1 INTRODUCTION

This report is primarily intended for users of NavLab, but it can also be read to get general information about the software.

With an increasing number of different NavLab users with different ranges of application, NavLab is regularly upgraded with new functionality. Consequently, this document is extended accordingly, to include descriptions of the new features. The current version of this report covers the main functionality in NavLab as of November 2003.

2 NAVLAB OVERVIEW

This chapter gives a brief overview of NavLab and its usage, and is also relevant for readers not planning to use NavLab themselves. Definitions of the notation and coordinate systems used are found in Appendix A. In Appendix F a list of abbreviations and acronyms is given.

2.1 What is NavLab?

NavLab (Navigation Laboratory) is a powerful and versatile tool intended for navigation system research and development, navigation system accuracy analysis and navigation data post-processing.

Figure 2.1 shows the structure of NavLab, which consists of two main parts:

Simulator

The Trajectory Simulator can simulate any vehicle trajectory specified by the user, and true position, velocity, attitude etc are calculated. In addition, the user specifies a set of available sensors and their characteristics. Based on this information, the sensor simulators add characteristic errors to the true values, and by this calculate a set of artificial sensor measurements.

Estimator

Based on the available sensor measurements, the Estimator makes both Kalman filtered and smoothed estimates of position, velocity, attitude and sensor errors. This is done by first integrating the IMU (Inertial Measurement Unit) measurements in the navigation equations and comparing the result with the aiding sensors that are available. The differences are then sent as measurements to the Kalman filter (see section 2.4 and 2.5 for more details). Note that the measurements can be either simulated (from the NavLab Simulator) or real (from the sensors of a vehicle).

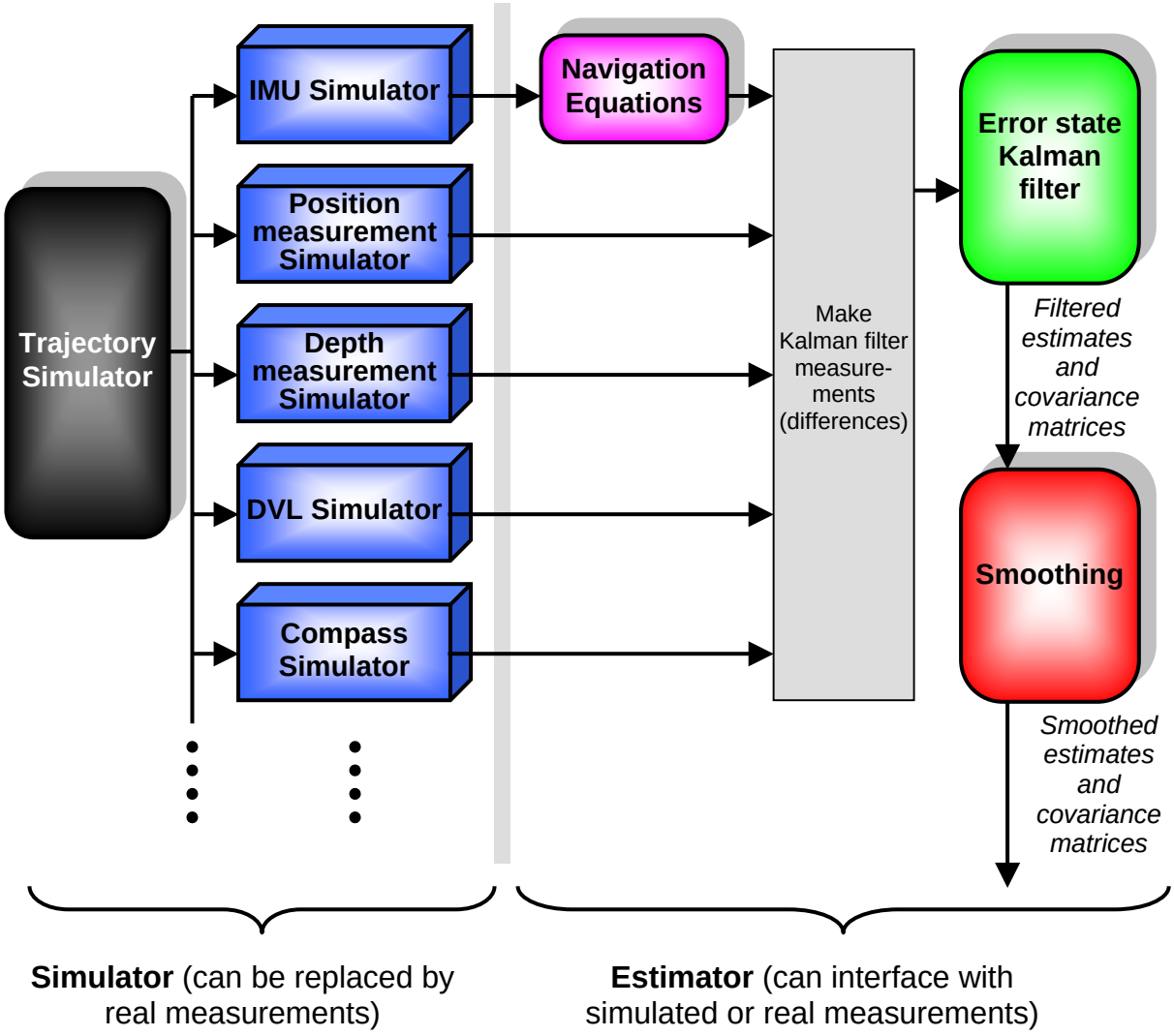

Figure 2.1 *NavLab structure. Note: The colors used in the figure correspond to the colors of the graphs generated by the different parts of NavLab (blue is the measurement, red is the smoothed estimate etc).*

2.1.1 NavLab's theoretical foundation

The most significant feature of NavLab is probably its solid theoretical foundation. NavLab is the result of an innovative research process to establish a completely generic theoretical basis for navigation and for implementation of navigation systems. The development has lead to the following contributions:

- A new stringent and unified system for notation and mathematical representation
- A unified design and implementation of algorithms and aiding techniques for the Kalman filter, where statistical optimality is maintained throughout the entire system
- Elimination of numerical problems by
  - Deducing and implementing exact formulas (rather than approximations)
  - Using only singular free representations
  - Controlling accumulation of Matlab's inherent round-off errors

Many articles from the above work will be published, but currently the most relevant report available is (1).

