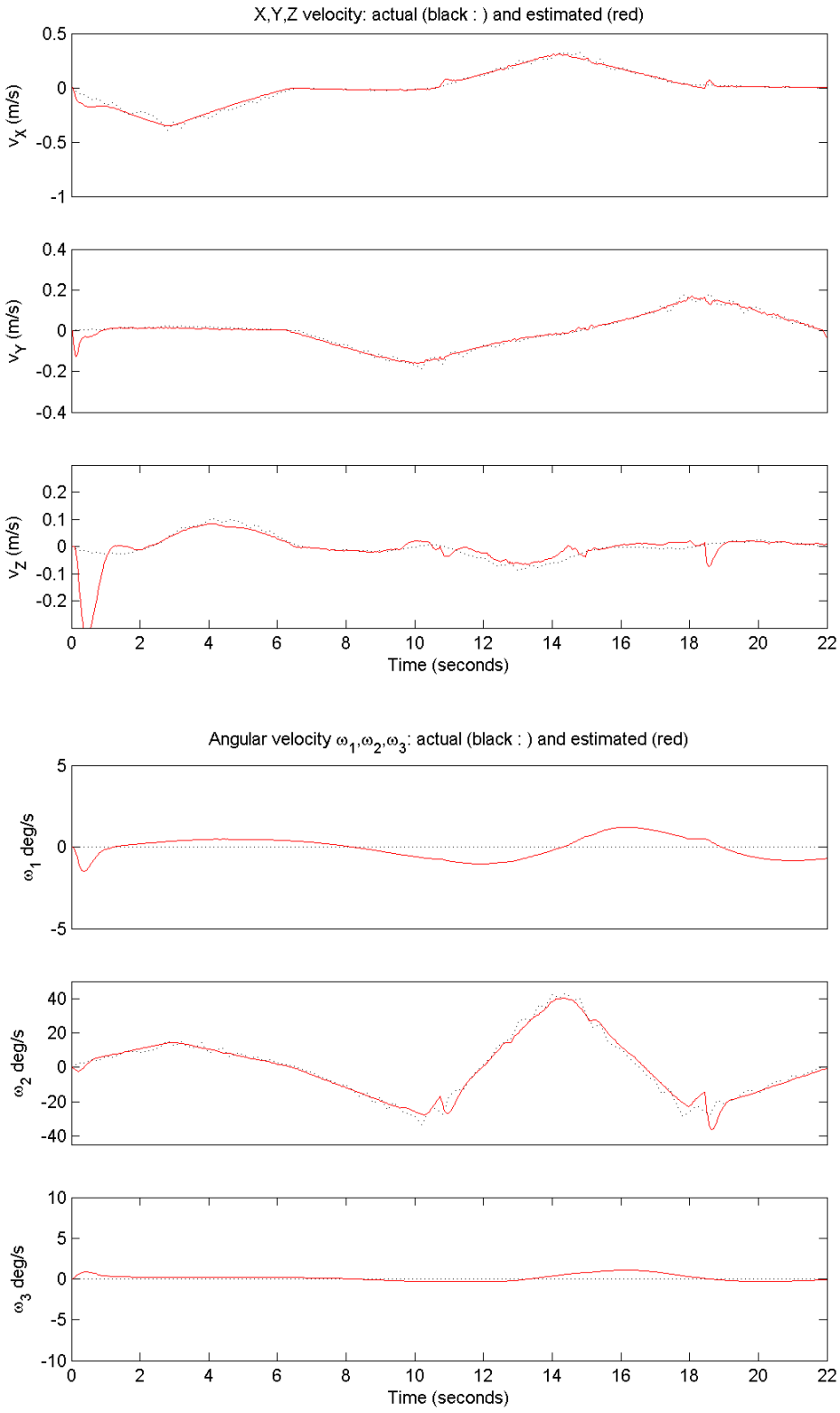



**Figure E.12:** Robot-controlled: Split EKF with concatenated features – linear and angular velocities

## E.3 About the Kalman Filter matrices and numerical values

Setting up each of the covariance matrices for the different KF implementations is not always straightforward. Especially for the nonlinear extensions of the Kalman Filter, there exists no clear optimal method. It appears to be common practice by authors in the literature to choose these values heuristically (and furthermore to omit details in publications).

