NTNU

Norwegian University of
Science and Technology

Vision Aided Inertial Navigation

Tor Erik N Fredrikstad

Master of Science in Cybernetics and Robotics
Submission date: July 2016
Supervisor: Oddvar Hallingstad, ITK
Co-supervisor: Kjetil Ånonsen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem formulation

(In Norwegian)

Et treghetsnavigasjonssystem (TNS) er basert på å integrere opp akselerasjons- og vinkelakselerasjonsmålinger fra akselerometre og gyroer for å bestemme posisjon og orientering for en plattform. Et slikt system vil alltid ha en avdrift i posisjon og orientering på grunn målefeil i sensorene. Uavhengig av kvaliteten på sensorene, vil det derfor før eller siden være behov for støttemålinger fra eksterne sensorer. En mulighet er å bruke en bildestrøm fra et optisk kamera som en støtte for treghetsnavigasjonssystemet.

Prinsippet bak bildenavigasjon er å følge et sett med gjenkjennelige bildemerker fra bilde til bilde i en videostrøm. Bildemerkene er avbildninger av landemerker i scenen. Et grunnproblem innenfor bildenavigasjon er å bestemme skalaen for scenen, det vil si størrelsen på objektene kameraet avbilder og hvor langt plattformen har beveget seg i scenen. Dersom man integrerer bildenavigasjon med treghetsnavigasjon, vil denne skalaen bli observerbar. Et relatert problem er å bestemme avstanden til de observerte bildepunktene . Målet for oppgaven er å studere disse problemene gjennom simuleringer. Oppgaven tar utgangspunkt i banegenerator og simulator utviklet i foregående fordypningsprosjekt.

Naturlige arbeidspunkter for oppgaven blir:

- Lage et Kalmanfilter for et forenklet treghetsnavigasjonssystem støttet av posisjonsmålinger (GPS) og bildepunkter fra kameramålinger basert på arbeid gjort i fordypningsprosjekt
- Lage en simulator for problemet som gjør det mulig å teste ut ytelsen til systemet i ulike scenarioer, særlig med tanke på avstandsestimering og skala for scenen
- Bruke simulatoren og estimatoren til å bestemme ytelsen i systemet i ulike scenarier og med ulike sensorkvaliteter , f.eks. ulike klasser av treghetssensorer, GPS med og uten fasemålinger osv.
- Dersom tiden tillater det, kan det også bli aktuelt å teste ut resultatene på ekte data på et datasett innsamlet av Forsvarets forskningsinstitutt (FFI).

Start date: 2016-02-25

Due date: 2016-07-28

Thesis performed at: Department of Engineering Cybernetics, NTNU

Supervisor: Professor II Oddvar Hallingstad, Dept. of Eng. Cybernetics, NTNU

Co-supervisor: Kjetil Bergh Ånonsen, Dept. of Eng. Cybernetics, NTNU

Abstract

Camera measurements can be used to replace or supplement GPS measurements for aiding inertial navigation systems. This is especially useful in cases where GPS is unavailable or unreliable, whether it is because the signals are blocked by the environment or intentionally corrupted. In this thesis a system is developed for aiding inertial navigation by a simple vision system consisting of a single camera, and computer simulations are done to test its performance. The system is represented as a state space model which is used in an extended Kalman filter to estimate the navigation states.

The system model is created by relating the acceleration and angular velocity to the rate of change of the orientation, velocity and position of a vehicle and to the relative position of a stationary landmark, which is represented by a pinhole camera projection. Noise values used for simulating inertial sensor noise are calculated based on specifications obtained from a user manual. The state space model is based on considering the inertial measurements as control inputs, directly affecting the states, whle the camera and GPS measurements are outputs of the system. An extended Kalman filter is then implemented, which uses measured inputs and outputs to estimate the state based on a linearized model.

Simulations are done which show that the system works as intended and is able to restrain the drift in position from the inertial navigation. Monte Carlo simulations show that the best results are achieved when the vehicle swings from side to side as it approaches the landmark, however the filter has problems estimating the standard deviations correctly. Suggested further work includes using more landmarks and replacing the extended Kalman filter with an unscented filter.

Preface

This thesis is written as the final work toward a master's degree in Cybernetics and Robotics at NTNU. The work was done at UNIK – University Graduate Center.

I would like to thank my supervisors Oddvar Hallingstad and Kjetil Bergh Ånonsen, as well as Ove Kent Hagen at FFI (Forsvarets forskningsinstitutt – Norwegian Defence Research Establishement) for their help and guidance during the project.

Contents

List of Figures

1 Introduction

An inertial navigation platform, consisting of a gyroscope and an accelerometer, can be used to calculate the position and orientation of a body with respect to an inertial frame of reference, as long as the body is subject to acceleration and angular velocities, and the initial placement is known. However, since the values are obtained by integrating the measurements, the calculated trajectory will diverge quickly from the true value as small errors are accumulated in the integration. In order to rectify this, inertial navigation systems are usually aided by additional sensors.

Direct position updates from satellite navigation systems like GPS are commonly used. These systems give good performance when the satellite signals are available, but there are many situations in which this is not the case. Examples of situations where satellite navigation is not available are indoor, underground or underwater applications, as well as extra-terrestrial navigation. In military situations the satellite signals may also be jammed or spoofed, making alternative means of determining position necessary.

One alternative method for determining position is through computer vision, where the software picks out certain features in an image and tracks the same feature over several images. The movement of the image features is used to determine the placement of the vehicle. In this thesis, an effort is made to study the effects of different factors on the performance of a simplified camera aided inertial navigation system via computer simulations. The navigation system that is developed uses a single camera as the aiding sensor for the INS by tracking one or more stationary landmarks.

The central problem, in this case, is to create a good model of the navigation

system that takes all the measurements into account and to use this model to implement an estimation algorithm that gives as good an estimate of the placement of the body as possible using data from the sensors and information about their uncertainties. In order to use common techniques for estimation, it is necessary to create a state-space model of the system. Important states in the system are the position and orientation, as well as disturbances and those noise terms that can not be modeled as simple white noise.

1.1 Related work

Huster [1] has shown a strategy for determining the relative position of a moving observer relative to a stationary object, using inertial sensors and single camera measurements of a single feature. This is achieved using an unscented Kalman filter. The filter uses information about the force-input that causes the acceleration, in addition to the measurements, and also contains a disturbance model based on the specific application as a navigation system for an underwater vehicle.

In [2] an algorithm is presented where a single camera is used to constrain drift in inertial navigation by estimating both the image points and the vehicle position. The algorithm was able to handle environments of arbitrary scale. In [3] a different kind of parametrization for image features is presented where each feature is related to where it was first observed, and it includes discussion about how to initialize features when the range is unknown. In [4] it is shown analytically that the vehicle speed, feature position, roll and pitch angles, and inertial biases are observable when using camera aided inertial navigation, even when only one feature is used.

1.2 Thesis structure

The thesis is organized into 7 chapters. Chapter 2 contains a descrWiption of the mathematical models of the sensors and their relation to the physical environment. Chapter 3 presents the general estimation concepts as a simple example before introducing the model and the calculations used for estimation. Chapter 4 contains desriptions of the different tests that were done of the simulated system. Chapter 5 contains the results of the tests. Chapter 6 contains some interpretations of the results and discussion based on those, and chapter 7 provides

conclusions along with some suggestions for further work.

Additionally three appendices are included. In appendix A some mathematical background is presented, concerning the description of random signals and processes. Appendix B contains the mathematical descriptions of the trajectories from the trajectory generator that is used for the simulations. The main part of the Matlab code used for the simulations is included in appendix C.

2 System model

This chapter presents mathematical models relating the physical attributes of the vehicle and the environment to the otputs of the sensors used for navigation.

2.1 Inertial sensor model

For the accelerometer, the total error is modelled as the sum of a zero mean white noise term $\mathbf{n}$, and a bias term $\mathbf{b}$ which is initialized based on a Gaussian distribution and held constant over the simulation. For the gyroscope, a white noise term and a bias term is used in addition to a first order Markov process (see appendix A), which will be termed bias instability. The measurements, from the accelerometer and the gyroscope respectively, can then be written as

$$\tilde{\mathbf{f}}_a = \mathbf{f}_a + \mathbf{b}_a - \mathbf{n}_a, \tag{2.1}$$

$$\tilde{\boldsymbol{\omega}} = \boldsymbol{\omega} + \mathbf{b}_g + \boldsymbol{\mu}_g - \mathbf{n}_g, \tag{2.2}$$

where $\mathbf{f}_a$ and $\boldsymbol{\omega}$ are the actual specific force and angular velocity, and $\mathbf{n}_a$ and $\mathbf{n}_g$ are white noise terms. The subscript a is used on the specific force vector $\mathbf{f}_a$ to distinguish it from the state derivative function $\mathbf{f}(\mathbf{x})$ that is introduced later. The Gaussian noise terms have negative signs here for consistency, as this gives positive signs in the state space model. As the noise has zero mean, the signs have no practical relevance.

The bias terms are constant, so their differential equations are

$$\dot{\mathbf{b}}_a = \mathbf{0}, \tag{2.3}$$

$$\dot{\mathbf{b}}_\omega = \mathbf{0}, \tag{2.4}$$

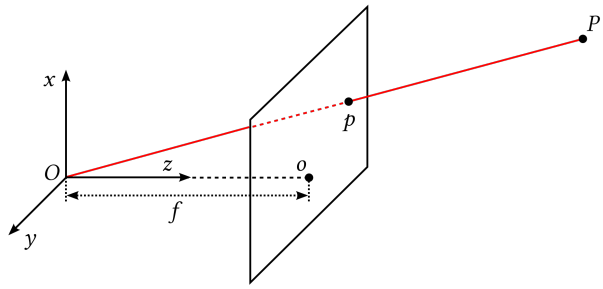

Figure 2.1: Projection of a point in 3D space onto the image plane

and the bias instability of the gyroscope is defined by

$$\dot{\boldsymbol{\mu}}_g = -\frac{1}{T}\boldsymbol{\mu}_g + \mathbf{n}_\mu \tag{2.5}$$

where $\mathbf{n}_\mu$ is a white noise driving term for the Markov process and T is the time constant. It has been shown in [5] that a first order Markov process like this can be a good model for the bias instability.

2.2 Camera model

The camera is modeled using a simple pinhole model, with uncertainties and noise modeled as simple white noise. The camera measurements consist of point coordinates in a two-dimensional image, corresponding to points on landmarks in the world. Figure 2.1 shows the geometric relation between a point P in space, seen from the camera centered reference frame, and its projection p on the image plane in a simple pinhole model.

To simplify the state equations, the image points will be represented by normalized pinhole coordinates. This means that the focal length f, shown on figure 2.1, is set equal to one, and the origin of the camera frame is assumed to be in the same point as the body frame origin. Thus, the normalized pinhole coordinates are related to the position of the landmark relative to the vehicle, seen from the

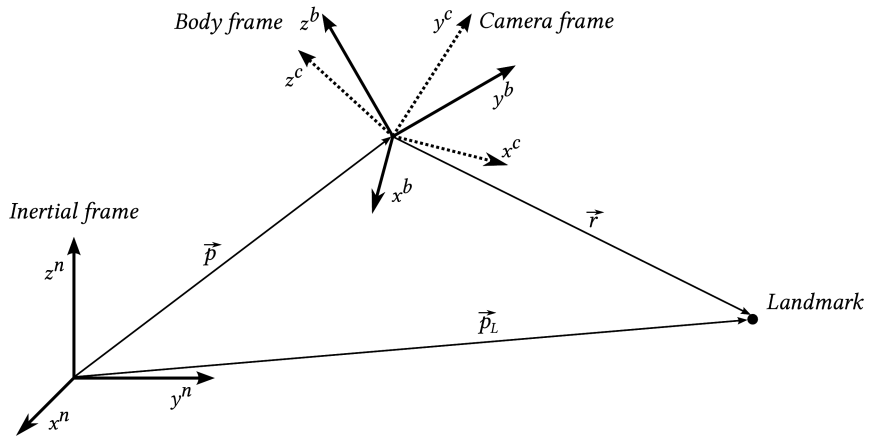

Figure 2.2: Reference frames and position vectors used in the model

camera reference frame, by

$$\begin{bmatrix} s_x \\ s_y \end{bmatrix} = \frac{1}{r_z^c} \begin{bmatrix} r_x^c \\ r_y^c \end{bmatrix}, \tag{2.6}$$

where $\mathbf{r}^c = [r_x^c, r_y^c, r_x^c]^T$ is the position of the landmark relative to the camera as shown on figure 2.2, expressed in camera frame coordinates. The superscripts n, b, and c are used to denote that the coordinates of a vector are given in terms of the inertial frame, the body frame, or the camera frame, respectively.

The normalized coordinates are obtained from the image data by converting the actual pixel coordinates. The pixel coordinates are

$$u_p = f_x s_x + c_x, \quad v_p = f_y s_y + c_y, \tag{2.7}$$

where f_x and f_y are the effective focal lengths for each dimension and (c_x, c_y) is the pixel coordinate of the optical axis (the midpoint of the image). The effective focal lengths are given by the ratio of the actual focal length to the pixel width

w and height h:

$$f_x[\text{px}] = \frac{f[\text{m}]}{h[\text{m/px}]}, \quad f_y[\text{px}] = \frac{f[\text{m}]}{w[\text{m/px}]}. \tag{2.8}$$

The correct units for the effective focal lengths are pixels, as the normalized coordinates are dimensionless.

It may also be noted from figure 2.2 that the camera frame is assumed to be centered at the same point as the body frame. This means that any effects caused by the moment arm between the inertial sensors and the actual camera center are neglected.

2.2.1 Field of view

A point with line of sight to the camera is within its field of view if the pixel coordinates given by (2.7) are within the resolution of the image. That is, a camera measurement of a landmark can be taken if

$$0 < f_x s_x + c_x < \text{res}_x \quad \text{and} \quad 0 < f_y s_y + c_y < \text{res}_y. \tag{2.9}$$

The field of view can also be given as a maximum angle in each direction between the optical axis and the line of sight to the landmark. From figure 2.1 it can be seen that the angles are

$$\theta_x = \tan^{-1} s_x, \quad \theta_y = \tan^{-1} s_y, \tag{2.10}$$

and by rearranging (2.9), we get

$$\theta_{x,\max} = \tan^{-1}\left(\frac{\text{res}_\text{x} - c_x}{f_x}\right), \tag{2.11}$$

$$\theta_{y,\max} = \tan^{-1}\left(\frac{\text{res}_\text{y} - c_x}{f_y}\right). \tag{2.12}$$

2.3 Inertial navigation equations

A simplified inertial navigation model is used, where the earth is viewed as flat and non-rotating. This means that a navigation frame which is stationary with

respect to the ground is also an inertial frame, that is a reference frame that is not under acceleration. The basic kinamatic differential equations for the orientation, position and velocity of the vehicle are [6]

$$\dot{\mathbf{R}}_b^n = \mathbf{R}_b^n \mathbf{S}(\boldsymbol{\omega}^b), \tag{2.13}$$

$$\dot{\mathbf{v}}^n = \mathbf{R}_b^n \mathbf{a}^b, \tag{2.14}$$

$$\dot{\mathbf{p}}^n = \mathbf{v}^n, \tag{2.15}$$

where $\mathbf{R}_b^n$ is the rotation matrix that transforms a vector from the body frame to the inertial frame, $\mathbf{S}(\boldsymbol{\omega}^b)$ is the skew symmetric matrix formed by the angular velocity $\boldsymbol{\omega}^b$, $\mathbf{v}^n$ is the vehicle velocity, and $\mathbf{p}^n$ is its position, both given in inertial frame coordinates. The relative position vector from the vehicle to the landmark can be written as

$$\mathbf{r} = \mathbf{p}_L - \mathbf{p}, \tag{2.16}$$

where $\mathbf{p}_L$ is the position vector from the inertial frame origin to the landmark as shown on figure 2.2. Since the landmark is assumed to be stationary with respect to the inertial frame, this gives

$$\dot{\mathbf{r}}^n = -\mathbf{v}^n. \tag{2.17}$$

It is desirable to express the orientation of the vehicle as a vector and not a matrix. This is commonly done using Euler angles [6], often of the *roll-pitch-yaw* variety, or using quaternions. For *roll-pitch-yaw* Euler angles, stored in the vector

$$\boldsymbol{\lambda} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{roll} \\ \text{pitch} \\ \text{yaw} \end{bmatrix}, \tag{2.18}$$

the kinematic equation is

$$\dot{\boldsymbol{\lambda}} = \mathbf{E}(\boldsymbol{\lambda}) \mathbf{R}_b^n \boldsymbol{\omega}^b, \tag{2.19}$$

where

$$\mathbf{E}(\boldsymbol{\lambda}) = \begin{bmatrix} \cos\psi/\cos\theta & \sin\psi/\cos\theta & 0 \\ -\sin\psi & \cos\psi & 0 \\ \sin\theta\cos\psi/\cos\theta & \sin\theta\sin\psi/\cos\theta & 1 \end{bmatrix}, \tag{2.20}$$

as shown in [1]. The Euler angles $\boldsymbol{\lambda}$ that correspond to the rotation matrix $\mathbf{R}_b^n(\boldsymbol{\lambda})$ are defined as the angles of rotation about each axis that when performed successively gives the rotation described by $\mathbf{R}_b^n$. When using the roll-pitch-yaw definition, the first rotation is the *roll*, about the x-axis , the next is the *pitch* about the y-axis (now in the partially rotated reference frame), and the last angle is the *yaw* about the z-axis [7]. This can be written on matrix form as

$$\mathbf{R}_b^n(\boldsymbol{\lambda}) = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi) \tag{2.21}$$

where $\mathbf{R}_x$, $\mathbf{R}_y$, and $\mathbf{R}_z$ are defined by

$$\mathbf{R}_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.22}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \tag{2.23}$$

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}. \tag{2.24}$$

The resulting matrix is

$$\mathbf{R}_b^n(\boldsymbol{\lambda}) = \begin{bmatrix} \mathrm{c}\psi\mathrm{c}\theta & -\mathrm{s}\psi\mathrm{c}\phi + \mathrm{c}\psi\mathrm{s}\theta\mathrm{s}\phi & \mathrm{s}\psi\mathrm{s}\phi + \mathrm{c}\psi\mathrm{s}\theta\mathrm{c}\phi \\ \mathrm{s}\psi\mathrm{c}\theta & \mathrm{c}\psi\mathrm{c}\phi + \mathrm{s}\psi\mathrm{s}\theta\mathrm{s}\phi & -\mathrm{c}\psi\mathrm{s}\phi + \mathrm{s}\psi\mathrm{s}\theta\mathrm{c}\phi \\ -\mathrm{s}\theta & \mathrm{c}\theta\mathrm{s}\phi & \mathrm{c}\theta\mathrm{c}\phi \end{bmatrix}, \tag{2.25}$$

where c and s stand for cosine and sine respectively.