2.1.2 Overview of NavLab usage

NavLab has been extensively used by numerous different users since 1999, including several international research groups and commercial mapping companies. The flexible structure of NavLab makes it useful for a wide range of applications. Some users use only simulated data, whereas others use the Estimator alone to process real data. Finally, there are many cases where both simulations and real data processing are of interest. A summary of current NavLab usage is given below:

Navigation system research and development (using simulations and real data)

- Development, testing and comparison of new navigation concepts and algorithms, including new aiding sensors and aiding techniques.
- Development of real-time navigation systems where the algorithms are tested in NavLab, and then ported to the real-time navigation system. A typical development process is:
    - Test in simulations (NavLab)
    - Test with real data (NavLab)
    - Implementation in the real-time system (using a real-time operating system and C++ or similar program language)
    - Test of real-time system

Analysis of a given navigation system (using simulations and real data)

- Analysis of navigation system behavior under different maneuvers/trajectories and sensor configurations.
- Robustness analysis. The performance of the Estimator is studied for the cases of:
    - Wrong sensor models used in the Kalman filter
    - Sensor dropouts
    - Sensor errors
    - Etc

Teaching navigation theory (using simulations)

By specifying appropriate simulations, everything from basic principles to complex mechanisms can be demonstrated and visualized.

Decision basis for navigation sensor selection/purchase (using simulations)

Simulations of the relevant scenarios are carried out to investigate how varying quality of the different sensors will affect the obtainable navigation performance. Typically it is established which sensor that has the critical accuracy. In addition, parameters for different sensors available in the market are usually entered for comparison.

Decision basis for mission planning (using simulations)

Even if the set of sensors is given, the navigation accuracy can vary significantly during a mission. These variations are determined by specifications in the mission plan:

- Activating/deactivating sensors or changing their measurement rate (reasons to deactivate might be to stay covert, avoid interfering with other systems or simply to save power)
- Going to areas where certain measurements are available or are more accurate (e.g. go close to bottom to get DVL bottom track, go close to a transponder or go to surface to get GPS)
- Doing maneuvers to increase the observability of the Estimator
- Going in patterns to cancel out error growth

When setting up complex mission plans, simulations are crucial to ensure effective plans that also meet the navigation accuracy requirements for all parts of the mission (transient phase, mapping phase etc).

Post-processing of real navigation data (using real data)

Post-processing of real data will improve both the navigation performance and the integrity. See 2.1.3 for more details.

Tuning of real-time and post-processing navigation systems (using real data)

The Kalman filter tuning is essential for the estimation accuracy. The tuning might be based on the sensor specifications, but the actual sensor performance often differs from these numbers, and in such cases the tuning should be based on empirical data. For this purpose, the error estimates from the smoothing algorithm are preferred as basis.

Improving sensor calibration (using real data)

Sometimes systematic errors are present in a sensor, typically due to imperfect calibration or misaligned mounting. Such (deterministic) errors should be removed before sending the measurements to the Estimator, otherwise the performance will be reduced (in particular for the real-time Kalman filter). To find these systematic errors, the smoothing algorithm can be used, as it is significantly better than the real-time filter at estimating such errors. When the errors are known they can be compensated for in future missions.

2.1.3 Using NavLab for real data post-processing

For vehicles storing their navigation sensor measurements during missions, it is possible to make post-processed estimates of position, velocity and attitude. There are many situations where these estimates are of great interest after the mission is finished, for instance this is the case if the vehicle has recorded data that should be positioned accurately afterwards (e.g. map production or georeferencing of mine like objects). Since the Estimator works equally well with simulated and real measurements, NavLab is well suited and extensively used to produce optimal post-processed navigation results. These results are valuable also when the vehicle has calculated and stored real-time navigation estimates. Post-processed estimates are in general preferred to the real-time estimation results, since both the estimation accuracy and the integrity are improved:

- The increased accuracy is mainly due to the use of smoothing, which is an optimal estimation technique that utilizes both past and future measurements. In addition, real-time problems like delayed measurements and incomplete data sets from remote sensors are eliminated.
- The improved integrity is partly due to the smoothing algorithm, which in general is more robust against degraded sensor performance than the real-time Kalman filter. In addition, the possibility to rerun the data increases the ability to recover a faulty data set. To do this, one can modify both the degraded sensor measurements and the filter tuning to get the best navigation possible.

2.1.4 NavLab program modules

In addition to the Simulator and Estimator, a preprocessing tool (Preproc) is used to handle real measurements (by removing wildpoints, converting measurements to the correct format etc), and an export tool creates files for export (which contain the estimated position etc). Figure 2.2 shows the NavLab program modules. Different modules are used in different cases. Typical examples are:

- *Simulations*: Simulator $\rightarrow$ Estimator
- *Processing of real data*: Preproc $\rightarrow$ Estimator $\rightarrow$ Export

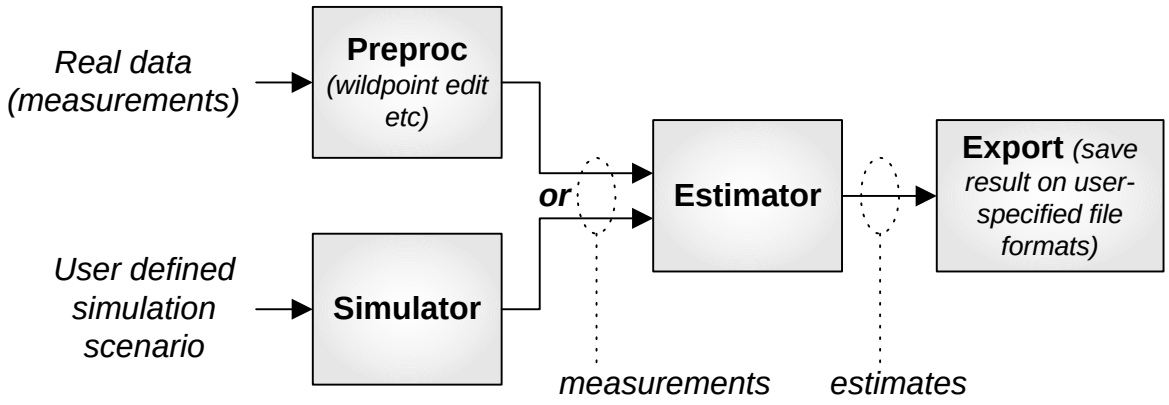

Figure 2.2 NavLab program modules

2.2 Trajectory Simulator

The following coordinate systems[1] are simulated:

- *I* (Inertial)
- *E* (Earth)
- *L* (Local)
- *B* (Body)

All relevant forces, accelerations, (angular) velocities, positions, orientations etc are returned.