Although a large covariance matrix allows for many subtle modeling options by means of cross covariances and its large dimensions, suitable values are often difficult to compute analytically. An easy way of simplifying this task is to choose the covariance matrices as diagonal (when possible). This implies that we model the different elements as statistically independent. Although generally not optimal, this simplifies the whole process considerably, and still leads to acceptable results.

Choosing the numerical values for the covariance matrices $\boldsymbol{P}$, $\boldsymbol{Q}$ and $\boldsymbol{R}$ is slightly different for each problem, and was optimised individually according to performance requirements. Accessing the individual scripts on the data CD, as described in appendix F, will supply the reader with specifics.

# Appendix F

# Software and Data

This appendix discusses the directory structure of the accompanying compact disc, which contains the video sequences and code used in this thesis. After the disc's logical layout is explained there will be a short description of how experiments can be reproduced.

## F.1 Directory Structure

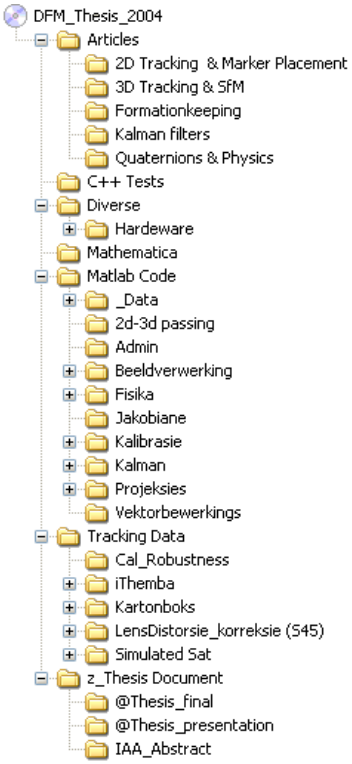



**Figure F.1:** The directory structure on the accompanying CD

### F.1.1 Articles

This directory and its subfolders contain many of the articles referenced in this thesis. This collection is not complete, since some of the articles could only be obtained in hard copy.

### F.1.2 C++ Tests

Some C++ code for interfacing with "webcam" devices, as well as 2D feature tracking algorithms, can be found here. This code was not written by the author – the contained documentation indicate the relevant authors.

### F.1.3 Diverse

The contents of this directory include hardware drivers, data sheets, some LaTeX style sheets, an SfM demo and general information which is only tangentially relevant to this thesis.

### F.1.4 Mathematica

Some mathematical derivations were done with Mathematica. The "notebook" files, with some commentary, can be found in this directory.

### F.1.5 Matlab Code

Most of the software for this thesis was written in the MATLAB programming environment. This directory contains all the code for the different KF variations, as well as the vector manipulation code and physics models. Also included are some image processing routines.

#### F.1.5.1 Tracking Data

All the image and video sequences for each of the practical tests are contained here. Also included are scripts for running various experiments, setup information, as well as the figures of results obtained.

#### F.1.5.2 Thesis document

The electronic version of this thesis is contained in this directory. The LaTeX source code and project files are also included. Separate articles and mini presentations related to this thesis are included in subfolders.

## F.2 Running an experiment

All the scripts for running experiments on the real or simulated image sequences are contained in the `"thesis\tracking data"` directory. It is *important* that the `"Matlab code"` directory and all its subdirectories are included in the path, since this directory contains functions needed by most of the experiments. The following commands, when run from the MATLAB command prompt, will reproduce an experiment – in this case for the robot-controlled target.

Assuming that the user is already in the `"thesis\tracking data"` directory, run the following commands:

```
cd 'Simulated Sat'
load SimSat
exec_nocal_saam_EKF(SimSat,SimSat_groundtruth,'initvar_SimSat')
```

This will compute and display the graph for the single EKF with concatenated measurement as reproduced in this thesis.