2.4 GPS

Inertial navigation systems are commonly aided by use of GNSS (global navigation satellite system). The most well known GNSS is the American GPS (Global Positioning System). A GPS receiver uses radio signals from several satellites (at

least four) to calculate the distance to each satellite, which is then used to pinpoint the receiver location. The satellite signal contains data that can be used to calculate satellite position and velocity, clock error, and satellite health. The distance to a satellite is calculated using the signal transmission time, however, this calculated value will be biased by errors particularly due to clock inaccuracy. Therefore, the calculated distance is called the *pseudorange*, to distinguish it from the actual range. An estimate for the true range is found by subtracting an estimate of the clock error from the pseudorange. [8]

To assess the performance of camera-based navigation working together with GPS measurements, this thesis will consider a loosely coupled approach, meaning that the GPS error states are not estimated in the filter, but are assumed to have been corrected beforehand. The simulated GPS measurements are then represented by simple position updates with Gaussian noise added to represent inaccuracy.

2.5 Noise values

The noise on the gyroscope and accelerometer measurements is described as a sum of white noise and a constant bias term for both sensors, and an additional bias fluctuation on the gyroscope, modeled as a 1st order Markov process. The power spectral densities of the white noise processes (see appendix A) are obtained from nominal values in the Xsens MTi user manual [9].

The most relevant specifications are the noise density, which is related to the white noise, the bias repeatability, which is interpreted as the standard deviation of the bias, and the in-run bias stability for the gyroscope, which defines the white noise driving term of the bias fluctuations.

The units for the noise density is $[^\circ/\mathrm{s}/\sqrt{\mathrm{Hz}}]$ for the gyroscope and $[\mu g/\sqrt{\mathrm{Hz}}]$ for the accelerometer. The bias stability of the gyroscope is given in $[^\circ/\mathrm{h}]$.

The proper units for the noise terms are the same as the units of the measured values, $[\mathrm{rad\,s^{-1}}]$ and $[\mathrm{m/s^2}]$, so the unit for the gyroscope bias fluctuation term should be $[\mathrm{rad/s^2}]$.

Power spectral density represents "power" per unit frequency interval, so the units are $[(\mathrm{rad/s})^2/\mathrm{Hz}]$, $[(\mathrm{m/s^2})^2/\mathrm{Hz}]$, $[(\mathrm{rad/s^2})^2/\mathrm{Hz}]$, $[(\mathrm{m/s^3})^2/\mathrm{Hz}]$. The PSD of the white noise terms have the same dimensions as the square of the noise

densities. Converting the units gives

$$S_g = \left(\frac{2\pi}{360^\circ} D_g\right)^2, \tag{2.26}$$

$$S_a = \left(\frac{1}{g}\mathrm{m/s^2} \cdot 10^{-6} D_a\right)^2, \tag{2.27}$$

where S_g and S_a are the PSD of the gyroscope white noise and the accelerometer white noise respectively, and D_a and D_g are the noise densities as given in the user manual.

The in-run bias stability is interpreted as the steady-state value of the standard deviation of the Markov process, and is used to calculate the PSD of the driving noise $\mathbf{n}_\mu$. The covariance of a random process described by the differential equation

$$\dot{\mathbf{x}}(t) = \mathbf{F}\mathbf{x}(t) + \mathbf{G}\mathbf{n}(t), \tag{2.28}$$

where $\mathbf{n}(t)$ is a white noise term, is described by the differential equation [10]

$$\dot{\mathbf{P}}_x(t) = \mathbf{F}\mathbf{P}_x(t) + \mathbf{P}_x(t)\mathbf{F}^T + \mathbf{G}\tilde{\mathbf{Q}}\mathbf{G}^T, \tag{2.29}$$

where $\tilde{\mathbf{Q}}$ is the PSD of $\mathbf{n}(t)$. For the bias stability, given by (2.5), this becomes

$$\dot{\mathbf{P}}_\mu(t) = -\frac{2}{T}\mathbf{P}_\mu(t) + \tilde{\mathbf{Q}}_{n\mu}, \tag{2.30}$$

where T is the time constant. The steady-state value $\mathbf{P}_{\mu,s}$ is found by setting (2.30) equal to zero, which gives

$$\tilde{\mathbf{Q}}_{n\mu} = \frac{2}{T}\mathbf{P}_{\mu,s}. \tag{2.31}$$

Denoting the in-run bias stability value given in the user manual as σ_μ, we have

$$\mathbf{P}_{\mu,s} = \left(\frac{2\pi}{360^\circ} \cdot \frac{1}{3600}\right)^2 \begin{bmatrix} \sigma_\mu^2 & & \\ & \sigma_\mu^2 & \\ & & \sigma_\mu^2 \end{bmatrix}. \tag{2.32}$$

The time constant is chosen as $T = 100$ [s].

The noise is simulated as a discrete-time sequence. The relation between the spectral densities of the continuous-time signal and the covariances of the noise samples is found by looking at the effect of integrating the noise. One sample of simulated noise is represented by

$$\boldsymbol{\nu}_{sim,k} = \int_{t_k}^{t_{k+1}} \mathbf{n}(\tau)\, \mathrm{d}\tau, \tag{2.33}$$

where $\mathbf{n}(t)$ is the continuous time white noise vector. The covariance of the discrete-time simulated noise samples is

$$\mathbf{Q}_{sim} = \mathrm{E}[\boldsymbol{\nu}_{sim,k}\boldsymbol{\nu}_{sim,k}^T] = \int_{t_k}^{t_{k+1}} \tilde{\mathbf{Q}}\, \mathrm{d}\tau = (\Delta t)\tilde{\mathbf{Q}}, \tag{2.34}$$

where the power spectral density matrix is

$$\tilde{\mathbf{Q}} = \begin{bmatrix} \tilde{\mathbf{Q}}_a & & \\ & \tilde{\mathbf{Q}}_g & \\ & & \tilde{\mathbf{Q}}_{n\mu} \end{bmatrix} \tag{2.35}$$

and the diagonal terms are the PSD matrices for the white noise of each sensor and for the bias stability driving noise for the gyroscope. Assuming uncorrelated noise with the same PSD for each axis, we have

$$\tilde{\mathbf{Q}}_a = \begin{bmatrix} S_a & & \\ & S_a & \\ & & S_a \end{bmatrix} \tag{2.36}$$

for the accelerometer white noise PSD, and similar for the other PSD-matrices.

3 State estimation

An estimator is developed for the system presented in the previous chapter. First a simple example is presented in section 3.1 to illustrate the general concepts used, and then the complete estimator model is presented in section 3.2.

3.1 Simple scalar example

Consider a point moving along a straight line. The goal is to accurately estimate its position using an acceleration measurement as well as a position measurement. Both measurements are modeled as the true value added with zero mean Gaussian white noise. The position is related to the acceleration by the kinematic equations

$$\dot{p} = v, \tag{3.1}$$

$$\dot{v} = a. \tag{3.2}$$

The acceleration measurement is modeled as

$$\tilde{a} = a - n_a, \tag{3.3}$$

so that the derivative of the velocity can be given in terms of the measurement as

$$\dot{v} = \tilde{a} + n_a. \tag{3.4}$$

The position measurement is modeled as

$$\tilde{p}_k = p(k\Delta t) + n_{p,k}, \tag{3.5}$$

where δt is the time interval between position measurements.

In this way, we get a state space model that can be written as

$$\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{L}u + \mathbf{G}n_a, \tag{3.6}$$
$$z_k = \mathbf{H}\mathbf{x}_k + n_{p,k} \tag{3.7}$$

where the state vector is $\mathbf{x} = [p \quad v]^T$, $u = \tilde{a}$ is the acceleration measurement, n_a is the process noise, $z_k = \tilde{p}_k$ is the position measurement and n_p is the measurement noise. The matrices are

$$\mathbf{F} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \ \mathbf{L} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \ \mathbf{G} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \ \mathbf{H} = \begin{bmatrix} 1 & 0 \end{bmatrix}, \tag{3.8}$$

and the noise terms are defined by their mean and autocorrelation (for continous-time noise) or covariance (for discrete-time noise):

$$\mathrm{E}[n_a] = 0, \quad \mathrm{E}[n_a(t)n_a(t+\tau)] = q\delta(\tau) \ [\mathrm{m}^2/\mathrm{s}^3], \tag{3.9}$$
$$\mathrm{E}[n_p] = 0, \quad \mathrm{E}[n_{p,k}n_{p,l}] = r\delta_{kl} \ [\mathrm{m}^2], \tag{3.10}$$

where $\delta(\tau)$ is the Dirac delta function and δ_{kl} is the Kronecker delta.

This model is continuous, but we want to use a discrete-time filter which can be easily implemented, so we have to create an equivalent discrete-time model. The matrices for the discrete-time model can be calculated from the exact solution to the system of differential equations, which is

$$\mathbf{x}(t) = e^{\mathbf{F}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{F}(t-\tau)} \left(\mathbf{L}u(\tau) + \mathbf{G}n_a(\tau)\right) \mathrm{d}\tau. \tag{3.11}$$

However, the calculations can be very complicated, and most nonlinear systems do not have a suitable closed form solution at all. Therefore, we will instead consider methods for numerical integration, the simplest of which is Euler's method. Denoting the discrete-time model as

$$\mathbf{x}_{k+1} = \mathbf{\Phi}\mathbf{x}_k + \mathbf{\Lambda}u_k + \mathbf{n}_{a,k}, \tag{3.12}$$
$$z_k = \mathbf{H}\mathbf{x}_k + n_{p,k}, \tag{3.13}$$

the matrices derived from applying Euler's method are

$$\mathbf{\Phi} = \mathbf{I} + \mathbf{F}\Delta t, \quad \mathbf{\Lambda} = \mathbf{L}\Delta t. \tag{3.14}$$

The discrete-time process noise $\mathbf{n}_{a,k}$ is described by its covariance matrix $\mathbf{Q_d}$. This is derived from equation (3.11), which when compared to (3.12) yields

$$\mathbf{n}_{a,k} = \int_{t_k}^{t_{k+1}} e^{\mathbf{F}(t_{k+1}-\tau)}\mathbf{G}n_a(\tau)\,\mathrm{d}\tau. \tag{3.15}$$

Taking the covariance of this expression we get

$$\mathbf{Q_d} = q\mathbf{\Phi}\mathbf{G}\mathbf{G}^T\mathbf{\Phi}^T\int_{t_k}^{t_{k+1}}\mathrm{d}\tau = q\mathbf{\Phi}\mathbf{G}\mathbf{G}^T\mathbf{\Phi}^T\Delta t. \tag{3.16}$$

The optimal estimate of position and velocity is then calculated using the Kalman filter algorithm [10]:

$$\bar{\mathbf{x}}_k = \mathbf{\Phi}\hat{\mathbf{x}}_{k-1} + \mathbf{\Lambda}u_{k-1} \tag{3.17}$$

$$\bar{\mathbf{P}}_k = \mathbf{\Phi}\hat{\mathbf{P}}_{k-1}\mathbf{\Phi}^T + \mathbf{Q_d} \tag{3.18}$$

$$\mathbf{K}_k = \bar{\mathbf{P}}_k\mathbf{H}^T\left(r + \mathbf{H}\bar{\mathbf{P}}_k\mathbf{H}^T\right)^{-1} \tag{3.19}$$

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + \mathbf{K}_k\left(z_k - \mathbf{H}\bar{\mathbf{x}}_k\right) \tag{3.20}$$

$$\hat{\mathbf{P}}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\bar{\mathbf{P}}_k \tag{3.21}$$

The first two lines form the prediction step, or time update, which is based on the acceleration measurement and the previous estimate. The next step is the computation of the Kalman gain, and the last two lines form the estimation step, or measurement update, which incorporates the aiding position measurement z_k. The initial estimates $\hat{\mathbf{x}}_0$ and $\hat{\mathbf{P}}_0$ are chosen based on prior knowledge or assumptions about the initial state:

$$\hat{\mathbf{x}}_0 = \mathrm{E}[\mathbf{x}_0], \quad \hat{\mathbf{P}}_0 = \mathrm{Cov}(\mathbf{x}_0) \tag{3.22}$$

In practice, the inertial measurements are often available at a much higher rate than the aiding measurements. This can be accounted for by computing the

time update at the rate of the inertial measurements, and only computing the Kalman gain and measurement update when the aiding measurement is available. If the time between inertial measurements is Δt_1 and the time between aiding measurements is $\Delta t_2 = N\Delta t_1$, the estimate can be computed as shown in algorithm 1, only performing the measurement update a times $t_k = k\Delta t_1$ where k is an integer multiple of N. It is assumed here that the first aiding measurement is available at $k = N$.

Algorithm 1 Kalman filter with update rate N times the measurement rate

{Time update}
$\bar{\mathbf{x}}_k = \mathbf{\Phi}\hat{\mathbf{x}}_{k-1} + \mathbf{\Lambda}u_{k-1}$
$\bar{\mathbf{P}}_k = \mathbf{\Phi}\hat{\mathbf{P}}_{k-1}\mathbf{\Phi}^T + \mathbf{Q_d}$
if $k \mod N == 0$ **then**
  {Measurement update}
  $\mathbf{K}_k = \bar{\mathbf{P}}_k\mathbf{H}^T\left(r + \mathbf{H}\bar{\mathbf{P}}_k\mathbf{H}^T\right)^{-1}$
  $\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + \mathbf{K}_k\left(z_k - \mathbf{H}\bar{\mathbf{x}}_k\right)$
  $\hat{\mathbf{P}}_k = \left(\mathbf{I} - \mathbf{K}_k\mathbf{H}_k\right)\bar{\mathbf{P}}_k$
else
  $\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k$
  $\hat{\mathbf{P}}_k = \bar{\mathbf{P}}_k$
end if

3.2 Estimator model

The state space model that is used for the estimator consists of a simplified INS model as shown in section 2.3, using Euler angles to represent orientation, and seeing the earth as flat and non-rotating. The non-Gaussian noise terms in the sensor model are also modeled as separate states. This INS model is then augmented with three scalar states for the relative position of each landmark. The first two of these are normalized image coordinates, and the third is the inverse range. This choice of states is taken from [11], where it is shown that it results in comparatively good estimation properties. One particular advantage of representing the landmark position using the camera coordinates as states is that the

measurement model becomes linear.

The total state vector is

$$\mathbf{x}^T = [\mathbf{p}^n, \mathbf{v}^n, \boldsymbol{\lambda}, \mathbf{b}_a, \mathbf{b}_g, \boldsymbol{\mu}_g, s_x^1, s_y^1, \zeta^1, ..., s_x^N, s_y^N, \zeta^N] \tag{3.23}$$

where $\mathbf{p}^n$ is the vehicle position in global frame coordinates, $\mathbf{v}^n$ is the velocity in global frame coordinates, $\boldsymbol{\lambda}$ is the orientation given as roll-pitch-yaw Euler angles, $\mathbf{b}_a$ and $\mathbf{b}_g$ are constant bias terms on the accelerometer- and gyroscope measurements, $\boldsymbol{\mu}_g$ is the slowly varying part of the gyroscope noise, s_x^i and s_y^i are normalized image coordinates for landmark number i, ζ^i is the inverse feature range, and N is the total number of landmarks that are tracked by the vision system. For the rest of this thesis only one one single landmark will be used, but it should be possible to include an arbitrary number simply by expanding the state and measurement vectors as shown. The state dynamics are modeled as

$$\dot{\mathbf{p}}^n = \mathbf{v}^n, \tag{3.24}$$
$$\dot{\mathbf{v}}^n = \mathbf{R}_b^n(\boldsymbol{\lambda})\mathbf{f}_a^b - \mathbf{g}, \tag{3.25}$$
$$\dot{\boldsymbol{\lambda}} = \mathbf{E}(\boldsymbol{\lambda})\mathbf{R}_b^n\boldsymbol{\omega}^b, \tag{3.26}$$
$$\dot{\mathbf{b}}_a = \mathbf{0}, \tag{3.27}$$
$$\dot{\mathbf{b}}_g = \mathbf{0}, \tag{3.28}$$
$$\dot{\boldsymbol{\mu}}_g = -\frac{1}{T}\boldsymbol{\mu}_g + \mathbf{n}_\mu, \tag{3.29}$$
$$\dot{s}_x = -v_x^c\zeta + s_x v_z^c\zeta + s_x s_y\omega_x^c - (1+s_x^2)\omega_y^c + s_y\omega_z^c, \tag{3.30}$$
$$\dot{s}_y = -v_y^c\zeta + s_y v_z^c\zeta + (1+s_x^2)\omega_x^c - s_x s_y\omega_y^c - s_x\omega_z^c, \tag{3.31}$$
$$\dot{\zeta} = v_z^c\zeta^2 + \zeta s_y\omega_x^c - \zeta s_x\omega_y^c, \tag{3.32}$$

where $\mathbf{f}_a$ is the specific force acting on the vehicle, and $\boldsymbol{\omega}$ is the angular velocity vector. The equations for s_x, s_y, and ζ are taken directly from [11]. The specific force and angular velocity are related to the inertial measurements by

$$\mathbf{f}_a = \tilde{\mathbf{f}}_a - \mathbf{b}_a + \mathbf{n}_a, \tag{3.33}$$
$$\boldsymbol{\omega} = \tilde{\boldsymbol{\omega}} - \mathbf{b}_g - \boldsymbol{\mu}_g + \mathbf{n}_g, \tag{3.34}$$

where $\tilde{(\cdot)}$ denotes the measured value. $\mathbf{n}_\mu$, $\mathbf{n}_a$, and $\mathbf{n}_\omega$ are white noise terms. The matrix $\mathbf{E}(\boldsymbol{\lambda})$ in the derivative of the Euler angles is given in (2.20). The camera frame velocity $\mathbf{v}^c$ is computed from the global frame velocity using

$$\mathbf{v}^c = \mathbf{R}_b^c \mathbf{R}_n^b \mathbf{v}^n \tag{3.35}$$

where the orientation of the camera relative to the body, represented by $\mathbf{R}_b^c$ is assumed to be known, and $\mathbf{R}_n^b$ is calculated from the euler angles $\boldsymbol{\lambda}$. Similarly for the camera frame angular velocity,

$$\boldsymbol{\omega}^c = \mathbf{R}_b^c \boldsymbol{\omega}^b. \tag{3.36}$$

For the rest of this thesis the camera frame is assumed to concide with the body frame, that is $\mathbf{R}_b^c = \mathbf{I}$, so $\mathbf{v}^c = \mathbf{v}^b$, $\boldsymbol{\omega}^c = \boldsymbol{\omega}^b$

The states are predicted from the inertial measurement during the time-update step of a Kalman filter, and updated using camera- and position (GPS) measurements in the measurement-update step. This means that the inertial measurements are used as inputs, giving the input vector

$$\mathbf{u}^T = [\tilde{\mathbf{f}}_a, \tilde{\boldsymbol{\omega}}^b], \tag{3.37}$$

while the camera- and position measurements are modeled as outputs. The measured image coordinates are given in pixels, and are related to the normalized image coordinates in the state vector by (2.7). In addition to position and camera measurements, velocity measurements can also be used. This gives the output vector

$$\mathbf{z} = \mathbf{h}(\mathbf{x}) + \mathbf{n}_{meas} = \begin{bmatrix} f_x s_x^1 + c_x \\ f_y s_y^1 + c_y \\ \vdots \\ f_x s_x^N + c_x \\ f_y s_y^N + c_y \\ \mathbf{p}^n \\ \mathbf{v}^n \end{bmatrix} + \begin{bmatrix} n_{sx} \\ n_{sy} \\ \vdots \\ n_{sx} \\ n_{sx} \\ \mathbf{n}_p \\ \mathbf{v}_v \end{bmatrix} \tag{3.38}$$

The size of the output vector changes depending on what measurements are available at the time, and unavailable measurements are simply removed from the vector.