Features:

- Any trajectory in the vicinity of Earth can be simulated (with unlimited complexity).
- All positions on Earth can be simulated with no singularities.
- All vehicle attitudes can be simulated with no singularities.
- Includes all Coriolis and centripetal effects due to own movement, rotating Earth, Earth curvature etc.
- Includes elliptic Earth model and gravity model.

2.3 Sensor simulators

The most important error types are included:

- White-noise
- Colored noise
- Scale factor error
- Misalignment error (to be implemented)
- Random constant error (to be implemented)

[1] A more detailed description of the different coordinate systems is found in Table A.1.

The magnitude, time-constants etc that describe the different errors (sensor parameters) are user selectable. They can be given fixed values or vary as a function of time.

The user can specify any time-vectors to simulate the individual sensors (including variable time-step lengths). Thus any sensor measurement may be present at any time.

2.4 Navigation Equations

The navigation equations are incremented each time new IMU measurements becomes available.

Features:

- Singular free for all positions and attitudes
- Foucault wander azimuth
- Direction cosine matrix attitude update
- Necessary coning and sculling compensation
- Numeric drift control
- Elliptic Earth model and gravity model
- Trapezoid updates to prevent systematic errors from forward or backward Euler

2.5 Estimator

The error state Kalman filter is an optimal estimator (given certain assumptions). The structure of the Kalman filter is shown in Figure 2.3.

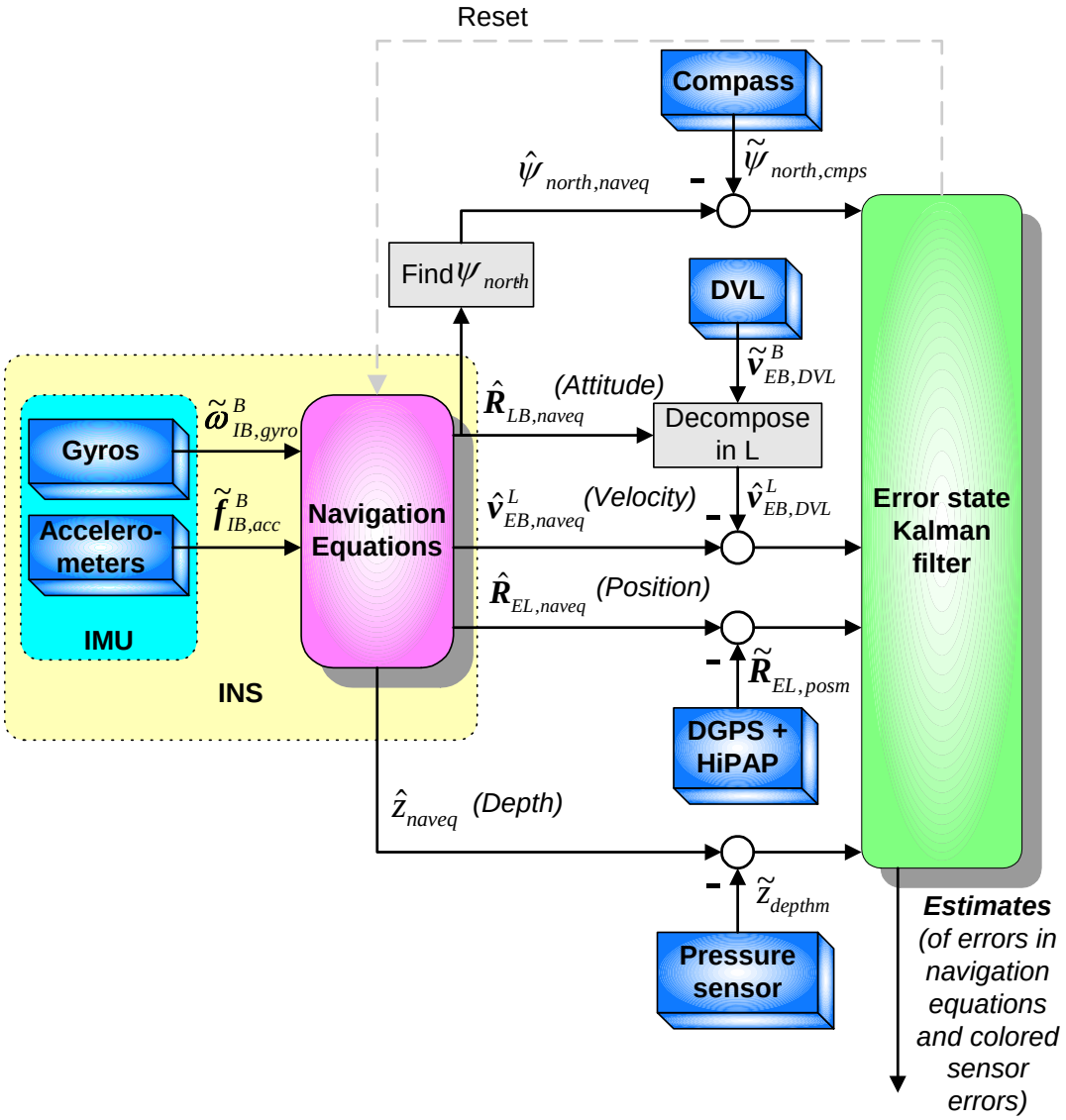

Figure 2.3 Kalman Filter structure

Features:

- Includes optimal smoothing (Rauch-Tung-Striebel implementation)
- The Estimator accepts arbitrary measurement series (time-vectors) from all sensors.
- Along with each single sensor measurement, new sensor parameters can be specified, describing that particular measurement.
- Zero velocity update (ZUPT), and depth/height measurement are included in the same Kalman filter in an optimal manner.
- Nonsingular horizontal position measurements
- Nonlinear
- Asynchronous
- Semi-continuous
- Variable process and measurement noise

2.6 User interface

In (Simulator):

Trajectory simulator: The user specifies initial position, velocity and attitude, and all changes in velocity and attitude during the simulation.

Sensor simulators: The user specifies the error behavior in each sensor to be included and the time intervals the sensor measurements are available at given rates.

In (Estimator):

The Estimator uses measurement vectors as input (either simulated or real). The user specifies the initial estimate, and sensor models used by the Kalman filter (sensor parameters).

Out (common):

After a simulation or estimation, data such as true values, measured values, estimated values, estimation uncertainties etc is available.

All the available data (more than 450 possible graphs) is displayed with a general multi-menu based plot function, see Figure 2.4.