To run an experiment for the robot-controlled target, we proceed in a similar manner:

```
cd iThemba
load iThemba
robot_cal_EKF(Boks,seq2D,robot_4_groundtruth,'initvar_robot');
```

The above script will run a calibration algorithm with LKF/EKF experiment, as explained in the thesis. The other tests are run in an analogous fashion. Limited documentation is included in the script files for reference purposes.

## F.3 System specifications

The experiments were run on a Intel Pentium 4 processor, running at 1600 MHz, on a platform equipped with 512 MB of RAM. The operating system used was Microsoft Windows XP, while MATLAB 6.5 (release 13) was mainly used for code development and testing.

## F.4 File formats

Due to the unwieldy size of raw video data, the included video files have been compressed using the DivX 5.1.1 codec. A suitable decompressor is needed for viewing. Some raw MATLAB data files, among others, have been compressed using archiving algorithms such as BZip2 or 7-Zip. A suitable freeware file manager, supporting the relevant decompression schemes, is included in the CD'sroot directory.

# List of References


[1] T.J. Broida, S. Chandrashekhar, and R. Chellappa, "Recursive 3-d motion estimation from a monocular image sequence," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 26, no. 4, pp. 639–656, July 1990. (Cited on pages 2, 18, 32, 51, 54, 65, 66, and 79.)

[2] Steven D. Blostein and Lin Zhao, "Recursive 3-d motion estimation from a monocular image sequence by combining optical flow and position measurements," September 1998. (Cited on page 2.)

[3] James S. Goddard Jr., *Pose and Motion Estimation from Vision using Dual Quaternion-based Extended Kalman Filtering*, Ph.D. thesis, The University of Tennessee at Knoxville, 1997. (Cited on pages 2, 6, 54, and 80.)

[4] Chris Venter, "Structure from motion estimation using a nonlinear kalman filter," M.S. thesis, University of Stellenbosch, July 2002. (Cited on pages 2, 4, 27, 32, 54, 56, and 79.)

[5] Pieter Rautenbach, "Facial feature reconstruction using structure from motion," M.S. thesis, University of Stellenbosch, December 2004. (Cited on pages 2 and 56.)

[6] Richard Hartley and Andrew Zisserman, *Multiple View Geometry*, Cambridge University Press, 2001. (Cited on page 7.)

[7] Rafael C. Gonzalez and Richard E. Woods, *Digital Image Processing*, Prentice Hall, second edition, 2002. (Cited on page 7.)

[8] "Digital photography review," http://www.dpreview.com. (Cited on page 8.)

[9] James R. Wertz, *Spacecraft Attitude Determination and Control*, D Reidel Publishing Company, Boston USA, 1986, Written by Members of the Technical Staff, Computer Sciences Corporation. (Cited on pages 9, 10, 39, 98, 101, and 103.)

[10] Willem H. Steyn, *A Multi-Mode Attitude Determination and Control System for Small Satellites*, Ph.D. thesis, University of Stellenbosch, Dec. 1995. (Cited on pages 10 and 101.)

[11] Andrew Sparks, "Formationkeeping for a pair of satellites in a circular orbit," *Journal of Guidance, Control and Dynamics*, vol. 8, no. 2, pp. 235–242, March 1985. (Cited on pages 11 and 12.)

[12] Gao Yunfeng Baoyinhexi and Li Junfeng, "Study on spacecraft formation flying by the orbital elements method," Tech. Rep., Department of Engineering Mechanics, Tsinghua University, Beijing, China. (Cited on page 11.)

[13] Air Force Research Laboratory, *Satellite Formationkeeping Control in the Presence of Gravity Perturbations, Presented at the American Control Conference*, Wright Patterson AFB, Ohio, July 2000. (Cited on page 12.)

[14] Joaquim Salvi, Xavier Armangué, and Joan Batlle, "A comparative review of camera calibrating methods with accuracy evaluation," *Pattern Recognition*, vol. 35, pp. 1617–1635, 2002. (Cited on pages 13, 14, 16, 17, and 65.)