The complete model can be written as

$$\dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{n}_{sys}(t)\right), \tag{3.39}$$

$$\mathbf{z}_k = \mathbf{h}\left(\mathbf{x}(t_k)\right) + \mathbf{n}_{meas,k}, \tag{3.40}$$

where the state transition function $\mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{n}_{sys}(t)\right)$ is given by equations (3.24 – 3.35).

An extended Kalman filter is implemented by discretizing the state dynamics using Euler's method:

$$\mathbf{x}_{k+1} = \mathbf{f}_d(\mathbf{x_k}) = \mathbf{x}_k + \Delta t \mathbf{f}(\mathbf{x}_k), \tag{3.41}$$

and calculating the jacobian at each iteration using complex step differentiation. For a scalar function of a single scalar variable, the complex step differentiation is

$$\frac{\mathrm{d}f}{\mathrm{d}x_0} \approx \frac{\mathfrak{Im}\left(f(x_o + ih)\right)}{h}, \tag{3.42}$$

where h is a small, real valued parameter, and $\mathfrak{Im}(z)$ denotes the imaginary part of a complex number z. In the Matlab implementation by [12], h is chosen to be the floating point relative accuracy times the number of states. For an n-dimensional state vector, the algorithm for complex step differentiation is

for $j = 1$ to n **do**
  $\mathbf{x}^* = \begin{bmatrix} x_1 & \dots & x_j + ih & \dots & x_n \end{bmatrix}^T$
  $\frac{\partial \mathbf{f}}{\partial \mathbf{x}_j} = \mathfrak{Im}\left(\mathbf{f}(\mathbf{x}^*)\right)/h$
end for
$\mathbf{F} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial \mathbf{x}_1} & \dots & \frac{\partial \mathbf{f}}{\partial \mathbf{x}_n} \end{bmatrix}$

To make the different types of aiding measurements available at the determined rate and over the determined time intervals, the arrays of simulated measurements are filled with the value "NaN" (not a number) for all the time instances where the measurement is not supposed to be available. The program then does a check for "NaN" values for all the measurement arrays before each iteration of the extended Kalman filter and defines the measurement function $\mathbf{h}(\mathbf{x})$ and the measurement covariance matrix $\mathbf{R}$ according to the correct number and types of measurements avalable at that iteration.

3.3 Error state model

In order to implement the extended Kalman filter it is also necessary to find the proper values for the state covariance matrix $\mathbf{Q}_k$ that is passed to the filter at each iteration. One way to derive this matrix is to look at the error state model. The error state vector $\delta\mathbf{x}$ is defined as the difference between the true state vector $\mathbf{x}$, and a nominal or mechanized state vector $\check{\mathbf{x}}$ which is computed from the INS-measurements $\tilde{\mathbf{f}}^b$ and $\tilde{\boldsymbol{\omega}}^b$, that is

$$\delta\mathbf{x} = \mathbf{x} - \check{\mathbf{x}} \tag{3.43}$$

The true state dynamics are defined by the true specific force $\mathbf{f}^b$ and angular velocity $\boldsymbol{\omega}^b$.

The error equations are derived below. Note that the orientation is represented by rotation matrices and not Euler angles when deriving the error state dynamics. The rotation error itself, however, is represented using error angles $\boldsymbol{\epsilon}$, such that

$$\mathbf{R}_b^n = \mathbf{R}_b^n(\epsilon)\check{\mathbf{R}}_b^n, \tag{3.44}$$

where

$$\mathbf{R}(\boldsymbol{\epsilon}) = \mathbf{I} + \mathbf{S}(\epsilon). \tag{3.45}$$

The position error is

$$\delta\dot{\mathbf{p}}^n = \delta\mathbf{v}^n. \tag{3.46}$$

The velocity error is

$$\begin{aligned}
\delta\dot{\mathbf{v}}^n &= \mathbf{R}_b^n\mathbf{f}_a^b - \check{\mathbf{R}}_b^n\tilde{\mathbf{f}}_a^b \\
&= (\mathbf{I} + \mathbf{S}(\boldsymbol{\epsilon}))\check{\mathbf{R}}_b^n(\tilde{\mathbf{f}}_a^b + \delta\mathbf{f}_a^b) - \check{\mathbf{R}}_b^n\tilde{\mathbf{f}}_a^b \\
&= \check{\mathbf{R}}_b^n(\tilde{\mathbf{f}}_a^b + \delta\mathbf{f}_a^b) + \mathbf{S}(\boldsymbol{\epsilon})\check{\mathbf{R}}_b^n(\tilde{\mathbf{f}}_a^b + \delta\mathbf{f}_a^b) - \check{\mathbf{R}}_b^n\tilde{\mathbf{f}}_a^b \\
&= \check{\mathbf{R}}_b^n\delta\mathbf{f}_a^b + \mathbf{S}(\boldsymbol{\epsilon})\check{\mathbf{R}}_b^n\tilde{\mathbf{f}}_a^b + \mathbf{S}(\boldsymbol{\epsilon})\check{\mathbf{R}}_b^n\delta\mathbf{f}_a^b.
\end{aligned} \tag{3.47}$$

We want the equations to be linear in the error terms, so products of error terms are ignored, giving

$$\delta\dot{\mathbf{v}}^n = \check{\mathbf{R}}_b^n\delta\mathbf{f}_a^b - \mathbf{S}(\check{\mathbf{R}}_b^n\tilde{\mathbf{f}}_a^b)\boldsymbol{\epsilon} \tag{3.48}$$

where the following property of skew symmetric matrices has been used:

$$\mathbf{S}(\mathbf{a})\mathbf{b} = \mathbf{a} \times \mathbf{b} = -\mathbf{b} \times \mathbf{a} = -\mathbf{S}(\mathbf{b})\mathbf{a} \tag{3.49}$$

The error equation for the orientation is

$$\dot{\boldsymbol{\epsilon}} = \check{\mathbf{R}}_b^n \delta\boldsymbol{\omega}. \tag{3.50}$$

The error equations for the first image coordinate is

$$\begin{aligned}\delta\dot{s}_x = &- v_x^c\zeta + s_x v_z^c\zeta + s_x s_y\omega_x^c - (1+s_x^2)\omega_y^c + s_y\omega_z^c \\ &- \left(-\check{v}_x^c\check{\zeta} + \check{s}_x\check{v}_z^c\check{\zeta} + \check{s}_x\check{s}_y\tilde{\omega}_x^c - (1+\check{s}_x^2)\tilde{\omega}_y^c + \check{s}_y\tilde{\omega}_z^c\right).\end{aligned} \tag{3.51}$$

After replacing the true states with the sum of computed states and error states, and removing products of error states, we have

$$\begin{aligned}\delta\dot{s}_x = &\check{\zeta}\delta v_x^c + \check{s}_x\check{\zeta}\delta v_z^c + (\check{s}_x\check{v}_z^c - \check{v}_x^c)\delta\zeta + (\check{v}_z^c\check{\zeta} + \check{s_y}\tilde{\omega}_x^c - 2\check{s}_x\tilde{\omega}_y^c)\delta s_x \\ &+ (\check{s}_x + \tilde{\omega}_z^c)\delta s_y + \check{s}_x\check{s}_y\delta\omega_x^c - (1+\check{s}_x^2)\delta\omega_y^c + \check{s}_y\delta\omega_z^c.\end{aligned} \tag{3.52}$$

Similarly for the second image coordinate,

$$\begin{aligned}\delta\dot{s}_y = &\check{\zeta}\delta v_y^c + \check{s}_y\check{\zeta}\delta v_z^c + (\check{s}_y\check{v}_z^c - \check{v}_y^c)\delta\zeta + (\check{v}_z^c\check{\zeta} - \check{s_x}\tilde{\omega}_y^c + 2\check{s}_y\tilde{\omega}_x^c)\delta s_y \\ &+ (\check{s}_y + \tilde{\omega}_z^c)\delta s_x + (1+\check{s}_y^2)\delta\omega_x^c - \check{s}_y\check{s}_x\delta\omega_y^c - \check{s}_x\delta\omega_z^c,\end{aligned} \tag{3.53}$$

and the inverse range,

$$\begin{aligned}\delta\dot{\zeta} = &\check{\zeta}^2\delta v_z^c + (2\check{\zeta}\check{v}_z^c + \check{s}_y\tilde{\omega}_x^c - \check{s}_x\omega_y^c)\delta\zeta \\ &- \check{\zeta}\tilde{\omega}_y^c\delta s_x + \check{\zeta}\tilde{\omega}_x^c\delta s_y + \check{\zeta}\check{s}_y\delta\omega_x^c - \check{\zeta}\check{s}_x\delta\omega_y^c.\end{aligned} \tag{3.54}$$

The error equations can be used to find a linear approximation to the influence of the inertial sensor noise on the states, which will give the values for the state covariance matrix in the Kalman filter. Using the error model

$$\delta\boldsymbol{\omega} = \mathbf{b}_g + \boldsymbol{\mu}_g + \mathbf{n}_g, \quad \delta\mathbf{f}_a = \mathbf{b}_a + \mathbf{n}_a \tag{3.55}$$

as presented in section 2.1, and augmenting the error state vector with the sensor bias states, we have

$$\delta\dot{\mathbf{x}} = \mathbf{F}\delta\mathbf{x} + \mathbf{G}\mathbf{n}_{sys}. \tag{3.56}$$

where

$$\mathbf{n}_{sys}^T = [\mathbf{n}_a, \mathbf{n}_g, \mathbf{n}_\mu]. \tag{3.57}$$

The system matrix is given by

$$\mathbf{F} = \begin{bmatrix} \mathbf{F}_{ins} \\ \mathbf{F}_{cam} \end{bmatrix} \tag{3.58}$$

where

$$\mathbf{F}_{ins} = \begin{bmatrix} \mathbf{0} & \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & -\mathbf{S}(\check{\mathbf{R}}_b^n \tilde{\mathbf{f}}_a) & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & -\frac{1}{T}\mathbf{I} & \mathbf{0} \end{bmatrix} \tag{3.59}$$

The noise matrix is given by

$$\mathbf{G} = \begin{bmatrix} \mathbf{G}_{ins} \\ \mathbf{G}_{cam,1} \\ \vdots \\ \mathbf{G}_{cam,N} \end{bmatrix} \tag{3.60}$$

where N is the number of image features,

$$\mathbf{G}_{ins} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \check{\mathbf{R}}_b^n & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \check{\mathbf{R}}_b^n & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{I} \end{bmatrix}, \tag{3.61}$$

and

$$\mathbf{G}_{cam} = \begin{bmatrix} \mathbf{0} & \check{s}_x\check{s}_y & -(1+\check{s}_x^2) & \check{s}_y & \mathbf{0} \\ \mathbf{0} & (1+\check{s}_y^2) & -\check{s}_y\check{s}_x & -\check{s}_x & \mathbf{0} \\ \mathbf{0} & \check{\zeta}\check{s}_y & -\check{\zeta}\check{s}_x & 0 & \mathbf{0} \end{bmatrix}. \tag{3.62}$$

The general solution to (3.56) is [13]

$$\delta\dot{\mathbf{x}}(t) = \mathbf{\Phi}(t,t_0)\delta\mathbf{x}(t_0) + \int_{t_0}^{t} \mathbf{\Phi}(t,\tau)\mathbf{G}(\tau)\mathbf{n}_{sys}(\tau)\,\mathrm{d}\tau, \tag{3.63}$$

where $\mathbf{\Phi}(t,t_0)$ is the state transition matrix from t_0 to t such that

$$\dot{\mathbf{\Phi}}(t,t_0) = \mathbf{F}(t)\mathbf{\Phi}(t,t_0), \qquad \mathbf{\Phi}(t_0,t_0) = \mathbf{I}. \tag{3.64}$$

The solution is used to derive an equivalent discrete-time error model of the form

$$\delta\mathbf{x}_{k+1} = \mathbf{\Phi}_k\delta\mathbf{x}_k + \boldsymbol{\nu}_{sys,k}, \tag{3.65}$$

where $\boldsymbol{\nu}_{sys,k}$ denotes discrete-time noise which has an equivalent effect on the discrete-time state as the $\mathbf{n}_{sys}$ has on the continuous-time state. Comparing (3.63) and (3.65), we have

$$\mathbf{\Phi}_k = \mathbf{\Phi}(t_{k+1},t_k), \tag{3.66}$$

$$\boldsymbol{\nu}_{sys,k} = \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1},\tau)\mathbf{G}(\tau)\mathbf{n}_{sys}(\tau)\,\mathrm{d}\tau. \tag{3.67}$$

The noise vector $\boldsymbol{\nu}_{sys,k}$ has samples from a Gaussian distribution with zero mean, and covariance

$$\mathbf{Q}_k = \mathrm{E}[\boldsymbol{\nu}_{sys,k}\boldsymbol{\nu}_{sys,k}^T] = \int_{t_k}^{t_{k+1}} \mathbf{\Phi}(t_{k+1},\tau)\mathbf{G}(\tau)\check{\mathbf{Q}}(\tau)\mathbf{G}^T(\tau)\mathbf{\Phi}^T(t_{k+1},\tau)\,\mathrm{d}\tau, \tag{3.68}$$

where $\check{\mathbf{Q}}(t)$ is the power spectral density (PSD) of the noise vector $\mathbf{n}_{sys}(t)$ in the continuous-time model:

$$\mathrm{E}[\mathbf{n}_{sys}(t)\mathbf{n}_{sys}(\tau)] = \check{\mathbf{Q}}(t)\delta(t-\tau) \tag{3.69}$$

where $\delta(\cdot)$ is the Dirac delta function. The solution to (3.68) can be approximated by assuming $\mathbf{F}(t)$ to be constant from t_k to t_{k+1}. This gives

$$\mathbf{\Phi}(t_{k+1}, t_k) = e^{\Delta t \mathbf{F}(t_k)}, \tag{3.70}$$

and

$$\mathbf{\Phi}(t_{k+1}, \tau) = e^{(t_{k+1}-\tau)\mathbf{F}(t_k)}. \tag{3.71}$$

We also assume $\mathbf{G}(t)$ and $\check{\mathbf{Q}}$ to be constant over the interval, and get

$$\begin{aligned} \mathbf{Q}_k &= \int_{t_k}^{t_{k+1}} e^{(t_{k+1}-\tau)\mathbf{F}(t_k)} \mathbf{G}(t_k) \check{\mathbf{Q}}(t_k) \mathbf{G}^T(t_k) \left(e^{(t_{k+1}-\tau)\mathbf{F}(t_k)}\right)^T \mathrm{d}\tau \\ &= \int_0^{\Delta t} e^{\theta \mathbf{F}(t_k)} \mathbf{G}(t_k) \check{\mathbf{Q}}(t_k) \mathbf{G}^T(t_k) \left(e^{\theta \mathbf{F}(t_k)}\right)^T \mathrm{d}\theta, \end{aligned} \tag{3.72}$$

where the substitution $\theta = t_{k+1} - \tau$ is used to simplify the integral. This integral can be computed using a method developed by [14], which is based on taking the matrix exponential of a block triangular matrix and combining the sub-matrices of the resulting matrix to get the solution to the integral. Here we use the matrix

$$\mathbf{A} = \begin{bmatrix} -\mathbf{F}(t_k) & \mathbf{G}(t_k)\check{\mathbf{Q}}(t_k)\mathbf{G}^T(t_k) \\ \mathbf{0} & \mathbf{F}^T(t_k) \end{bmatrix}, \tag{3.73}$$

where

$$e^{\Delta t \mathbf{A}} = \mathbf{B}(\Delta t) \begin{bmatrix} \mathbf{B}_{11}(\Delta t) & \mathbf{B}_{12}(\Delta t) \\ \mathbf{0} & \mathbf{B}_{22}(\Delta t) \end{bmatrix}, \tag{3.74}$$

and

$$\mathbf{Q}_k = \mathbf{B}_{22}^T(\Delta t)\mathbf{B}_{12}(\Delta t). \tag{3.75}$$

A simpler, but less accurate, approximation to $\mathbf{Q}_k$ can be computed numerically from the differential equation

$$\dot{\mathbf{Q}}^*(t, t_k) = \mathbf{F}(t)\mathbf{Q}^*(t, t_k) + \mathbf{Q}^*(t, t_k)\mathbf{F}^T(t) + \mathbf{G}(t)\check{\mathbf{Q}}(t)\mathbf{G}^T(t), \tag{3.76}$$

defined for $t \in [t_k, t_{k+1}]$, with the initial condition $\mathbf{Q}^*(t_k, t_k) = \mathbf{0}$, and

$$\mathbf{Q}_k = \mathbf{Q}^*(t_{k+1}, t_k). \tag{3.77}$$

A first order approximation to $\mathbf{Q}_k$, using Euler's method, is then

$$\begin{aligned}\mathbf{Q}_k &\approx \mathbf{Q}^*(t_k, t_k) + (\Delta t)\dot{\mathbf{Q}}^*(t_k, t_k) \\ &= (\Delta t)\mathbf{G}(t_k)\check{\mathbf{Q}}(t_k)\mathbf{G}^T(t_k)\end{aligned} \tag{3.78}$$

This approximation can be used as long as the sample period Δt is short compared to the rate of change of the system states and parameters. [13]

3.4 State initialization

3.4.1 Position

The position is assumed to be given in relation to some known location in the inertial frame, and can either be initialized by GPS measurements or entered manually. If the initial position is obtained from the GPS, the corresponding initial covariance is the same as the measurement covariance of the GPS.