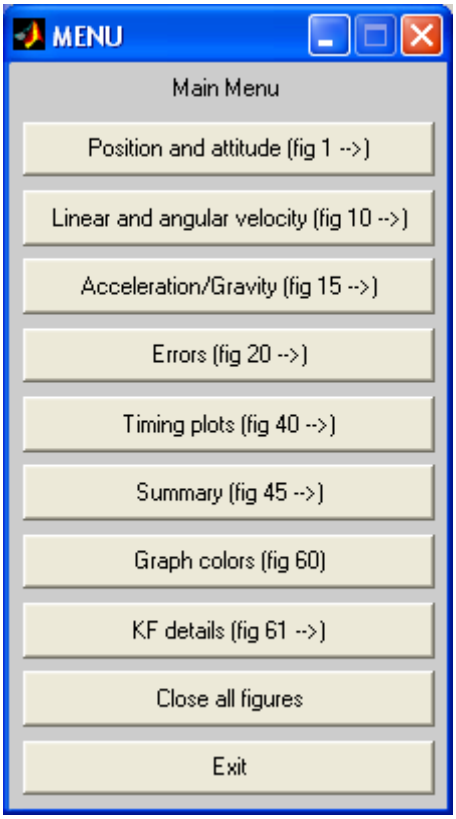

Figure 2.4 The main menu for plotting data

3 USING NAVLAB

To take full advantage of NavLab, a background in inertial navigation and Kalman filtering is beneficial. However, knowledge of the background theory is not required for users that are only using NavLab for real data post-processing.

3.1 System requirements and installation

3.1.1 System requirements

NavLab is written to run under Matlab[1] (no toolboxes are required), hence the system requirements are the same as for Matlab. Check The MathWorks homepage, www.mathworks.com, to view the system requirements for the relevant version of Matlab.

However, when working with real data or long simulations, a fast computer with plenty of memory is advantageous.

Example: When processing real data from a typical AUV-run, a 1.4 GHz Pentium 4, with 512 MB RAM gives a *real-time factor* of about 5, i.e. making estimates from 5 hours of data takes 1 hour. (If also smoothed results are needed, another 30 minutes are required.)

Multi-processor computers can be utilized by starting several Matlab processes, each processing one part of the entire run with NavLab.

3.1.2 Installation

NavLab does not need installation. The NavLab files are saved on any location, and then made visible for Matlab. See 3.4 for details.

3.2 Different versions of NavLab

NavLab is written as standard Matlab m-files (text files with names *.m). These m-files can be precompiled to corresponding *.p-files (binary). The *.p-files behave exactly the same under Matlab, but may run slightly faster (and the source code is not visible).

By default, NavLab is delivered as p-files. Usually, only scientific groups needing details about the algorithms and source code will need the m-files.

The different modules may be used separately, and thus NavLab can be delivered with only those modules needed by the user.

[1] It is tested to run under Matlab version 5.3, 6.1 or 6.5 under Windows or Unix.

Typical sub-packages of NavLab for different users:

1. Users processing real data only: **Preproc**, **Estimator** and **Export** as *.**p**-files

2. Scientific groups analyzing theoretical navigation performance for a vehicle: **Simulator** and **Estimator** as *.**p**-files. Scientific groups participating in collaboration programs with FFI may get *.**m**-files.

Note: When referring to the name of a specific m- or p-file in the user guide, the .m ending is used (representing any of the two).

3.3 File structure

3.3.1 NavLab

The NavLab package consists of m- or p-files in a 2-level directory structure. Common files like general mathematics and plotting are located in the top directory (NavLab directory). The independent Simulator, Estimator, Preproc and Export modules (shown in Figure 2.2) are located in separate subdirectories. The m/p-files in each of these subdirectories are only used by that particular module. Figure 3.1 shows the file structure of NavLab.

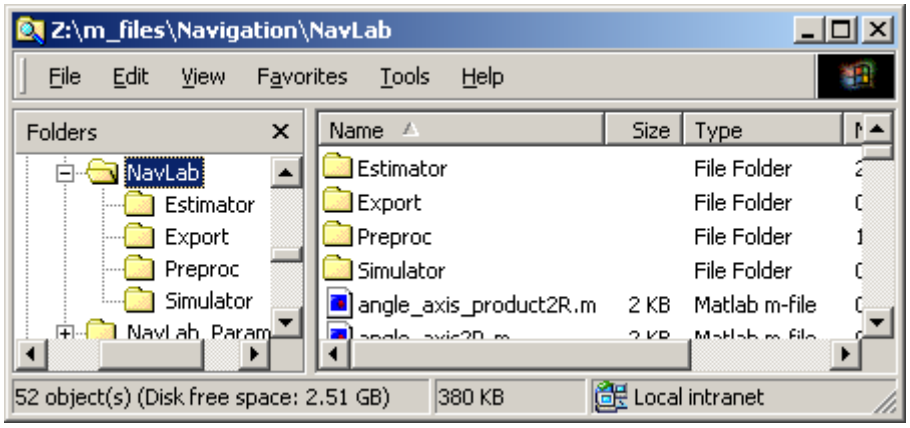

Figure 3.1 NavLab file structure (m-files)

The NavLab files are not edited by the user, and the NavLab directory can be made Read-only.

3.3.2 Working directories

When working with NavLab the user should make *working directories*, one for each simulation and/or estimation. Each directory contains all relevant information for that particular simulation/estimation.

To specify the user selectable parameters in a simulation/estimation, ini-files (text files) are used. For instance `simulator.ini` specifies simulation duration, which sensors to simulate etc. `estimator.ini` specifies Kalman filter tuning, if smoothing should be included etc. Appendix E contains examples of ini-files.

In each working directory, a copy of each relevant ini-file (with the selected values) is stored.

In this manner, a set of parameter files is sufficient to recreate exactly the same simulation/estimation (even the random noise created). Under the working directory, there should also be a subdirectory called `data` to store all data used in that simulation/estimation.

In general, files that are read/edited by the user are located in the working directory, and all the data-files (automatically generated and read) are "hidden" in the `data` directory. Figure 3.2 shows an example of a working directory (when using real data).