[15] R.E. Kalman, "A new approach to linear filtering and prediction problems," *Journal of Basic Engineering*, vol. 82, 1960. (Cited on page 23.)

[16] Arthur Gelb et al., *Applied Optimal Estimation*, M.I.T. Press, 1974. (Cited on pages 25, 26, 83, 88, and 91.)

[17] Ben Herbst and Barry Sherlock, "Introduction to the kalman filter and applications," Tech. Rep., Department of Applied Mathematics, University of Stellenbosch, 2003. (Cited on pages 25, 26, 28, and 88.)

[18] David F. Bizup and Donald E. Brown, "The over-extended kalman filter - don't use it!," in *Information Fusion*, System and Information Engineering, Univerity of Virginia, July 2003, vol. 1, pp. 40–46, International Society of Information Fusion. (Cited on pages 26 and 80.)

[19] Yaakov Bar-Shalom, X. Rong Li, and Thiagalingam Kirubarajan, *Estimation with Applications to Tracking and Navigation*, John Wiley & Sons, Inc., 2001. (Cited on pages 26, 83, 87, 88, 89, and 91.)

[20] Greg Welch and Gary Bishop, "An introduction to the kalman filter," Tech. Rep., University of North Carolina at Chapel Hill, 2004. (Cited on page 26.)

[21] Simon Julier and Jeffrey K. Uhlmann, "A general method for approximating nonlinear transformations of probability distributions," Tech. Rep., Robotics Research Group, Department of Engineering Science, University of Oxford, Nov. 1996. (Cited on pages 27, 28, and 56.)

[22] Rudolph van der Merwe, Arnaud Doucet, Nando de Freitas, and Eric Wan, "The unscented particle filter," Tech. Rep., Cambridge University Engineering Department, June 2000. (Cited on pages 27 and 28.)

[23] John L. Crassidis and F. Landis Markley, "Unscented filtering for spacecraft attitude estimation," *Journal of Guidance, Control, and Dynamics*, vol. 26, no. 4, pp. 536–542, July 2003. (Cited on pages 27 and 56.)

[24] Edgar Kraft, "A quaternion-based unscented kalman filter for orientation tracking," *Information Fusion*, vol. 1, pp. 47–54, July 2003. (Cited on pages 27 and 56.)

[25] Joseph J. La Viola, "A comparison of unscented and extended kalman filtering for estimating quaternion motion," *Proceedings 2003 - American Control Conference*, pp. 2435–2440, June 2003. (Cited on pages 27 and 32.)

[26] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, The MIT Press, 1998. (Cited on page 29.)

[27] F. Landis Markley, "Attitude error representations for kalman filtering," *Jounal of Guidance, Control and Dynamics*, March 2003. (Cited on pages 32, 101, and 103.)

[28] J. Schmidt and H. Niemann, "Using quaternions for parametrizing 3-d rotations in unconstrained nonlinear optimization," *Vision, Modeling and Visualization (VMV01)*, November 2001. (Cited on pages 32 and 101.)

[29] Tony Jebara, Ali Azarbayejani, and Alex Pentland, "3d structure from 2d motion," Tech. Rep., MIT Media Laboratory. (Cited on page 32.)

[30] Alonzo Kelly, "A 3d state space formulation of a navigation kalman filter for autonomous vehicles," Tech. Rep., The Robotics Institute, Carnegie Mellon University, May 1994. (Cited on pages 48, 83, 84, 88, and 91.)

[31] Stan Birchfield, "An introduction to projective geometry (for computer vision)," March 1998. (Cited on page 97.)

[32] Herbert Goldstein, *Classical Mechanics*, Addison-Wesley Publishing Company, 2nd edition, 1980. (Cited on page 100.)

[33] Eric W. Weisstein et al., "Mathworld," http://mathworld.wolfram.com/. (Cited on page 100.)

[34] Evangelos A. Coutsias and Louis Romero, "The quaternions with an application to rigid body dynamics," February 1999. (Cited on pages 101 and 103.)