3.4.2 velocity and orientation

The initial estimates of the velocity and orientation of the vehicle and their covariances depend on how well these states are assumed to be known. For the tests that are used here, the velocity and orientation are assumed to be well known and the estimated and actual initial covariances are low.

3.4.3 Landmark

For many practical purposes, the position of the landmark is completely unknown before it is observed by the camera. A way of initializing the feature after a sinlge observation is presented in [3]. At the first observation of a new feature, the estimator state vector is augmented with the states encoding the position of the new feature. The initial bearing estimates are taken directly from the measurements, while the inverse depth is initialized based on the fact that the feature has to be in front of the camera. This is done by choosing an initial value and standard deviation such that a 95 % confidence interval spans a range from close to the camera up to infinity. Emphasis is placed on the importance of including infinity in the confidence interval, even though this may cause the estimated inverse range to become negative. This allows for the utilization of points at infinity, such as stars, to estimate orientation.

The values chosen in [3] is 0.1 for the initial inverse depth estimate, and 0.5 for the initial standard deviation. It should be noted that [3] defines inverse depth as the inverse of the *length* of the vector from the camera to the feature. This differs from the definition used in this thesis, which is the inverse of the *z-coordinate* of the vector in the camera frame, when z is chosen as the optical axis. This definition is the one used by [11] and [2].

4 Description of tests

The estimator was tested using values obtained from the trajectory generator software developed in [15]. This software takes a list of trajectory segments and generates the proper values for position, velocity and orientation given in the global reference frame, and the acceleration and angular velocity given in the body frame of reference.

Following is a description of the different simulations done to test the performance of the system. The results from the tests are presented in chapter 5.

4.1 Model verification and numeric accuracy

The model is verified by computing the state over the simulation interval, using (3.41) with the true specific force and angular velocity as inputs, no added noise and complete knowledge about the initial state. The trajectory that is used here is defined as follows:

- Stand still for 10 seconds
- Accelerate to 5 m/s over 50 meters
- Decelerate back to 0 m/s over 50 meters
- Stand still for 10 seconds

The vehicle moves in a straight line along the x-axis of the body frame, which is always aligned with the inertial frame as there are no rotations in this trajectory. The position, velocity and acceleration over time is shown on figure 4.1.

4.2 Kalman filter assessment

The performance of the Kalman filter is assessed using the same trajectory as in the previous section. For this test gaussian noise is added to the inertial mea-

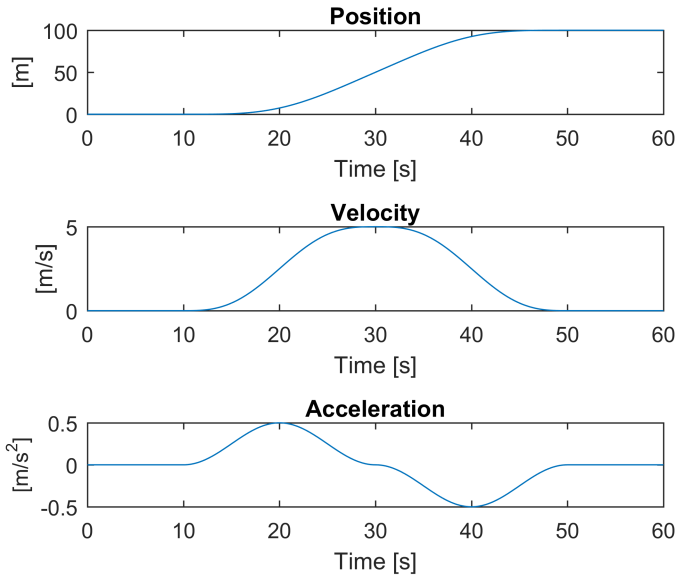

Figure 4.1: Position, velocity and acceleration of vehicle along the test trajectory used for model verification. All movement happens along the x-axis in the inertial frame.

surements and the aiding measurements. Additionally, the initial state estimate is drawn from a normal distribution to simulate uncertainty about the true state. The covariance of the initial state estimate is

$$\mathbf{P}_0 = \begin{bmatrix} \sigma_{\text{P0}}^2\mathbf{I} & & & & & & & & \\ & \sigma_{v0}^2\mathbf{I} & & & & & & & \\ & & \sigma_{\lambda 0}^2\mathbf{I} & & & & & & \\ & & & \sigma_{ba0}^2\mathbf{I} & & & & & \\ & & & & \sigma_{b\omega 0}^2\mathbf{I} & & & & \\ & & & & & \sigma_{\mu 0}^2\mathbf{I} & & & \\ & & & & & & \sigma_{cam0}^2\mathbf{I} & & \\ & & & & & & & \sigma_{\zeta 0}^2 \end{bmatrix}, \tag{4.1}$$

where

$$\sigma_{p0} = 2, \tag{4.2}$$
$$\sigma_{v0} = 0.01, \tag{4.3}$$
$$\sigma_{\lambda 0} = 1 \times 10^{-3}, \tag{4.4}$$
$$\sigma_{ba0} = 0.03, \tag{4.5}$$
$$\sigma_{b\omega 0} = 0.2, \tag{4.6}$$
$$\sigma_{\mu 0} = 1.745 \times 10^{-6}, \tag{4.7}$$
$$\sigma_{cam0} = 0.01, \tag{4.8}$$
$$\sigma_{\zeta 0} = 0.5. \tag{4.9}$$

The position, velocity, orientation and camera coordinates are drawn from the normal distribution, using the true values as the mean. The inertial measurement error states on the other hand are set to zero in the initial estimate, because for these states it is the true values that are randomized. The inverse depth state is also set to a fixed value of 0.1 each time, with a large initial covariance to simulate a lack of knowledge about its true value, as discussed in section 3.4.3. The standard deviations for position, velocity, and camera coordinates are the same as those used to produce the simulated measurements. The same is true for the INS error states, where $\sigma_{\mu 0}$ was calculated using (2.34).

For this test the inertial measurements are given at a rate of 50 Hz, while camera measurements are given at 1 Hz. Additionally position (GPS) and velocity measurements are available at 1 Hz for the first 10 seconds when the vehicle is standing still. The velocity measurements were added together with the position measurements to get a better calibration of the inertial biases before the vehicle starts moving. This is justified by the assumption that the vehicle can be forced to stand still in the beginning so that zero velocity can be assumed with reasonable accuracy during that time. The standard deviation for the noise added to the simulated velocity measurements was set equal to σ_{v0}, which is chose as 1 cm/s. The camera field of view is set to 140 degrees, which is too large to be a realistic value, but is useful in the simulations to study the effects of camera measurements without implementing camera rotations and/or additional landmarks. The landmark is positioned at $(40, 5, 20)$, and is visible to the camera at the start of the trajectory.

4.3 Comparison of different trajectories

Monte Carlo simulations are performed for four different trajectories to study their effects on the accuracy of the estimates. The trajectories that are tested are one with constant velocity past the landmark, one with acceleration and deceleration, one with several stops on the way past, and one with a swaying motion consisting of several turns from side to side.

Monte Carlo simulations consist of running many simulations of a system with random elements and taking the average of the results. For dynamic systems we are interested in looking at the average estimation error and the average estimated standard deviations. The initial state and the noise values are re-generated for each simulation. This allows us to isolate the effects on the estimator performance that are consistent over different possible realizations of the random variables. It also allows us to compare the estimated standard deviation to the actual standard deviation over all the simulated errors. The mean estimation error $\mathbf{m}(\mathbf{e}_k)$, mean standard deviation estimate $\hat{\boldsymbol{\sigma}}_k$, and the actual standard

deviation $\boldsymbol{\sigma}_k$ are computed by

$$\mathbf{m}(\mathbf{e}_k) = \frac{1}{N}\sum_{i=1}^{N}\mathbf{e}_k^i, \tag{4.10}$$

$$\hat{\boldsymbol{\sigma}}_k = \frac{1}{N}\sum_{i=1}^{N}\hat{\boldsymbol{\sigma}}_k^i, \tag{4.11}$$

$$\boldsymbol{\sigma}_k = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(\mathbf{e}_k^i - \mathbf{m}(\mathbf{e}_k^i)\right)}, \tag{4.12}$$

$$\mathbf{e}_k^i = \mathbf{x}_k^i - \hat{\mathbf{x}}_k^i, \tag{4.13}$$

where $\mathbf{x}_k^i$ is the true state at time step k and simulation run number i, $\hat{\mathbf{x}}_k^i$ is the estimated state, and N is the total number of runs. The simulations were done with $N = 100$.

All the trajectories start by standing still for 10 seconds while GPS and velocity measurements are available like in the previous test, after which the camera measurements are the only measurements available to aid the INS. The landmark is placed at $(70, 5, 20)$, which is a bit further away in the x-direction than in the previous test. This causes the landmark to not be visible to the camera until the vehicle reaches a distance along the x-axis determined by

$$p_x^n = p_{L,x}^n - p_{L,z}^n \tan\frac{\theta_{fov}}{2}, \tag{4.14}$$

where $p_{L,i}^n$ is the i-component of the landmark position vector in the inertial frame of reference and θ_{fov} is the angle determining the field of view of the camera. The values used here gives $p_x^n \approx 46$ m. The landmark drops out of view again at $p_x^n \approx 94$ m.

Since the landmark is not visible at $t = 0$, the normalized pinhole coordinates can not be assumed known in the initial estimate as it was in the previous test. Instead, the initial value is set to zero for both coordinates, with a standard deviation of 2, which represents a very large uncertainty.

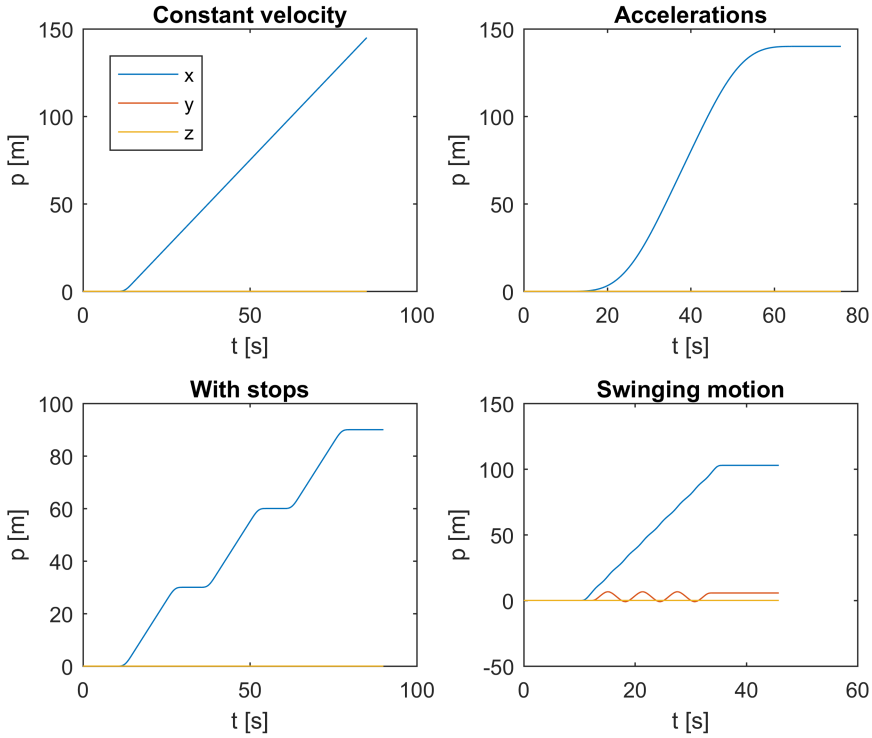

Figure 4.2: The four different position trajectories used for Monte Carlo simulations

4.3.1 Constant velocity

For this test the vehicle is accelerated to 2 m/s over 5 meters, which is before the landmark becomes visible, and after that moves in a straight line at constant velocity. The trajectory is build from the following segments:

- Stand still for 10 seconds
- Accelerate to 2 m/s over 5 meters
- Move at a constant velocity for 70 seconds (140 meters)

The landmark is visible between t = 21 s and t = 74 s.

4.3.2 Acceleration

The vehicle is accelerated slowly up to 5 m/s before it slows back down to zero. The list that defines this trajectory is as follows:

- Stand still for 10 seconds
- Accelerate to 5 m/s over 70 meters
- Decelerate back to 0 m/s over 70 meters
- Stand still for 10 seconds

The landmark is visible between t = 26 s and t = 50 s.

4.3.3 Stops along the way

This trajectory consists of sveral accelerations and decelerations with stops along the way, as defined by the following list:

- Stand still for 10 seconds
- Accelerate to 2 m/s over 5 meters
- Move at a constant velocity for 10 seconds
- Decelerate to 0 m/s over 5 meters
- stand still for 5 seconds
- Accelerate to 2 m/s over 5 meters
- Move at a constant velocity for 10 seconds
- Decelerate to 0 m/s over 5 meters
- stand still for 5 seconds
- Accelerate to 2 m/s over 5 meters

- Move at a constant velocity for 10 seconds
- Decelerate to 0 m/s over 5 meters
- Stand still for 10 seconds

The landmark is visible between t = 21 s and t = 90 s.

4.3.4 Swaying motion

The last trajectory consists of several turns back and forth in the xy-plane, defined as follows:

- Stand still for 10 seconds
- Accelerate to 5 m/s over 5 meters
- Turn 45 degrees left
- Turn 90 degrees right
- Turn 90 degrees left
- Turn 90 degrees right
- Turn 90 degrees left
- Turn 90 degrees right
- Turn 90 degrees left
- Turn 45 degrees right
- Decelerate to 0 m/s over 5 meters
- Stand still for 10 seconds

The landmark is visible between t = 13 s and t = 45 s.

4.4 Higher inertial sensor inaccuracy

The test described in the previous sections are done using the "typical" values for the MTi 10-series in [9] as a basis for generating the inertial measurements and the Kalman filter system covariance matrix. An additional Monte Carlo simulation is done using the "max" values for the same series of sensors, in order to study the effects of increased inaccuracy of the accelerometer and gyroscope. The relevant values from [9] are shown in table 4.1. The trajectory used in this test will be the one that gives the best results in the trajectory comparison tests.

Table 4.1: Relevant performance specifications for the MTi 10-series inertial sensors

		Typical	Max
Bias repeatability (gyro)	$[^\circ/\mathrm{h}]$	0.2	0.5
In-run bias stability (gyro)	$[^\circ/\mathrm{h}]$	18	-
Noise density (gyro)	$[^\circ/\mathrm{s}/\sqrt{\mathrm{Hz}}]$	0.03	0.05
Bias repeatability (accel)	$[\mathrm{m}/\mathrm{s}^2]$	0.03	0.05
Noise density (accel)	$[\mu g/\sqrt{\mathrm{Hz}}]$	80	150

5 Results

5.1 Model verification and numeric accuracy

As figure 5.1 shows, the numeric error reaches 0.1 m in the x-direction and is then reduced back to 0. The increase in error happens during acceleration, while the decrease happens during deceleration. The error is consistent with what should be expected from numerical integration with Euler's method, thus verifying that the model is correct, at least for this type of trajectory.

5.2 Estimator assessment

Figure 5.2 shows that the filter was able to estimate the position with relatively good accuracy over the entire trajectory. The error in position stays below 5 meters in each direction as shown on figure 5.3, and the error in velocity stays below 0.5 m/s as shown on figure 5.4. Figure 5.5 shows the error and standard deviation in the normalized pinhole coordinates s_x and s_y, and the inverse depth ζ. The inverse depth is estimated within 20 seconds, when it reaches a standard deviation of around 2×10^{-3}. Figure 5.6 shows that good estimates are aquired for the gyroscope biases during the first 10 seconds when the vehicle is standing still, although the z-axis bias estimate has a larger error than the bias on the other two axes. The accelerometer bias on the z-axis is estimated very closely, while the y- and x-axis bias estimates have larger standard deviation. All the plots show estimation errors that seem to correspond well to the estimated standard deviations. Comparing figure 5.3 to figure 5.7 shows that the camera measurements were able to effectively constrain drift in the position estimates.