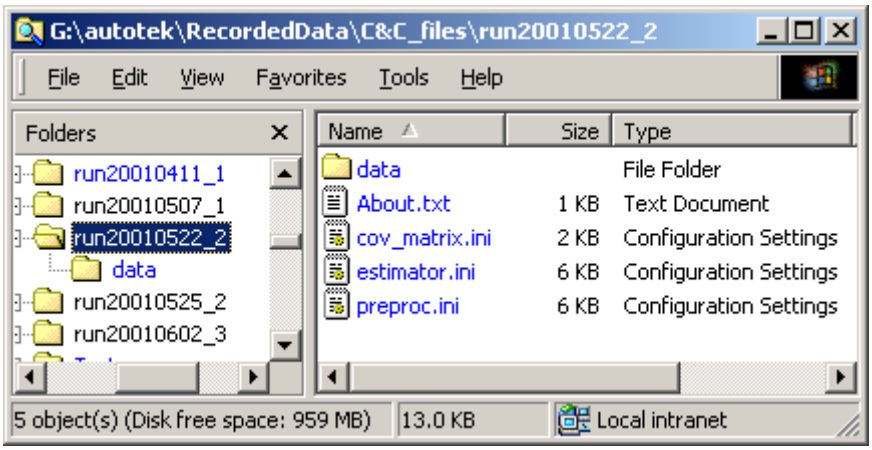

Figure 3.2 Example of working directory. (The About.txt-file is an optional file containing the user's own comments/description of this particular run.)

Note that the working directories are usually located somewhere else than the static (Read-only) NavLab directory.

3.4 Detailed user procedure

Note: Users that are only using NavLab for real data post-processing may now jump to Appendix C, which contains a short user guide only for that purpose.

To start using NavLab, you should have received the following:

- NavLab (several m or p-files in a directory structure)
- Example parameter files (examples of `simulator.ini`, `estimator.ini` etc)

What to do:

1. Save the copy of NavLab at a suitable location, and make it visible for Matlab (add the paths to the front of Matlab's path).

2. Create the first working directory, and store a copy of the relevant ini-files there. Also make a subdirectory called `data` for all data storage relevant to that particular simulation/estimation.

3. Change current directory in Matlab to the working directory (but not the data directory).

4. Edit the relevant ini-files (see the next chapters for more details).

5. Run either of these: `simulator.m` / `estimator.m` / `preproc.m`

6. After simulation/estimation is finished, the result is present in memory (and also written to files if that was selected). To view the result now residing in memory, `plot_general.m` is used. This displays a multi-level menu from which you can select different graphs to view. `Plot_general` is automatically called at the end of both Simulator and Estimator.

Repeat steps 2-6 for each new simulation/estimation.

3.5 Simulator details

For the simulation, the parameters are held in seven files. A copy of each file should be placed in the current working directory:

1. `simulator.ini`: Contains general simulator parameters

2. `IMU_sim.ini`: IMU parameters, used by `IMU_sim.m`

3. `posm_sim.ini`: Position measurement parameters, used by `posm_sim.m`

4. `depthm_sim.ini`: Depth measurement parameters, used by `depthm_sim.m`

5. `DVL_sim.ini`: DVL parameters, used by `DVL_sim.m`

6. `cmps_sim.ini`: Compass parameters, used by `cmps_sim.m`

7. `Trajrate.m`: This is an m-file describing the changes in the trajectory, used by `Traj_sim.m` during simulation.

Examples of these files are listed in Appendix E.1.

Simulator user guide:

1. Change the current directory in Matlab to the working directory (but not the `data` directory).

2. Set the correct values in the ini-files.

3. Run `simulator.m`.

The Simulator generates measurement-files (located in the `data` subdirectory). These files will later be found and used by the Estimator. Files containing the true trajectory are also generated. These will be loaded by `plot_general` when the estimation is finished.

3.6 Estimator details

For the estimation, the parameters are located in two files. A copy of each file should be located in the current working directory:

1. `estimator.ini`: Contains general Estimator parameters, initial position, attitude and velocity, and Kalman filter tuning.

2. `cov_matrix.ini`: Contains the parameters used to create the Kalman filter initial covariance matrix.

Examples of these files are listed in Appendix E.2.

Estimator user guide:

1. **Change the current directory** in Matlab to the working directory (but not the `data` directory).

2. **Open `estimator.ini`** and set the initial position, attitude and velocity (if using real data, see Estimator user guide in Appendix C.2 for details on how to find this). Set the different sensor parameters to be used in the Kalman filter and the general Estimator info.

3. Normally (if the initialization was not totally perfect) you want to **specify an initial covariance matrix** to use in the Kalman filter. This matrix describes the uncertainty in the initial estimate that was specified in `estimator.ini`. The initial covariance matrix is stored on a file (`initial_P_KF_u.txt`), and this file can be created by the following:

    a. ***Make sure* `estimator.ini` *has the correct values and is saved before continuing with this.*** Open `cov_matrix.ini` and set the uncertainty in the initial position, attitude and velocity. Save this file, and run `make_and_save_cov_matrix.m`. A file called `initial_P_KF_u.txt` is created in the working-directory. This file is used by the Estimator.

    Note that running `make_and_save_cov_matrix.m` might be automated, rather than doing it manually as described above. This is done by selecting `auto_create_cov_matrix = 1` in `estimator.ini`, and the Estimator will then run it before starting the estimation.

4. Run `estimator.m`

5. **Recommended: Save the Matlab workspace** for later use. Just exit the menu and type `save` (a file `matlab.mat` will be saved in the working directory[1]). To get the menu back, run `plot_general`. The workspace can also be saved after finishing the plotting, but it will then be larger, due to all the new variables calculated by the plotting routine. The advantage of a larger file is that the next time the plotting will be faster, as the variables are already calculated.

3.7 Preproc and Export details

See special version of the user guide (when using real data) in Appendix C.

4 UNDERSTANDING NAVLAB

It is not required to read this chapter to be able to use NavLab, but it may give relevant information and a more thorough understanding of NavLab.

4.1 Information about user input

In general the different parameters in the ini-files are (briefly) explained in the files themselves. Additional information is included in this chapter.

4.1.1 Sensor error parameters

A sensor error may consist of several components, and different parameters are used to describe each part. The relevant parameters are highlighted underlined and bold in the following explanations.

4.1.1.1 White-noise

This error is uncorrelated from one measurement to the next, and is described by its **standard deviation**.

In the Simulator it is simulated as Gaussian white-noise. For gyros and accelerometers, a parameter describing *continuous* white-noise is used. This parameter is called power density, and has a different unit than the measurement. The reason for using this parameter, and an explanation of the unit is found in section 3.2.2 and 4.2.3 in (1). A general discussion about continuous white-noise is also found on page 42 in (2).

[1] This file is simply loaded later (when the memory is cleared) by typing load.

4.1.1.2 Colored noise (bias)

A colored error (also called bias) is modeled as a first order Markov process:

$$\dot{x} = -\frac{1}{T}x + \gamma \tag{4.1}$$

where x is the colored error, T is the time-constant, and γ is white-noise. Such a process is described by its **standard deviation** (of x) and its **time-constant**. The standard deviation will describe the magnitude of the colored noise, and the time-constant describes how fast it changes. In the Simulator, the white-noise that is driving the bias is Gaussian.

4.1.1.3 Scale factor error

This error depends on the value to be measured, and the error is a constant times the measurement. This constant (or scale factor error) may not change for one specific sensor unit, but can be viewed as a stochastic variable, varying from unit to unit. Thus the parameter describing its magnitude is its **standard deviation**. The Simulator draws the scale factor errors used in one simulation using a Gaussian distribution.

4.1.2 Changing sensor availability

Note that the Estimator uses all sensor measurements located under `\data`. If you have simulated with a sensor available, and later turn that sensor off, still using the same working directory (by setting 0 in `simulator.ini`), the Simulator will not delete your old txt files for that sensor. Thus the files will be found by the Estimator and the sensor measurement will be used. The solution is to rename or delete the relevant time-vector- and/or measurement-file (located under `\data`). Another solution is of course to use a new working directory for the new simulation.

4.1.3 Trajectory simulation

During a simulation the function `Trajrate.m` (located in the working directory) defines the changes in the trajectory, see Appendix E.1.7. for an example of this file. Inside this function the user can specify any mathematical function producing $\boldsymbol{\omega}_{LB}^{B}$ and $\dot{\boldsymbol{v}}_{EB}^{B}$ as function of the current time, `t` (and possibly time-step length, `h`).

For instance, the following line will make the vehicle oscillate in pitch, with a maximum rate of 7 deg/s, and with a frequency of 0.17 Hz:

```
omg_LB_B_a_deg = [0 , 7*sin(0.17*2*pi*t) , 0 ]'; % oscillate in pitch
```

Standard maneuvers are typically specified by an interval and a rate. The following line will increase the forward speed with 2.1 m/s in the interval 100 to 110 seconds:

```
if t >= 100  & t < 110,  v_EB_B_d_a = [0.21,0,0]';  end; % accelerate
```

Note that if no changes are specified, $\dot{\boldsymbol{v}}_{EB}^{B}$ and $\boldsymbol{\omega}_{LB}^{B}$ are by default zero. $\dot{\boldsymbol{v}}_{EB}^{B} = 0$ means that the velocity-vector of the vehicle is constant seen from the *B*-system. $\boldsymbol{\omega}_{LB}^{B} = 0$ means that the attitude follows the *L*-system (defined in Table A.1). Thus, with an initial horizontal velocity, the vehicle will travel around the Earth at constant depth/height, crossing equator at the same angle each time (or traveling along it at latitude exactly zero).

4.2 Information about the NavLab output

This part contains explanations and comments on the graphs, and other outputs from NavLab. The format of exported files is described in D.1.

4.2.1 Summary

If the standard deviation of a bias estimate is more than 3 times the bias modeled in the Kalman filter, this number is shown in **bold red** text. Typically this means that the corresponding sensor has an error that is significantly larger than what has been modeled.

4.2.2 Details about the “std”

The number called “std” in the legends of the graphs in the error plots is based on the data set shown in the corresponding graph and has the same unit. The number is an estimate of the standard deviation of the stochastic process shown. Note that this estimate is not found using the function “std” in Matlab (this would give a poor estimate for instance if we have a small part of a colored stochastic process with a sample mean different from zero). RMS (Root Mean Square) is used, and this is the best estimate of standard deviation based on a sample, see Appendix B for more details.

The std-numbers in the summary are the same as those found in the legends.

4.2.3 Timing plot

For the real-time Kalman filter, an x is plotted for each time-step it was run. In addition:

- A circle is plotted around those time-steps the Kalman filter propagated without any measurements (due to a too long period without measurements).
- A plus sign is plotted on those time-steps the Kalman filter had more than one measurement.

4.2.4 Verifying the accuracy of the Kalman filter in simulations

In general, the graphs showing true estimation error and Kalman filter standard deviation can be used to verify the accuracy of the Kalman filter. The true estimation error should be within the theoretical 3-sigma value from the Kalman filter. If the true estimation error seems too large, the Kalman filter is typically too optimistic, and probably the modeled sensor errors are too small.

4.2.4.1 Exception for bias estimates (detailed)

The method described above is a quick way to verify the Kalman filter performance, and it is acceptable in most cases. However, for the bias estimates there may be cases where the estimation error seems too large, although it really is not.

The simulated sensor error typically consists of other components in addition to the bias, usually white-noise, and sometimes also other errors like scale factor error and misalignment error. The Estimator only “sees” the total error, and in the current Kalman filter the total is modeled as a bias plus white-noise. Thus the bias estimate from the Kalman filter is really an estimate of the total colored sensor error. If this error is observable and a significant part of it originates from another source than the simulated bias, there will be a difference between the simulated bias and the estimated bias, corresponding to the other component. The plotted estimation error, calculated as the bias estimate minus the true simulated bias will then seem too large. To see the real estimation error, one should compare the total-error graph with the bias estimate.

Example where this may occur:

The Simulated error in the *z*-accelerometer consists of white-noise, bias and scale factor error. In the Kalman filter it is only modeled as white-noise and bias (the bias + scale factor is modeled as a bias with a larger magnitude). The filter will successfully find the sum (scale factor + bias) and make a good estimate of it. However when plotting true bias minus estimated bias the difference will correspond to the scale factor, and this may be significant in the *z*-direction due to the g-vector.

5 ADVANCED USAGE

5.1 Changing the rate of a sensor

In the Simulator a fixed measurement rate for each sensor can be specified. When using this, the sensor will be available in the entire simulation interval with a fixed rate. However, it is also possible to use any variable rate, i.e. define the time each single measurement is available. This can be used to simulate sensor dropouts and varying sensor rate.

To simulate a varying rate for a specific sensor, set *sensor*`_tv_from_file = 1`, in the *sensor*`_sim.ini` file (tv is time-vector). When this is set, the Simulator will look (in the working directory) for a file called *sensor*`_tv_desired.txt` containing all the time-steps the sensor should be available.

Note: The phrase *sensor* (in *italic*) represents any of the abbreviations for the sensors (IMU, posm, depthm, DVL or cmps).