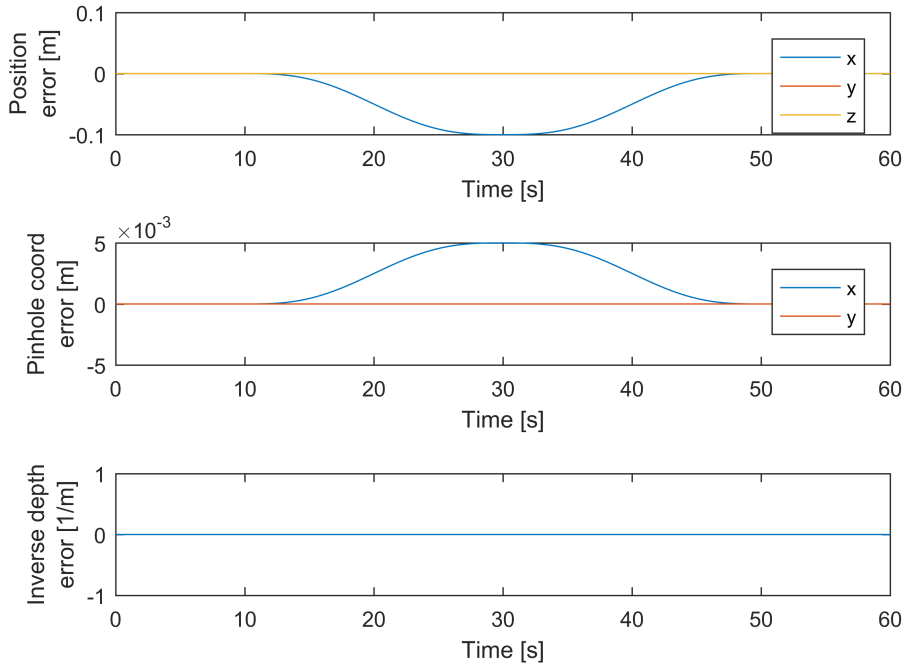

Figure 5.1: Errors in states predicted from INS with known initial state and no added noise.

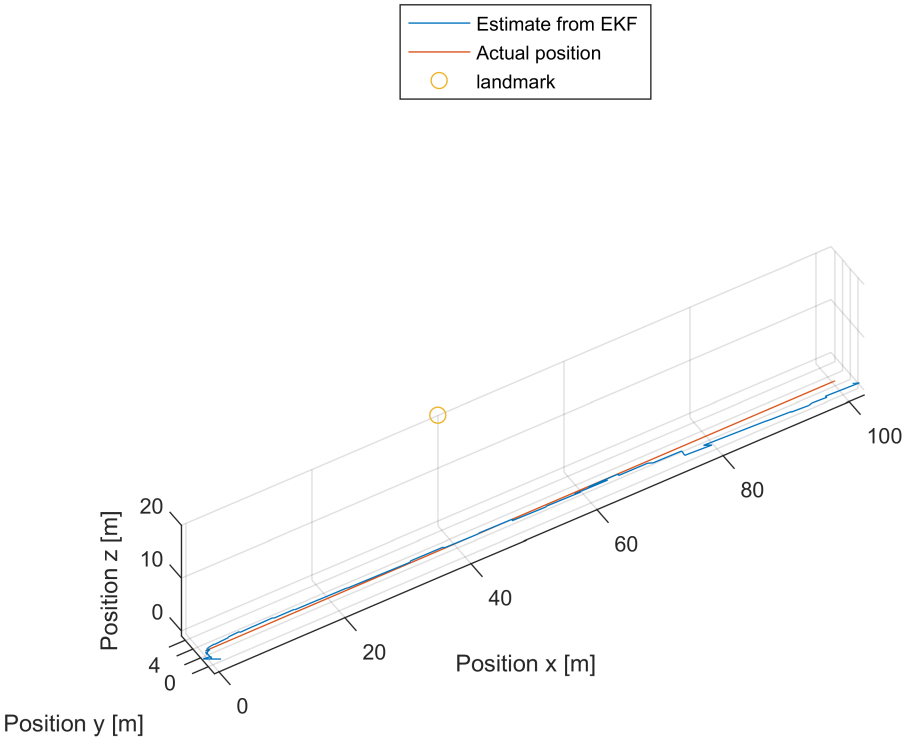

Figure 5.2: Estimated and actual trajectory shown in relation to the landmark position.

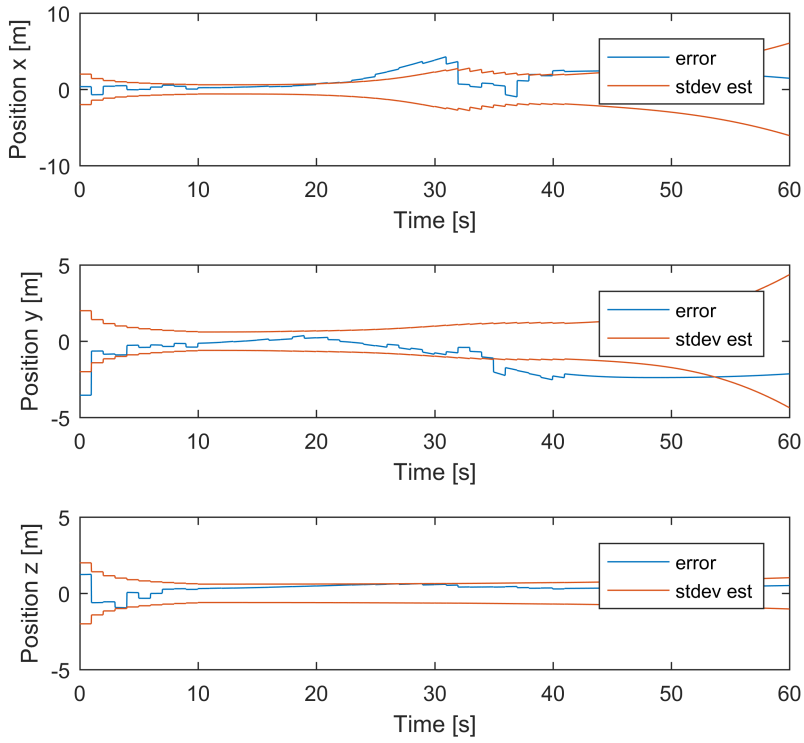

Figure 5.3: Error in estimated position shown with estimated standard deviation.

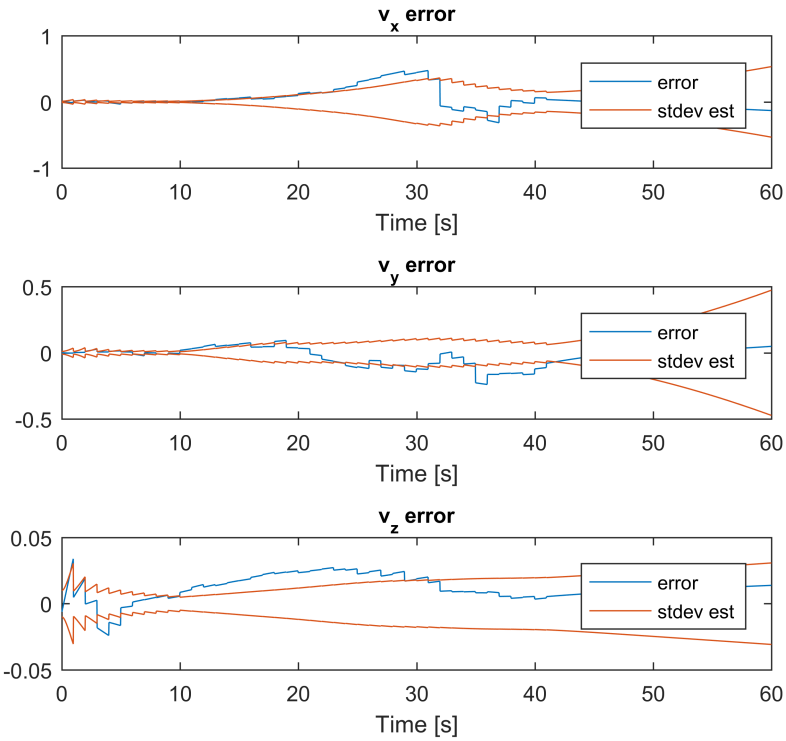

Figure 5.4: Error in estimated velocity shown with estimated standard deviation.

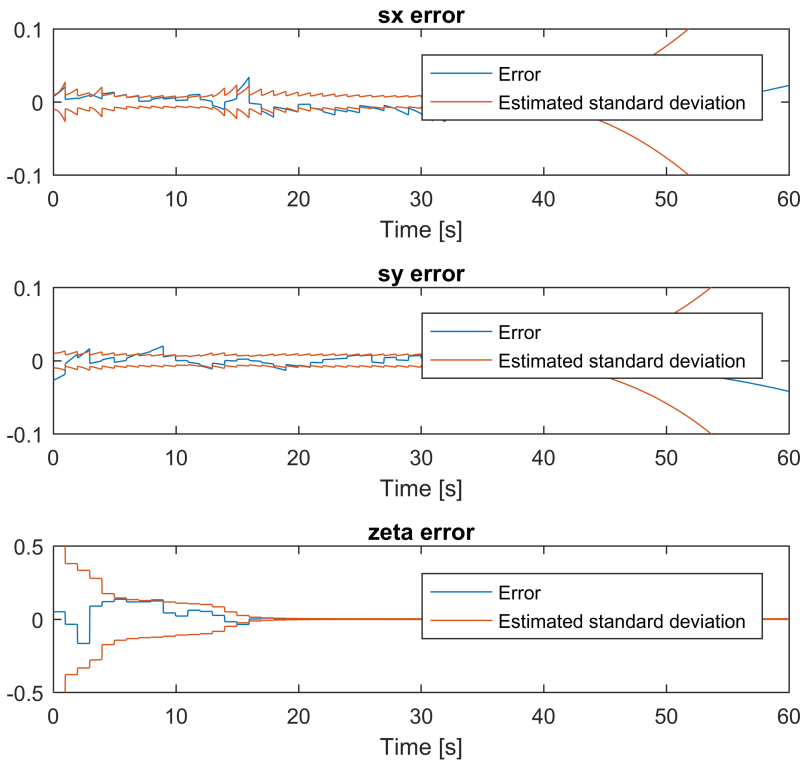

Figure 5.5: Error in estimated camera coordinates and inverse depth shown with estimated standard deviation.

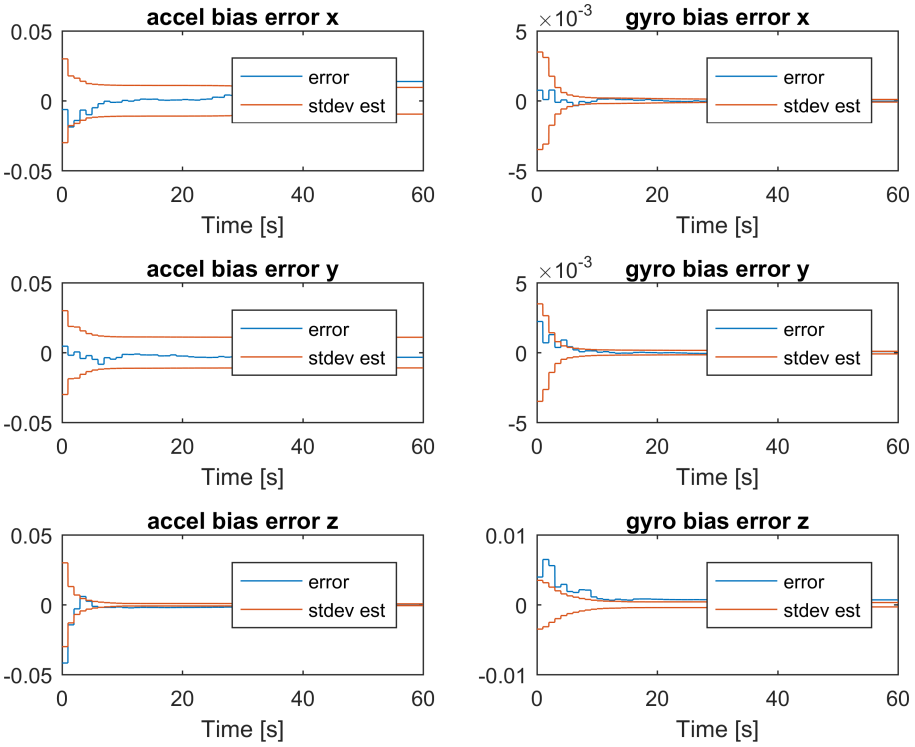

Figure 5.6: Error in INS biases shown with estimated standard deviation.

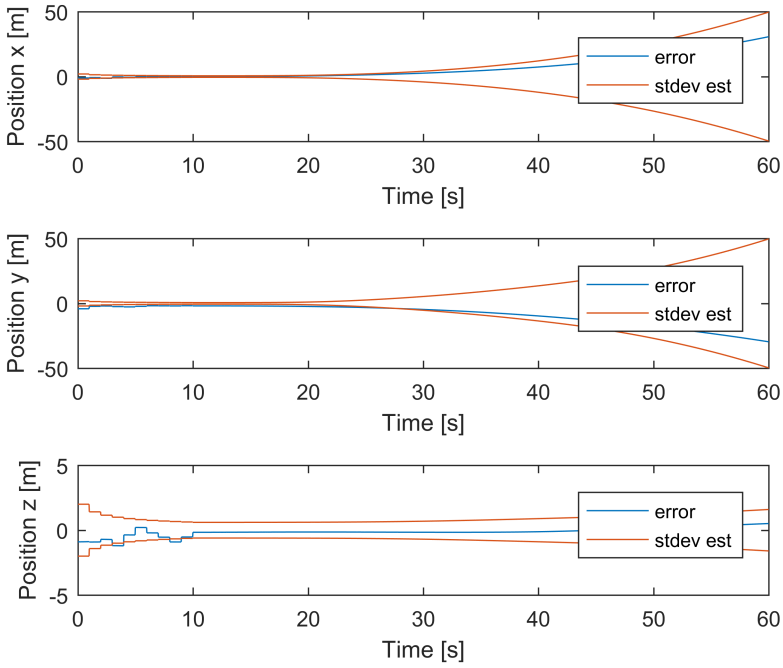

Figure 5.7: Position error when camera measurements are disabled.

5.3 Comparison of different trajectories

By looking at figures 5.9, 5.11, 5.13 and 5.15, it seems that the estimated range to the landmark converges quicker when the vehicle is moving with acceleration than when it moves at constant velocity. Making stops along the way seems to result in a worse performance than a single smooth acceleration. The quickest convergence is seen for the swaying motion, with an error in the inverse depth of less than 0.01 at the second camera measurement. The inverse depth error seems to approach zero for all trajectories except for the constant velocity trajectory, where there is a residual error of about 0.01.

Figures 5.8, 5.10, 5.12, and 5.14 show the results for position and velocity. The swaying trajectory has the smallest mean errors when comparing all the trajectories over the same time intervals. The standard deviation for the position and velocity errors are diverging for all the trajectories, even where the estimated standard deviation seems to have converged to a steady state. The actual standard deviation is also generally larger over the whole trajectory.

The fact that the estimated standard deviation is too small can be explained by the fact that it is obtained through a linearization of a highly nonlinear model. Linearization inevitably introduces errors, and these errors are not accounted for by the values in the covariance matrices that are given to the Kalman Filter. It could be possible to improve performance by increasing the assumed noise in the filter model for the states that are most affected by linearization errors, although it might be better to implement an unscented Kalman filter [16,17] and/or particle filter [18] to avoid the linearization altogether.

5.4 Higher inertial sensor inaccuracy

The test using less accurate inertial sensor specifications were done using the swaying trajectory (see figure 4.2), because this trajectory seemed to give the best estimation results. When using the highter noise values, the results of 11 of the 100 Monte Carlo runs had to be removed because of nonsensical values for the inverse range. The results were removed when the absolute value of the estimated inverse range was greater than 100, which corresponds to the landmark being closer than one centimeter in the z-direction.

Figure 5.16 shows a higher standard deviation on the x- and y-components

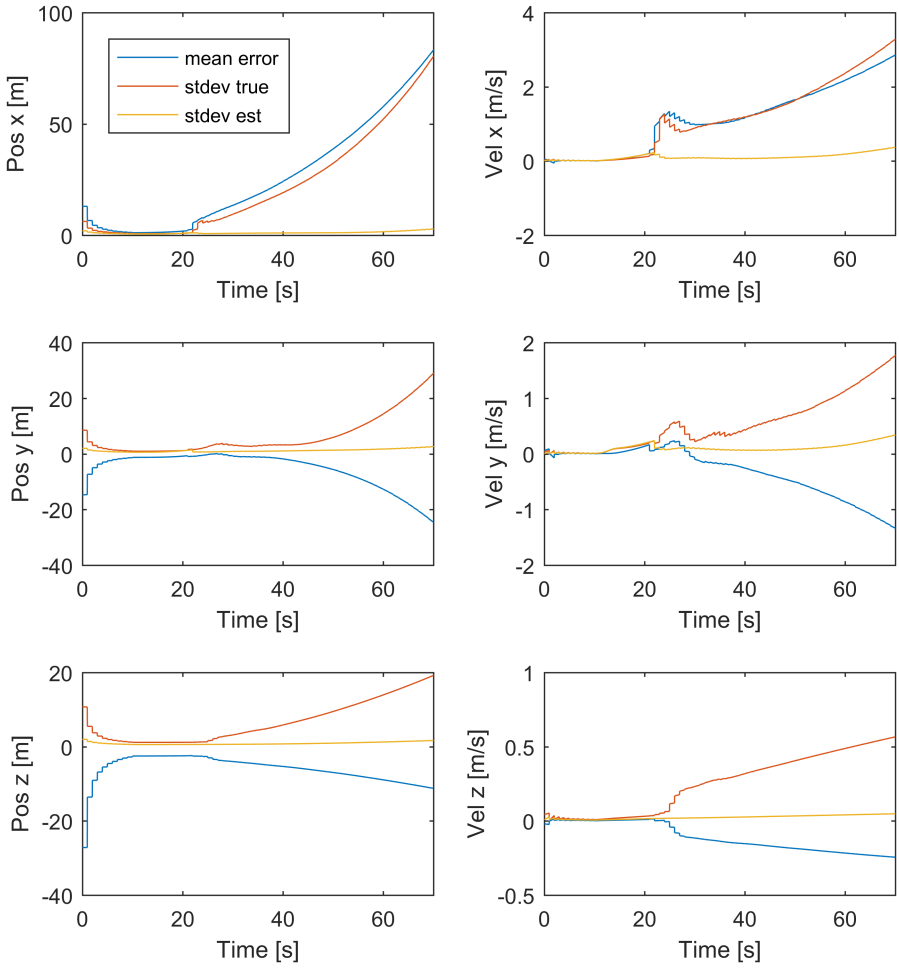

Figure 5.8: Monte Carlo results for position and velocity when moving at a constant velocity of 2 m/s past the landmark.