5.1.1 How the file is used

When the Simulator starts, it is always going through all the time-steps from start to stop-time with the rate given in `simulator.ini`. The sensor(s) with varying rate is simulated at the simulator time-steps nearest the desired time-steps specified in *`sensor`*`_tv_desired.txt`. Time-steps outside the simulator interval are ignored.

5.1.2 How the file is made

The file *`sensor`*`_tv_desired.txt` can be made manually or by the utility `make_and_save_sensor_tv.m`. In this script the user can specify an interval by a start and stop time, and the desired sensor rate within this interval. Any number of intervals can be specified.

Note: This script is located under the NavLab/Simulator directory. It is recommended to make a local copy of this m-file in the working directory, and then edit the local copy.

5.1.3 Changing the sensor rate in the Estimator

Changing the sensor rate is automatically taken care of by the Estimator: It always uses the sensor measurements at the time-steps they are available.

5.2 Changing the quality of a sensor

The parameters describing the sensor errors are, by default, constant. However it is possible to change these as a function of time (e.g. to describe degradation or improvement in a sensor).

Note: In the summary (from `plot_general`) the constant parameters are given as numbers, and varying parameters are plotted as graphs.

5.2.1 Changing quality in the Simulator

In the Simulator the constant parameters are given in the *`sensor`*`_sim.ini` files. But if there exists a file (in the working directory) called *`sensor`*`_sim_quality_intervals.txt` changing quality specified in this file is used in stead.

The file contains start and stop times for an interval, and the parameter values valid for that interval. Any number of intervals can be specified. Outside of the intervals specified, the fixed values from *`sensor`*`_sim.ini` are used.

Note that not all parameters must be specified, only those changing. For an *n*D measurement (DVL is 3D, posm is 2D) the first *n* values after the interval will be interpreted as the standard deviation of the measurement noise. The next *n*, if specified, will be interpreted as the standard deviation of the bias. If the last *n* are also specified, they will be interpreted as the bias time-constants.

Parameters that are not specified, will have the fixed values specified in *`sensor`*`_sim.ini`.

Example:

Assume `DVL_sim_quality_intervals.txt` looks like this:

```
 50  70  0.1 0.1 0.1  0.05 0.05 0.05  600 600 600
100 130  0.3 0.3 0.3  0.05 0.05 0.05   60  60  60
```

This means that in the interval 50 to 70 seconds, the standard deviation of the white-noise in DVL *x*, *y* and *z* direction is 0.1 m/s (the units are the same as used in `DVL_sim.ini`). In the same interval the standard deviations of the 3 biases are 0.05 m/s with time-constants of 600 seconds.

Between 70 and 100 seconds, the values from `DVL_sim.ini` are used.

In the interval 100 to 130 seconds, the white-noise is tripled, and the bias is changing ten times faster.

The file can be made manually or by the script `make_and_save_sensor_sim_quality.m`. The script also contains more detailed help.

Note: The script is located under the NavLab/Simulator directory. It is recommended to make a local copy of this m-file in the working directory, and then edit the local copy.

5.2.2 Changing quality in the Estimator

In the Estimator, the constant models of the sensors used are specified in `estimator.ini`. However, if a file called *sensor*`_est_quality.txt` exists in the working directory, the parameters specified in this file will be used (ignoring the corresponding parameter values in `estimator.ini`). This file has the same number of rows as the file containing the measurement itself, and each row describes the quality of the corresponding measurement. As for the Simulator, one or more parameters may be specified (parameters not specified are taken from `estimator.ini`).

Example:

Assume `DVL_est_quality.txt` looks like this:

```
0.1 0.1 0.1  0.05 0.05 0.05  600 600 600
0.1 0.1 0.1  0.05 0.05 0.05  600 600 600
0.1 0.1 0.1  0.05 0.05 0.05  600 600 600
0.3 0.3 0.3  0.05 0.05 0.05   60  60  60
0.3 0.3 0.3  0.05 0.05 0.05   60  60  60
```

This means that there exist 5 measurements from the DVL, and the first 3 have white-noise 0.1 m/s in *x*, *y* and *z* and long time-constants. The biases are the same for all 5 measurements.

The file can be made by `make_and_save_sensor_est_quality.m`, but since it is closely related to the measurement, this file is usually made by the measurement source (details are given below).

5.2.2.1 Data from Simulator

When changing sensor quality has been simulated, a file called *sensor*_sim_quality.txt is generated (in the working directory). This file has the same format as *sensor*_est_quality.txt (contains one row with parameters describing each measurement). Thus making a copy of this file with the filename changed from "sim" to "est" will give the Estimator the correct changing parameter values. Obviously, this new file can also be edited, to let the Estimator use a different model than what was simulated.

5.2.2.2 Data from preproc (real data)

Preproc automatically makes *sensor*_est_quality.txt – files for sensors with varying quality. The quality parameters are based on the sensors own quality numbers, knowledge of the sensor behavior etc.

5.3 Zero velocity update (ZUPT)

Zero velocity condition is valid when the vehicle has no movement relative to the Earth. This information can be used by the Estimator as measurements, and will improve the estimates. To tell the Estimator that the condition is valid (and that it can do ZUPT) a file called ZUPT_intervals.txt, should exist in the working directory. This file simply contains the start and stop time for the intervals where ZUPT is valid.

The file has this format:

```
start_time_interval_1  stop_time_interval_1
start_time_interval_2  stop_time_interval_2
start_time_interval_3  stop_time_interval_3
```

The file can be made manually, or by the script make_and_save_ZUPT_intervals.m.

Note that the script is located under the NavLab/Estimator directory. It is recommended to make a local copy of this m-file in the working directory, and then edit the local copy.

Soon the ZUPT functionality in NavLab will be further improved with more options (for instance the possibility to specify the accuracy of the ZUPT).

5.4 Programmable menu selections

After a simulation or estimation, the result is shown by calling plot_general, which displays the main menu (see Figure 2.4). From the menu/submenus the user can select any figure to be displayed.

Alternatively plot_general may automatically plot only one specific figure, not showing the menu. This is done by first creating the vector menu_preselection and then calling plot_general. The menu_preselection-vector contains the sequence of user selections that would normally be necessary to display a figure. The selections are expressed as button numbers (counting from the top) in each menu.

Example:

```
menu_preselection=[6 4 5]
plot_general

menu_preselection=[7]
plot_general
```

The above code will automatically select button 6 (Summary) on the main menu, then it will select button 4 (Real Data Quality Control) on the Summary-submenu, and finally button 5 (DVL) on the QC-submenu. A second call to plot_general is done selecting button 7, which shows the figure with graph colors.