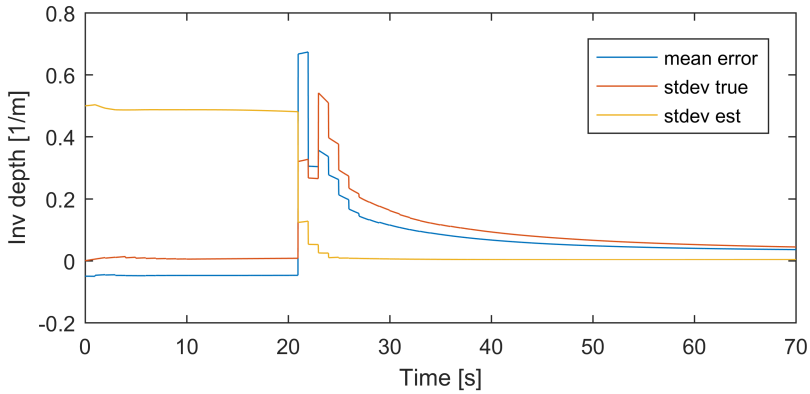

Figure 5.9: Monte Carlo results for inverse depth when moving at a constant velocity.

of the position and velocity errors. The z-component of the position error has a mean value that is much larger in the beginning, but after the initial calibration the error does not seem significantly greater than in the previous test with the same trajectory. The inverse feature range has an error after the second camera update that is slightly larger than in the previous test, but after the third camera update the mean error and true standard deviation are reduced to less than 0.01.

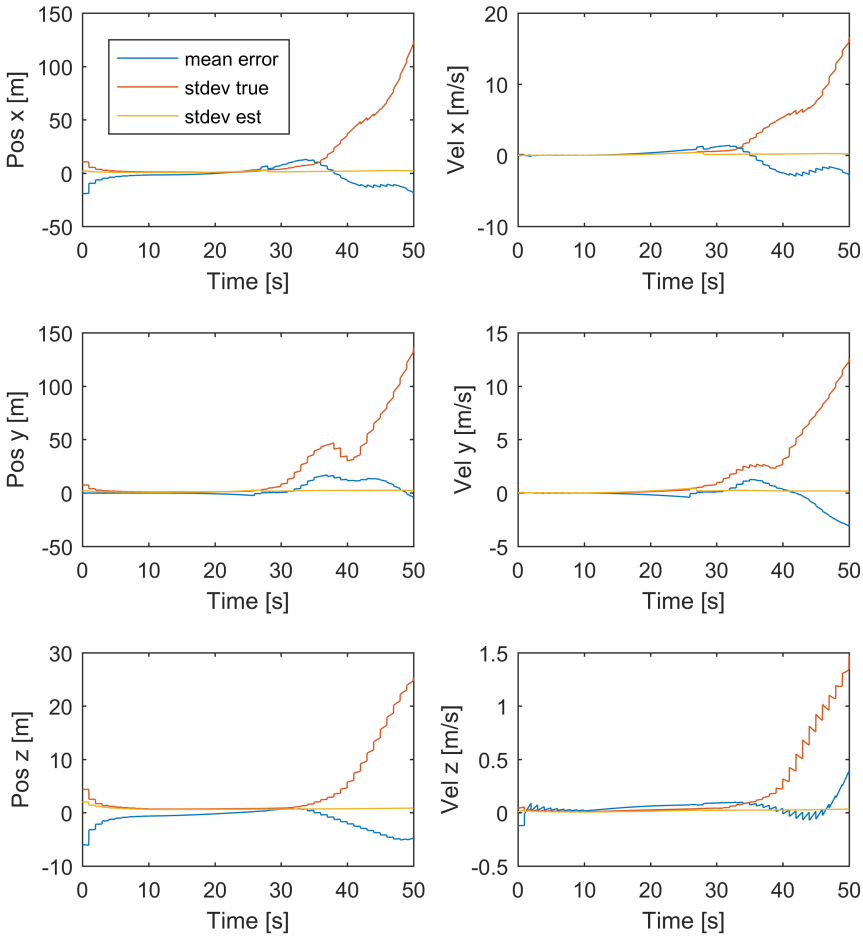

Figure 5.10: Monte Carlo results for position and velocity when moving with acceleration.

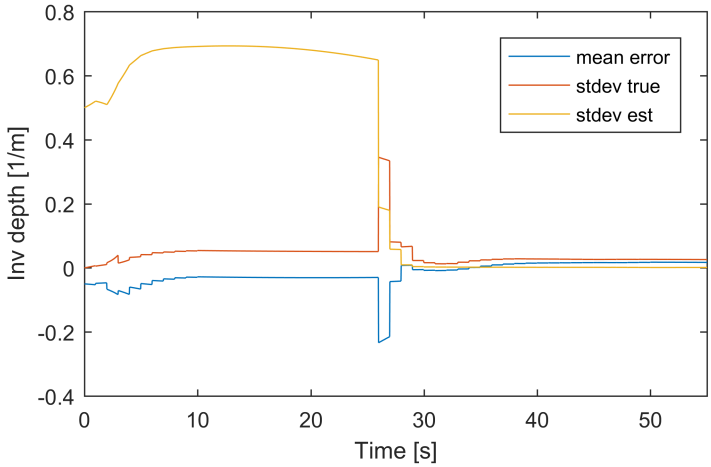

Figure 5.11: Monte Carlo results for inverse depth when moving with acceleration.

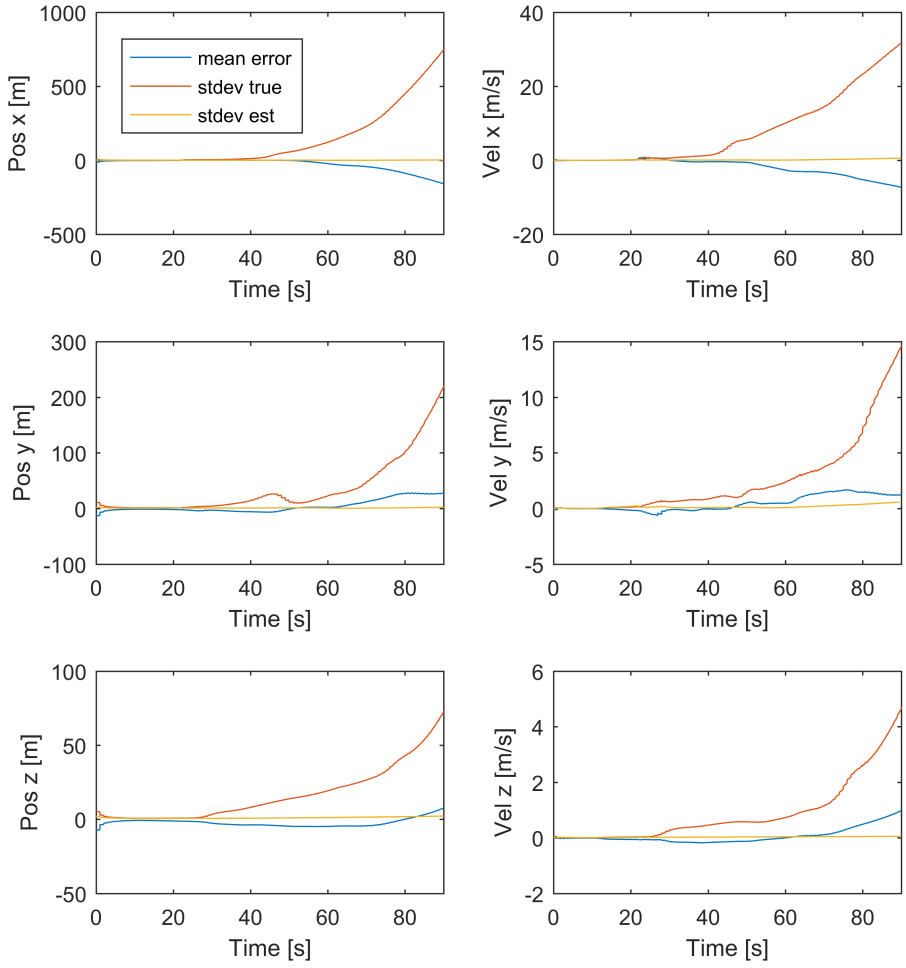

Figure 5.12: Monte Carlo results for position and velocity when moving with several stops underway.

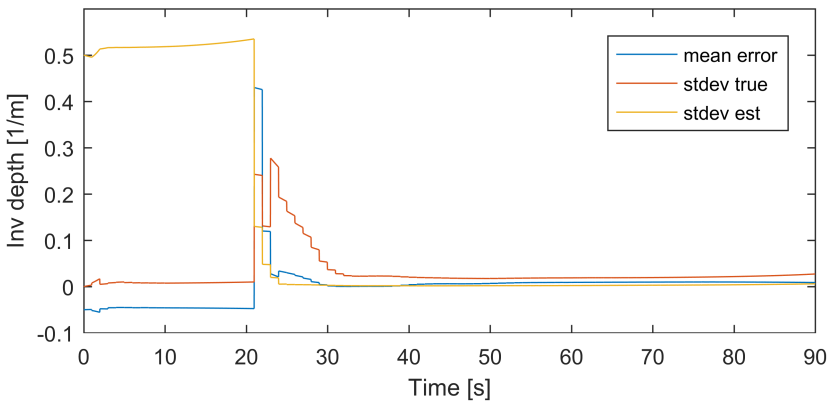

Figure 5.13: Monte Carlo results for inverse depth when moving with stops underway.

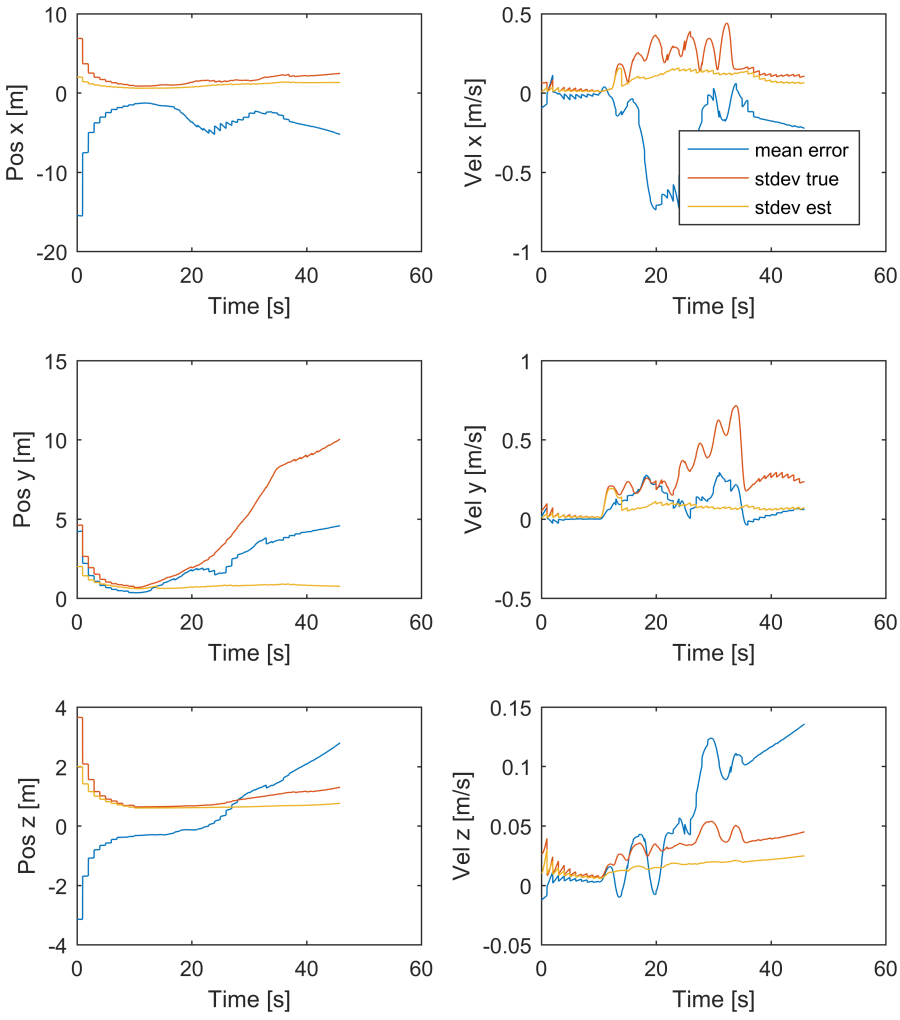

Figure 5.14: Monte Carlo results for position and velocity when swinging from side to side.

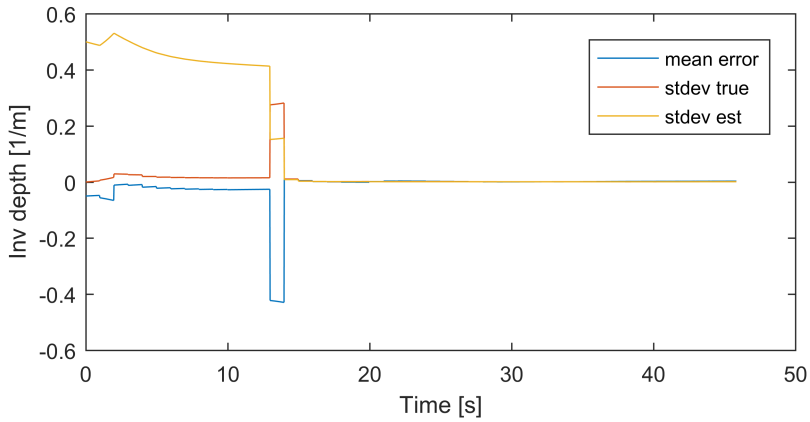

Figure 5.15: Monte Carlo results for position and velocity when swinging from side to side.

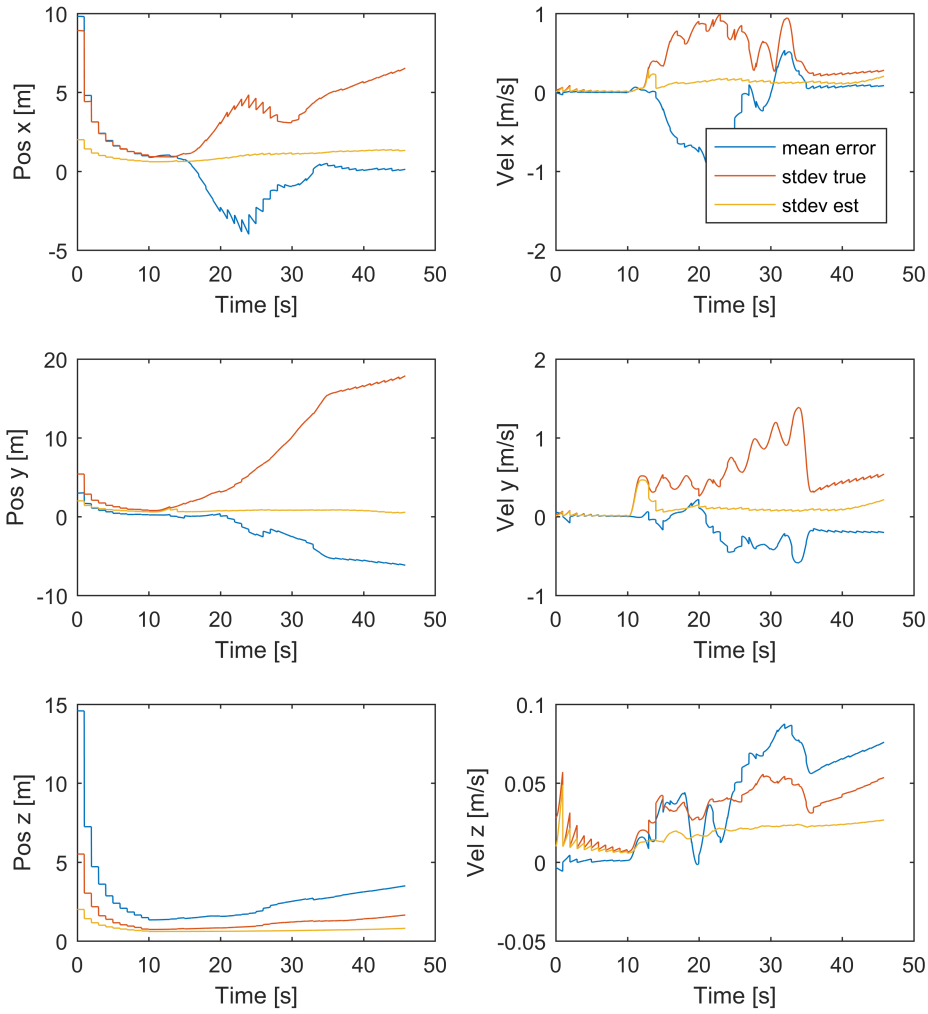

Figure 5.16: Position and velocity when using higher noise values for the swaying trajectory.

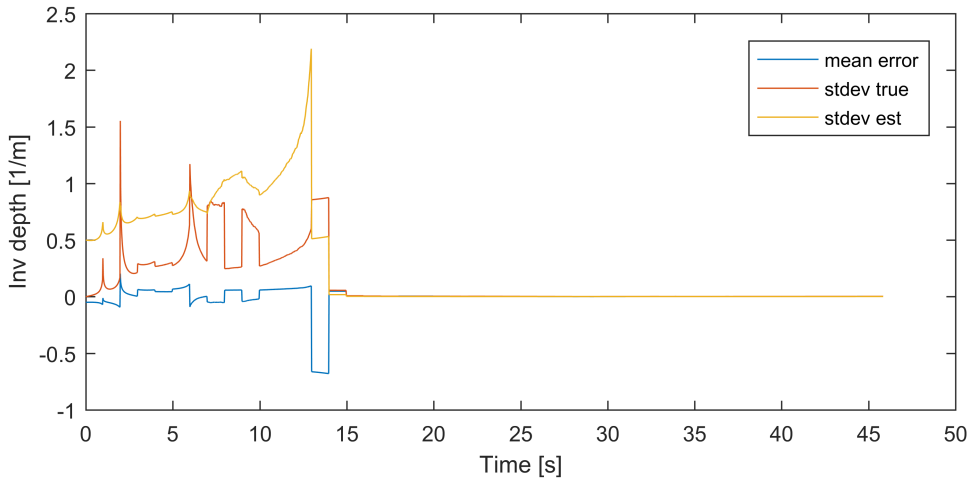

Figure 5.17: Inverse feature range when using higher noise values for the swaying trajectory.

6 Discussion

The state estimator that was developed has shown at least somewhat promising results with respect to the ability to constrain drift in an inertial navigation system. In first test that was done with the Kalman filter, the landmark was within view of the camera from the beginning and camera measurements were available together with the initial position and velocity measurements, and the initial estimate of the pinhole coordinates was chosen with the same accuracy as the measurements. In this situation the error in the position estimate was less than five meters over the 60 second trajectory and did not seem to be increasing. For the Monte Carlo simulations on the other hand, the landmark was positioned further away so that it was not visible at the start. This necessarily causes some drift in the estimated states between the period when GPS is available and when the landmark becomes visible.

It is interesting to note that for all the three trajectories that only have movement along the x-axis, the drift in the estimates actually seem to become larger when the camera measurements become available. This effect is particularly visible for the constant velocity trajectory, although the trajectory with stops underway has the largest drift, with a standard deviation of several hundred meters even though the camera measurements are available.

The estimated standard deviations however, are small and can not be seen to diverge, at least not to any significant degree when compared to the actual standard deviations. This means that the simulated system has behaviors that are not properly captured by the filter. The reason for this may be that the higher order terms in the state equations were significant during these tests and that the linearized equations used in the Kalman filter therefore were not accurate.

The discretization of the equations is also a form of linearization as it is based on Eulers method. Linearization of the system equations can be avoided by using an unscented Kalman filter, so it is possible that this would have given better results than the extended Kalman filter which has been used here. It is also possible that using a higher order Runge Kutta method [6] for discretizing the equations would give better results.

The swaying trajectory stands out from the others in that the standard deviation of the x-component of the position error seems to stabilize at a low value, and the y- and z-components are also much lower than for the other three trajectories. It could be that the estimates here are better simply because the landmark became visible earlier, as the speed was somewhat higher on this trajectory, but this alone can probably not explain why there is almost no divergence at all in some of the components. Rather it would seem that the swaying movement of the vehicle increases the accuracy of the estimator. This may be explained by the fact that time-varying angular velocities are introduced, and that there is more movement of the landmark's projection in the image plane. In fact, because of the alignment of the camera frame with the body frame, the swaying trajectory is the only trajectory where both image coordinates are changing, and this might affect the ability of the filter to use these states when estimating the velocity and position. Even though this trajectory gave better estimates however, the estimated standard deviations were still much smaller than the actual ones, suggesting that the extended Kalman filter is probably not the best choice.

7 Conclusion

In this thesis a state space model for a simplified camera and position measurement aided navigation system has been developed and used as a basis for implementing an extended Kalman filter to estimate the system states. Simulations were done using a previously developed trajectory generator, and measurement values were created based on inertial sensor product specifications.

The results show that such a system can be able to restrain drift in inertial navigation. The performance seem to be dependent on the type of trajectory followed by the vehicle, with a swaying motion from side to side showing the best results. It is also shown that the estimates become less accurate when less accurate inertial sensors are used, which confirms intuitive expectations.

The inverse feature range, which is used to represent the distance to the landmark, is estimated quickly and with small error for all the tests, so that given a relatively accurate camera measurements the system should have all information needed to give good estimates for position. Despite this, several of the tests resulted in large and growing estimation errors, indicating that too much trust is placed on the inertial measurements compared to the camera measurements. This assumption is supported by the fact that the estimated standard deviations remain small even as the actual estimation errors increase.

Suggestions for further work include doing more simulations to study the effects of when the landmark becomes visible, as all the Monte Carlo simulations were done when the landmark was not visible at the beginning of the trajectory. Simulations of the same trajectories under the same conditions but with the landmark becoming visible earlier or later may provide useful insights. Implementing another type of estimator, like the unscented Kalman filter, should

also be tried.

Furhter, the implementation used in this thesis was based on a single landmark only, but it should be relatively easy to expand it to include several landmarks. This will be necessary for most practical applications as new landmarks are needed as the old ones fall out of the field of view. It is also probable that the accuracy of the estimator could increase as more landmarks are observed at the same time. More simulations could be done to test this.

References

[1] Andreas Huster. *Relative position sensing by fusing monocular vision and inertial rate sensors.* PhD thesis, Stanford University, July 2003.

[2] Michael George and Salah Sukkarieh. Inertial navigation aided by monocular camera observations of unknown features. In *2007 IEEE International Conference on Robotics and Automation*, pages 3558–3564. IEEE, 2007.

[3] Javier Civera, Andrew J Davison, and JM Martinez Montiel. Inverse depth parametrization for monocular SLAM. *IEEE transactions on robotics*, 24(5):932–945, 2008.

[4] Agostino Martinelli. *Closed-Form Solutions for Attitude, Speed, Absolute Scale and Bias Determination by Fusing Vision and Inertial Measurements.* PhD thesis, INRIA, 2011.

[5] Christian Horn. Modellering av MEMS-treghetssensorer i et navigasjonssystem. Master's thesis, Universitetet i Oslo, 2015.

[6] Olav Egeland and Tommy Gravdahl. *Modeling and Simulation for Automatic Control.* Marine Cybernetics AS, Trondheim, Norway, 2003.

[7] Steven M. LaValle. *Yaw, pitch, and roll rotations.* Cambridge University Press, April 2012. `http://planning.cs.uiuc.edu/node102.html` Accessed 27. January 2016.

[8] Jay Farrell. *Aided Navigation: GPS with High Rate Sensors.* McGraw-Hill, Inc., New York, NY, USA, 1st edition, 2008.

[9] Xsens Technologies B.V. *MTi User Manual*, January 2014.

[10] Robert Grover Brown and Patrick Y. C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises*. John Wiley & Sons, Inc, fourth edition, 2012.

[11] Andreas Huster, Eric W Frew, and Stephen M Rock. Relative position estimation for AUVs by fusing bearing and inertial rate sensor measurements. In *OCEANS'02 MTS/IEEE*, volume 3, pages 1863–1870. IEEE, 2002.

[12] Yi Cao. Learning the Extended Kalman Filter. `http://www.mathworks.com/matlabcentral/fileexchange/18189-learning-the-extended-kalman-filter`, 2008. Accessed 15. April 2016.

[13] Peter S. Maybeck. *Stochastic models, estimation and control*, volume 1. Academic Press, London, UK, 1979.

[14] Charles Van Loan. Computing integrals involving the matrix exponential. Technical report, Cornell University, 1977.

[15] Tor Erik Fredrikstad. Bildestøttet treghetsnavigasjon. Engineering cybernetics specialization project, February 2016.

[16] Simon J Julier and Jeffrey K Uhlmann. New extension of the kalman filter to nonlinear systems. In *AeroSense'97*, pages 182–193. International Society for Optics and Photonics, 1997.

[17] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, pages 153–158. IEEE, 2000.

[18] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking. *IEEE Transactions on signal processing*, 50(2):174–188, 2002.

[19] Paul A. M. Dirac. *The Principles of Quantum Mechanics.* Oxford University Press, fourth edition, 1958.

[20] Eric W. Weisstein. From MathWorld–A Wolfram Web Resource. `http://mathworld.wolfram.com/KroneckerDelta.html`.

[21] Arthur Gelb. *Applied optimal estimation.* MIT press, 1974.

A Mathematical background

A.1 Gaussian random variables

A Gaussian random variable x, also called a normal random variable, has the probability density function

$$p(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{-(x-m)^2/(2\sigma^2)}, \tag{A.1}$$

where m is the mean and σ is the standard deviation. A realization of a Gaussian random variable has the highest probability of being close to the mean and valus further away from the mean are less likely. A Gaussian distribution can often be assumed when modelling uncertainty, as the total uncertainty can be seen as sum of many different contributions, and the sum of several random variables added together always tends toward a normal distribution. This is known as the central limit theorem. [10]

A.2 Autocorrelation and power spectral density

The autocorrelation function of a random process $x(t)$ is

$$R(t_1, t_2) = \mathrm{E}[x(t_1)x(t_2)] \tag{A.2}$$

and tells how much the process at t_1 is correlated with itself at t_2. If the process is stationary, meaning that its probability density function does not change over time, the autocorrelation can be written as

$$R(\tau) = \mathrm{E}[x(t)x(t+\tau)]. \tag{A.3}$$

Information about the frequency content of the random process can be obtained from $R(\tau)$ by taking the Fourier transform, which results in the power spectral density function of the process [10]

$$S(j\omega) = \mathfrak{F}\{R(\tau)\} = \int_{-\infty}^{\infty} R(\tau)e^{-j\omega\tau}\,\mathrm{d}\tau \tag{A.4}$$

A.3 White noise

White noise is a continuous-time random signal with a constant power spectral density function $S(j\omega) = S$, meaning that the amplitude of the signal does not depend on frequency. This means that the autocorrelation function is

$$R(\tau) = S\delta(\tau) \tag{A.5}$$

where $\delta(t)$ is the Dirac delta function, which is defined by $\int_{-\infty}^{\infty} \delta(t)\,\mathrm{d}t = 1$ and $\delta(t) = 0$ for $t \neq 0$ [19]. By this definition, any sample of the white noise has infinite variance, which does not make physical sense. It is still a useful model however, as when the white noise is used as an input to a model of a physical system the bandwidth of the signal is limited, so the output will be physically realizable. [10]

Continuous-time white noise is analogous to a discrete-time white sequence, which is a sequence of zero-mean, uncorrelated random variables. Since the variables are uncorrelated, the covariance of the white sequence can be written as

$$\mathrm{E}[x_k x_l] = \sigma^2 \delta_{kl}, \tag{A.6}$$

where δ_{kl} is known as the Kronecker delta, and is defined as $\delta_{kl} = 1$ for $k = l$ and $\delta_{kl} = 0$ for $k \neq l$ [20].

A.4 Markov processes

A continouos-time process $x(t)$ is first-order Markov if its probability distribution function $P[x(t)]$ satisfies

$$P[x(t_k)|x(t_{k-1}), ..., x(t_1)] = P[x(t_k)|x(t_{k-1})] \tag{A.7}$$

for all k and $t_1 < t_2 < ... < t_k$. A first-order Markov process can be represented by a differential equation of the form

$$\dot{x}(t) = -\beta(t)x(t) + w(t) \tag{A.8}$$

where $w(t)$ is white noise. [21]

B Trajectory generator

The trajectory generator developed in [15] creates a trajectory from a list segments. Each trajectory segment consists of a position $\mathbf{p}^a(t)$, velocity $\mathbf{v}^a(t)$, and acceleration $\mathbf{a}^a(t)$, given in a local stationary reference frame that coincides with the body frame at the beginning of the segment. The segments also have an associated rotation $\mathbf{R}_b^a(t)$ and a body frame acceleration $\mathbf{a}^b(t)$ and angular velocity $\boldsymbol{\omega}^b(t)$. Below are the mathematical definitions of each segment used in the tests in this report.

B.1 Straight segment

Inputs

V Desired speed at the end of the segment [m/s]
L Desired length of the segment [m]

Intermediate calculations

Total time from beginning to end of segment:

$$T = \frac{2L}{(V_0 + V)}, \tag{B.1}$$

where V_0 is the vehicle's speed at the beginning of the segment.
Maximal acceleration:

$$A = \frac{V - V_0}{T} \tag{B.2}$$

A parameter used for nicer equations:

$$\beta = \frac{2\pi}{T} \tag{B.3}$$

Outputs

$$\mathbf{a}^a(t) = \begin{bmatrix} A(1-\cos\beta t) \\ 0 \\ 0 \end{bmatrix} \tag{B.4}$$

$$\mathbf{v}^a(t) = \begin{bmatrix} V_0 + A(t-(\sin\beta t)/\beta) \\ 0 \\ 0 \end{bmatrix} \tag{B.5}$$

$$\mathbf{p}^a(t) = \begin{bmatrix} V_0 t + A(t^2/2 + (\cos\beta t - 1)/(\beta^2)) \\ 0 \\ 0 \end{bmatrix} \tag{B.6}$$

$$\mathbf{R}_b^a(t) = \mathbf{I} \tag{B.7}$$

$$\mathbf{a}^b(t) = \mathbf{a}^a(t) \tag{B.8}$$

$$\boldsymbol{\omega}^b(t) = \mathbf{0} \tag{B.9}$$

B.2 Constant velocity

This trajectory segment is the same as the straight segment, but defined by time instead of length, and with the speed restricted to $V = V_0$.

Inputs

T Desired time to spend on the segment [s]

Outputs

$$\mathbf{a}^a(t) = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{B.10}$$

$$\mathbf{v}^a(t) = \begin{bmatrix} V_0 & 0 & 0 \end{bmatrix}^T \tag{B.11}$$

$$\tag{B.12}$$

$$\mathbf{p}^a(t) = \begin{bmatrix} V_0 t & 0 & 0 \end{bmatrix}^T \tag{B.13}$$

$$\mathbf{R}_b^a(t) = \mathbf{I} \tag{B.14}$$

$$\mathbf{a}^b(t) = \mathbf{a}^a(t) \tag{B.15}$$

$$\boldsymbol{\omega}^b(t) = \mathbf{0} \tag{B.16}$$

B.3 Turn

Turns are made up of Euler-spiral segments, which has a curvature that increases linearly with the distance along the segment, in order to avoid jumps in the sideways acceleration. Each turn consists of first a regular Euler-spiral segment and then a reverse Euler-spiral segment appended to it. The reverse Euler-spiral has a curvature that *decreases* with the distance along the segment.

Inputs

$\theta_{end} = \theta(t_{end})$	Desired angle of the turn [°]
r	Desired turn radius [m]

B.3.1 Euler-spiral segment

Intermediate calculations

Total time from beginning to end of segment:

$$T = \frac{2r\theta_{end}}{2V}, \tag{B.17}$$

where V is the vehicle's speed at the beginning of the segment.
Acceleration:

$$A = \frac{V^2}{r} \tag{B.18}$$

Angle as a function of time:

$$\theta(t) = \frac{At^2}{2VT} \tag{B.19}$$

Outputs

$$\mathbf{a}^a(t) = \frac{At}{T} \begin{bmatrix} -\sin\theta(t) \\ \cos\theta(t) \\ 0 \end{bmatrix} \tag{B.20}$$

$$\mathbf{v}^a(t) = V \begin{bmatrix} \cos\theta(t) \\ \sin\theta(t) \\ 0 \end{bmatrix} \tag{B.21}$$

$$\tag{B.22}$$

$$\mathbf{p}^a(t) = V \int_0^t \begin{bmatrix} \cos\theta(t) \\ \sin\theta(t) \\ 0 \end{bmatrix} \mathrm{d}\tau \tag{B.23}$$

$$\mathbf{R}_b^a(t) = \begin{bmatrix} \cos\theta(t) & -\sin\theta(t) & 0 \\ \sin\theta(t) & \cos\theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.24}$$

$$\mathbf{a}^b(t) = (At/T) \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T \tag{B.25}$$

$$\boldsymbol{\omega}^b(t) = (At/VT) \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{B.26}$$

B.3.2 Reverse Euler-spiral segment

Intermediate calculations

Angle as a function of time:

$$\theta(t) = \frac{A}{V}\left(t - \frac{t^2}{2T}\right) \tag{B.27}$$

The parameters A and T are the same as for the regular Euler-spiral segment.

Outputs

$$\mathbf{a}^a(t) = \frac{A(T-t)}{T} \begin{bmatrix} -\sin\theta(t) \\ \cos\theta(t) \\ 0 \end{bmatrix} \tag{B.28}$$

$$\mathbf{v}^a(t) = V \begin{bmatrix} \cos\theta(t) \\ \sin\theta(t) \\ 0 \end{bmatrix} \tag{B.29}$$

$$\tag{B.30}$$

$$\mathbf{p}^a(t) = V \int_0^t \begin{bmatrix} \cos\theta(t) \\ \sin\theta(t) \\ 0 \end{bmatrix} \mathrm{d}\tau \tag{B.31}$$

$$\mathbf{R}_b^a(t) = \begin{bmatrix} \cos\theta(t) & -\sin\theta(t) & 0 \\ \sin\theta(t) & \cos\theta(t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.32}$$

$$\mathbf{a}^b(t) = (A(T-t)/T) \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T \tag{B.33}$$

$$\boldsymbol{\omega}^b(t) = (A(T-t)/VT) \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T \tag{B.34}$$

C Matlab code

The most important Matlab files are included here. The program also depends on a number of smaller functions that are not all included.

Main routine

```
function cam_aided_ins3()

addpath('fordypning','fordypning/generate_trajectory','fordypning/common');

% ********************************************************************* %
%   Parameters
%
% ********************************************************************* %
g = 9.81; % percieved acceleration due to gravity [m/s^2]
FREQ_INS = 50; % Samples per second
FREQ_CAM = 1; % Images per second
FREQ_GPS = 1; % Position updates per second
GPS_TIME = 10; % Time before the gps becomes unavailable [s]
RANDOM_INIT = 1; % Toggle randomized initial state

ND_GYRO = 0.05*[1;1;1]; % Noise density on gyroscope [degree/sqrt(Hz)] 0.03
ND_ACCEL = 150*[1;1;1]; % Noise density on accelerometer [micro g/sqrt(Hz)] 80

BIAS_STD_GYRO = 0.5*[1;1;1]; % Standard deviation of gyro bias [deg/s] 0.2
BIAS_STD_GYRO = deg2rad(BIAS_STD_GYRO); % Converted to [rad/s]

BIAS_STD_ACCEL = 0.05*[1;1;1]; % Standard deviation of accel. bias [m/s^2] 0.03

BIAS_STAB_GYRO = 18*[1;1;1]; % Bias stability [deg/h] for Markov model 18
BIAS_TIME_GYRO = 100; % Time constant for gyro bias Markov model [seconds]

```

```
STD_CAM = 0.01; % Standard deviation of camera measurement noise 0.01
STD_GPS = 2; % Standard deviation of GPS measurement [meters] 2
STD_VEL = 0.01; % Standard deviation of velocity measurement [m/s] 0.01

LANDMARK = [70;5;20]; % Position of landmark relative to inertial frame
CAM_FOV = 140; % Camera field of view [degrees]
CAM_FOV = deg2rad(CAM_FOV); % Converted to [rad]
%CAM_PIXELSIZE = 3.45e-6; % Pixel size [meters]
%CAM_FOCALLENGTH = 12e-3; % Focal length of camera [meters]
%CAM_RESOLUTION = [2448;2048]; % Image resolution [pixels]
%CAM_ORIENTATION = [cos(0); [0;1;0]*sin(0)]; % Quaternion for camera
                                              % orientation relative
                                              % to body frame

% Assign index values for each state variable and each measurement:
pn = 1:3; % Vehicle position in global frame
vn = 4:6; % Velocity in global frame
angles = 7:9; % Orientation in roll-pitch-yaw Euler angles
ba = 10:12; % Accelerometer bias
bw = 13:15; % Gyroscope bias
mu = 16:18; % Gyroscope bias instability
sx = 19; % Normalized image coordinates of landmark
sy = 20;
zeta = 21; % Inverse depth
n_states = 21; % Total number of states
n_sysnoise = 9; % Number of elements in noise vector for ins measurements

msx = 1; msy = 2; % Measured image coordinates
mpn = 3:5; % Measured global position
mvn = 6:8; % Measured global velocity
n_measurements = 8; % Total number of measurements

% Assumed standard deviations for measurement noise:
 measurement_stdevs = zeros(n_measurements,1);
 measurement_stdevs(msx) = STD_CAM;
 measurement_stdevs(msy) = STD_CAM;
 measurement_stdevs(mpn) = STD_GPS*[1;1;1];
 measurement_stdevs(mvn) = STD_VEL*[1;1;1];

% Initial states:
x0 = zeros(n_states,1);
```

```
x0(sx) = 0;  % Image feature x-coordinate
x0(sy) = 0;  % Image feature y-coordinate
x0(zeta) = 0.1;  % Inverse feature range
x0(vn) = [0;0;0];  % Velocity wrt inertial frame, in body frame coordinates
x0(ba) = [0;0;0];  % Accelerometer bias
x0(angles) = [0;0;0];  % Direction of gravity in body frame
x0(bw) = [0;0;0];  % Gyroscope bias
x0(pn) = [0;0;0];
x0(mu) = [0;0;0];

sample_time_ins = 1/FREQ_INS;
if FREQ_CAM == 0
  ratio_ins_cam = 0;
else
  ratio_ins_cam = FREQ_INS/FREQ_CAM;
end
if FREQ_GPS == 0
  ratio_ins_gps = 0;
else
  ratio_ins_gps = FREQ_INS/FREQ_GPS;
end

% Steady-state covariance for gyro bias stability:
covar_stab_gyro = ((2*pi/(360*3600))*diag(BIAS_STAB_GYRO)).^2;

% Power spectral densities for inertial sensor noise:
psd_gyro = diag((ND_GYRO*2*pi/360).^2);
psd_accel = diag((ND_ACCEL*1e-6/g).^2);
psd_stab_gyro = (2/BIAS_TIME_GYRO)*covar_stab_gyro;
Q_psd = blkdiag(...
    psd_accel, psd_gyro, psd_stab_gyro);

% Initial covariance for Kalman Filter
P0 = 0*eye(n_states);
P0(pn,pn) = (STD_GPS^2)*eye(3);
P0(vn,vn) =  (STD_VEL^2)*eye(3);
P0(angles,angles) = 1e-6*eye(3);
P0(ba,ba) = diag(BIAS_STD_ACCEL).^2;
P0(bw,bw) = diag(BIAS_STD_GYRO).^2;
P0(mu,mu) = sample_time_ins*psd_stab_gyro*eye(3);
P0(sx,sx) = 2^2;
```

```
108 P0(sy,sy) = 2^2;
109 P0(zeta,zeta) = 0.5^2;
110
111 % ********************************************************************* %
112 %   True values
    %
113 % ********************************************************************* %
114 % Generate a pre defined trajectory:
115 [t,...  % Time vector
116  p_n_true,... % Actual position wrt global (inertial) reference frame
117  v_n_true,... % Actual velocity wrt global frame
118  ~,... % Actual acceleration wrt global frame
119  q_true,... % Actual orientation given by quaternions
120  a_b_true,... % Actual acceleration wrt body frame
121  w_true] ... % Actual angular velocity about the body frame axes
122   = generate_trajectory(FREQ_INS);
123
124 n_samples = size(t,2);
125
126 % Relative landmark position from true position, rotated to body frame:
127 r_n_true = bsxfun(@minus,LANDMARK,p_n_true);
128 r_b_true = r_n_true; % Allocate storage
129 for k = 1:n_samples
130   r_b_true(:,k) = quaternionrotation(r_n_true(:,k),...
131                                      quatconjugate(q_true(:,k)));
132 end
133
134 % True normalized pinhole coordinates for landmark:
135 pinhole_coords_true =...
136   bsxfun(@times,[r_b_true(1,:);r_b_true(2,:)],(1./r_b_true(3,:)));
137
138 inversedepth_true = 1./r_b_true(3,:);
139
140 % Velocity in body frame coordinates:
141 v_b_true = zeros(3,n_samples);
142 for k=1:n_samples
143   v_b_true(:,k) = quaternionrotation(v_n_true(:,k),...
144                                      quatconjugate(q_true(:,k)));
145 end
146
147 % Convert the quaternions from the trajectory generator to
```

```
% roll-pitch-yaw-angles to compare with estimated states:
rpy_angles_true = zeros(3,size(q_true,2));
for k=1:size(rpy_angles_true,2)
  R = rotmatrix_from_quaternion(q_true(:,k));
  rpy_angles_true(:,k) = rpy_angles_from_rotmatrix(R);
end

% ********************************************************************* %
%   Monte Carlo simulation
%
% ********************************************************************* %
n_runs = 100;
%mc_states = zeros(n_states,n_samples,n_runs); % all estimated states
mc_stdevs = zeros(n_states,n_samples,n_runs); % all estimated stdevs
mc_errors = zeros(n_states,n_samples,n_runs); % estimation error for all runs
%mc_truth = zeros(n_states,n_samples,n_runs); % true states for all runs
deleted = 0;
wait = waitbar(0,'Running Monte Carlo simulation');

i = 1;
while i <= size(mc_errors,3)
  [state_est,mc_stdevs(:,:,i),state_true] = simulation_run();

  k = 1;
  while k <= n_samples
    % Do a check for nonsensical values of the inverse depth estimate
    if (state_est(zeta,k) > -100) && (state_est(zeta,end) < 100)
      % estimate is okay for current k
      if k == n_samples
        % estimate was okay for all k, save errors, go to next mc run
        mc_errors(:,:,i) = state_true - state_est;
        i = i + 1;
      end
      % go to next timestep
      k = k + 1;
    else
      % estimate not okay, remove from array, do not increment index
      mc_stdevs(:,:,i) = [];
      mc_errors(:,:,i) = [];
      deleted = deleted + 1;

```

```
188       % Go to next simulation run
189       k = n_samples + 1;
190     end
191   end
192   clear k;
193
194   waitbar(i/n_runs)
195   %x_est = state_est;
196   %sigma_est = mc_stdevs(:,:,i);
197   %x_true = state_true;
198 end
199 close(wait)
200
201 %error_est = x_est - x_true;
202 errors_mean = mean(mc_errors,3); % Use the mean of the estimated states
203 stdevs_true = std(mc_errors,0,3); % Actual standard deviation of estimates
204
205 %x_true = mean(mc_truth,3); % Mean of true values
206 sigma_est = mean(mc_stdevs,3); % Mean of estimated stdevs
207
208
209 % ********************************************************************* %
210 %   Functions
    %
211 % ********************************************************************* %
212
213   function [x_est,sigma_est,x_true] = simulation_run()
214     % ********************************************************************%
215     %   Measurements
    %
216     % ********************************************************************%
217     % Generate inertial measurements from sensor model using known trajectory
218     [a_b_measured,w_measured,accel_bias,gyro_bias,gyro_markov] = ...
219       sensor_model(a_b_true,w_true,q_true,Q_psd,BIAS_TIME_GYRO,...
220                    BIAS_STD_GYRO,BIAS_STD_ACCEL,sample_time_ins);
221
222     u = [w_measured; a_b_measured]; % Inertial measurements used as inputs
223
224     % Generate camera measurements using known trajectory
225     % First image at timestep = the ratio of ins to cam frequency
226     % q_cam = quatmultiply(q_true', CAM_ORIENTATION')';
```

```
227       %
228       %  cam_meas = camera_model_pinhole(...
229       %                    p_n_true, q_cam, LANDMARK, STD_CAM, ratio_ins_cam,...
230       %                    CAM_PIXELSIZE, CAM_FOCALLENGTH, CAM_RESOLUTION);
231       %cam_meas = pinhole_coords_true;
232       cam_meas = camera_model2(pinhole_coords_true,ratio_ins_cam,STD_CAM,CAM_FOV);
233
234       cam_meas_times = t(~isnan(cam_meas(1,:)));
235       if ~isempty(cam_meas_times)
236         disp(['first cam meas at t=',num2str(cam_meas_times(1))]);
237         disp(['last cam meas at t=',num2str(cam_meas_times(end))]);
238       end
239
240       gps_meas = NaN*zeros(3,n_samples);
241       vel_meas = NaN*zeros(3,n_samples);
242       k = 1;
243       while k <= n_samples && t(k) <= GPS_TIME
244         if ~mod(k,ratio_ins_gps)
245           % Add position and velocity measurements at the proper time instants
246           gps_meas(:,k) = p_n_true(:,k) + STD_GPS*randn(3,1);
247           vel_meas(:,k) = v_n_true(:,k) + STD_VEL*randn(3,1);
248         end
249         k = k + 1;
250       end
251       clear k;
252
253       % Save true state vector
254       x_true = zeros(n_states,n_samples);
255       x_true(pn,:) = p_n_true;
256       x_true(vn,:) = v_n_true;
257       x_true(angles,:) = rpy_angles_true;
258       x_true(ba,:) = accel_bias;
259       x_true(bw,:) = gyro_bias;
260       x_true(mu,:) = gyro_markov;
261       x_true(sx:sy,:) = pinhole_coords_true;
262       x_true(zeta,:) = inversedepth_true;
263
264
265       % ********************************************************************* %
266       % %  Kalman Filter
    %
```

```
267     % ********************************************************************* %
268     % Camera measurement parameters:
269     %focallength_eff = CAM_FOCALLENGTH/CAM_PIXELSIZE;
270     %image_origin = CAM_RESOLUTION/2;
271
272     % Initial state covariance:
273     P = P0;
274
275     % Draw inital state from normal distribution to simulate uncertainty:
276     if RANDOM_INIT
277       x0 = x0 + sqrt(P)*randn(n_states,1);
278       x0(zeta) = 0.1;
279       x0([ba;bw;mu]) = zeros(9,1);
280     else
281       % Everything known except ins bias
282       x0 = x_true(:,1);
283       x0([ba;bw;mu]) = zeros(9,1);
284     end
285
286     x = x0; % Initial state
287     x_est = zeros(n_states,n_samples); % Estimated states
288     %x_mech = zeros(n_states,n_samples);
289     sigma_est = zeros(n_states,n_samples); % Estimated standard deviations
290     kovar_est = zeros(n_states,n_samples);
291     kovar_est(:,1) = diag(P);
292     sigma_est(:,1) = sqrt(diag(P));
293     x_est(:,1) = x;
294     %x_mech(:,1) = x;
295     %f_ukf = @(x,u,n) f3(u,x,sample_time_ins,Rcb);
296     % H = zeros(2,n_states);
297     % H(1,sx) = 1;
298     % H(2,sy) = 1;
299   %  landmark_seen = 0; % set this flag when landmark is seen first time
300     for k=2:n_samples
301         f = @(x) f3(u(:,k-1),x,sample_time_ins);
302         Q = sys_covariance(x); % System covariance matrix
303
304         % Cam measurements are taken when k is an integer multiple of the
305         % ratio of INS freq to cam freq, similar with GPS-measurements
306         if ~isnan(cam_meas(1,k)) ...
307             && ~isnan(gps_meas(1,k))
```

```
308             % Both cam and gps measurement available
309             z = [cam_meas(:,k); gps_meas(:,k); vel_meas(:,k)];
310             h = @(x) h_cam_gps(x); %,focallength_eff,image_origin);
311             R = diag(measurement_stdevs.^2);
312             %R = eye(5);
313
314           elseif ~isnan(cam_meas(1,k))
315             % Only ins and camera measurement available
316             z = cam_meas(:,k);
317             h = @(x) h_cam(x);%,focallength_eff,image_origin);
318             R = diag(measurement_stdevs(msx:msy).^2);
319             %R = eye(2);
320           elseif ~isnan(gps_meas(1,k))
321             % Only ins and gps measurement available
322             z = [gps_meas(:,k); vel_meas(:,k)];
323             h = @(x) h_gps(x); % Measurement function
324             R = diag(measurement_stdevs([mpn,mvn]).^2);
325           else % Only ins-measurements available
326             z = [];
327             h = @(x) [];
328             R = [];
329
330           end
331           % Extended Kalman filter:
332           [x, P] = ekf(f,x,P,h,z,Q,R); % Get estimated state and covariance
333
334           x_est(:,k) = x; % Save state estimates
335           sigma_est(:,k) = sqrt(diag(P)); % Save standard deviation estimates
336           kovar_est(:,k) = diag(P);
337
338           % INS mechanization
339           % Use estimated bias and noise values
340       %     x_mech([ba,bw,mu],k-1) = x_est([ba,bw,mu],k-1);
341       %     x_mech(:,k) = f(x_mech(:,k-1));
342
343       end
344     end
345
346     function Q = sys_covariance(state)
347       Rnb = rotmatrix_from_rpy(state(angles));
348       G_ins = [zeros(3,n_sysnoise);blkdiag(Rnb,Rnb,zeros(3));
```

```
            zeros(3,n_sysnoise);zeros(3,6),eye(3);];

    G_cam = [state(sx)*state(sy),   -1-state(sx)^2,  state(sy);
             1+state(sy)^2,    -state(sx)*state(sy), -state(sx);
             state(zeta)*state(sy), -state(zeta)*state(sx), 0];
    G_cam = [zeros(3),G_cam,zeros(3)];

    G = [G_ins;G_cam];
    Q = sample_time_ins*G*Q_psd*G';
  end

  function z = h_cam_gps(state) %,focallength_eff,image_origin)
    z = zeros(8,1);
    z(1) = state(sx);%*focallength_eff + image_origin(1); % Image feature x-c
    z(2) = state(sy);%*focallength_eff + image_origin(2); % Image feature y-c
    z(3:5) = state(pn); % Vehicle position in global frame
    z(6:8) = state(vn); % Velocity in global frame
  end

  function z = h_cam(state)%,focallength_eff,image_origin)
    z = state(sx:sy);
  end

  function z = h_gps(state)
    z = zeros(6,1);
    z(1:3) = state(pn); % Vehicle position in global frame
    z(4:6) = state(vn); % Velocity in global frame
  end

end
```

State transition function

```
function next_state = f3(old_input,old_state,timestep)
  % Discrete time state transition function via Euler's method
  next_state = old_state + timestep*f_c(old_input, old_state);

  % Heun's method:
  %next_state = old_state ...
  %    + timestep*(f_c(old_input,old_state) + f_c(new_input,x_euler))/2;

```

```
function dx = f_c(u,x) % State derivative function
  % Continuous time state transition function
  g = 9.81;

  vn = x(4:6); % Velocity given in global frame coordinates
  rpy_angles = x(7:9); % Roll pitch yaw Euler angles
  ba = x(10:12); % Accelerometer bias
  bw = x(13:15); % Gyroscope bias
  mu = x(16:18); % Gyroscope bias stability
  sx = x(19); % Image feature x-coordinate
  sy = x(20); % Image feature y-coordinate
  zeta = x(21); % Inverse feature range

  Rnb = rotmatrix_from_rpy(rpy_angles); % Rotation from body to global
  vc = (Rnb')*vn; % Velocity in camera frame coordinates

  % Angle derivative matrix from Huster:
  pitch = rpy_angles(2);
  yaw = rpy_angles(3);
  E = [cos(yaw)/cos(pitch), sin(yaw)/cos(pitch), 0;
       -sin(yaw), cos(yaw), 0;
       sin(pitch)*cos(yaw)/cos(pitch), sin(pitch)*sin(yaw)/cos(pitch),1];

  omega_meas = u(1:3); % Measured angular velocity from gyro
  fa_meas = u(4:6); % Measured specific force from accelerometer

  omega = omega_meas - bw;% - mu; % Angular velocity
  fa = fa_meas - ba; % Specific force

  dpn = vn;
  dvn = Rnb*fa - [0;0;g];
  drpy_angles = E*Rnb*omega;
  dba = [0;0;0];
  dbw = [0;0;0];
  dmu = -(1/100)*mu;
  dsx = -vc(1)*zeta + sx*vc(3)*zeta + sx*sy*omega(1) ...
          - (1 + sx^2)*omega(2) + sy*omega(3);

  dsy = -vc(2)*zeta + sy*vc(3)*zeta + (1 + sy^2)*omega(1) ...
          - sx*sy*omega(2) - sx*omega(3);

```

```
50      dzeta = vc(3)*(zeta^2) + zeta*sy*omega(1) - zeta*sx*omega(2);
51
52      dx = [dpn;dvn;drpy_angles;dba;dbw;dmu;dsx;dsy;dzeta;];
53
54    end
55
56  end
```

Camera measurement generation

```
1  function cam_meas = camera_model2(pinhole_coords,ratio,stdev,fov)
2
3  n_samples = size(pinhole_coords,2);
4  cam_meas = NaN*ones(2,n_samples);
5
6  max_sx = tan(fov/2);
7  max_sy = tan(fov/2);
8
9  for k = 1:n_samples
10   if ~mod(k,ratio)
11       s_x = pinhole_coords(1,k);
12       s_y = pinhole_coords(2,k);
13       if (abs(s_x) <= max_sx) && (abs(s_y) <= max_sy)
14         cam_meas(:,k) = pinhole_coords(:,k) + stdev*randn(2,1);
15       end
16   end
17 end
```