The programmable menus might be useful when the Simulator/Estimator is part of an automated process, and specific figures should be shown automatically. In such cases the variables `skip_simulation_plot/skip_estimation_plot` in `simuator.ini/estimator.ini` can be set to 1, and only the automatically selected figures are shown.

Programmable menus also make it possible for the user to program his/her own customized menu, change figure properties, titles etc.

Note that the default menus in plot_general might be changed in future versions of NavLab (e.g. new figures might be included). If the relevant menu-preselection numbers are affected by the change (pointing to wrong figure in the new version), they must be updated.

APPENDIX

A MATHEMATICAL NOTATION

The notation is in accordance with (1) and a summary is given here.

A.1 General mathematics

Table A.1 contains a summary of the most important coordinate systems used in NavLab (simplified description).

Symbol	Description
I	**Inertial.** (Orientation and location of the origin not relevant.)
E	**Earth.** The origin coincides with Earth's center (geometrical center of ellipsoid model), the *yz*-plane coincides with the equatorial plane, the *y*-axis points towards longitude +90° (east) and the *x*-axis points towards north.
L	**Local.** The origin is directly beneath or above the vehicle, at Earth's surface (surface of ellipsoid model). The *z*-axis is pointing down. NED-version (North East Down): The *x*-axis points towards north, and the *y*-axis towards east. Foucault-version (Wander Azimuth): The *x*- and *y*-axes are rotating about *z* such that their angular velocity relative to the Earth has zero component along the *z*-axis. *NavLab uses the Foucault version.* Initially it coincides with the NED system (the wander azimuth angle is zero). Note: If the run is within a limited geographical area, and not close to any of the poles, the *L* system in NavLab will be close to a NED system through the entire run.
B	**Body**. The origin is in the vehicle's reference point. The *x*-axis points forward, the *y*-axis to the right (starboard), and the *z*-axis down.
M	**Map**. (Used in the plotting only. Useful when studying a local trajectory in meters). The origin is Earth fixed: Vertical position: At Earth's surface. Horizontal position: At the initial vehicle-position of the run being plotted. The *x*-axis points towards North, *y* towards East and *z* down (NED). Note: *M* is equal to an Earth fixed NED-version of *L* located at the initial position.

Table A.1 Definitions of coordinate systems (simplified).

The symbol usage in NavLab and the relevant documentation is summarized in Table A.2.

Symbol	**Description:**	**Example:**
Lowercase letter with arrow	Coordinate free (or "symbolic") vector (not decomposed in any coordinate system).	$\vec{v}$ (Velocity)
(Right) subscript	Specification of the value (coordinate systems involved, and further comma-separated specifications if needed).	$\vec{v}_{EB}$ (Velocity of the *B*-system relative to *E*)
Bold lowercase letter	Vector decomposed in a coordinate system.	$\boldsymbol{v}_{EB}^{B}$ (The above velocity decomposed in *B*)
Right superscript	In which coordinate system the vector is decomposed.	
Bold uppercase letter	Matrix (decomposed in a coordinate system).	$\boldsymbol{R}_{EB}$ (Orientation of the *B*-system relative to *E*)
Left superscript	In which coordinate system time-differentiation is done.	${}^{E}\frac{d}{dt}(\vec{v}_{EB})$ (The derivative relative to *E*)

Table A.2 Symbol usage.

Table A.3 shows the notation used to describe relations between two coordinate systems.

Symbol	**Definition**	**Description**
$\vec{p}_{AB}$	Defined by the description	A vector whose length and direction is such that it goes from the origin of coordinate system *A* to the origin of coordinate system *B*.
${}^{C}\vec{v}_{AB}$	${}^{C}\frac{d}{dt}(\vec{p}_{AB})$	The velocity of the origin in coordinate system *B* relative to coordinate system *A*, observed from coordinate system *C*.
${}^{C}\vec{a}_{AB}$	${}^{C}\frac{d}{dt}({}^{C}\vec{v}_{AB})$	The acceleration of the origin in coordinate system *B* relative to coordinate system *A*, observed from coordinate system *C*.
$\vec{\omega}_{AB}$	Not included	The angular velocity of coordinate system *B*, relative to coordinate system *A*.

Table A.3 Notation used to describe different relations between two coordinate systems.

Usually the velocity and acceleration are observed from the same system as they are relative to, thus we define:

$$\vec{v}_{AB} \triangleq {}^{A}\vec{v}_{AB} \tag{A.1}$$

and:

$$\vec{a}_{AB} \triangleq {}^{A}\vec{a}_{AB} \tag{A.2}$$

In addition *f* is used for specific force (*a* + gravitation), with the same use of sub- and superscripts as *a*.

Examples of the notation:

- $\vec{\omega}_{IE}$ Earth's angular rate relative to the inertial space
- $\boldsymbol{f}_{IB}^{B}$ The specific force sensed by three orthogonal strapdown accelerometers (if no gravitation is present this would equal $\boldsymbol{a}_{IB}^{B}$)
- $\boldsymbol{v}_{EB}^{L}$ The velocity of a vehicle relative the Earth, decomposed in the local level (typically calculated by the navigation equations)

A.2 NavLab specific notation

General symbols used in NavLab are summarized in Table A.4.

Variable	Description
`t`	Current time
`k`	Current time-step
`h`	Time since last time-step
`sav_x`	The general variable *x* has a certain value each time-step during a time interval. It is often useful to express the collection of all these values, and the prefix "sav" means that this is a vector (or matrix if *x* is a vector) that has **sav**ed all the values of *x* during the interval.
`source_tv`	A source has generated one or more variables at each time-step during a time interval. The suffix "tv" after a name of a source is used to express the **t**ime-**v**ector of all variables from the source, i.e. the times the variables from that source are valid. (Corresponds to sav_x, and has the same length. Example: IMU_tv: the time the IMU measurements are available.)
`n`	Normal vector. A special unit vector we have introduced to be able to represent horizontal position all over the Earth with no singularities. Approximate[1] description: The vector points from Earth's center to the vehicle position, but is normalized to have length 1. The exact direction is the local normal, perpendicular to the ellipsoid (corresponding to geodetic latitude). The *n*-vector decomposed in *E* equals the last column in $\boldsymbol{R}_{EL}$ with opposite sign and is easily converted to or from longitude and geodetic latitude.

Table A.4 General symbols used in NavLab.

[1] Due to the ellipticity of the Earth.

Table A.5 contains various variants of the general variable *x*.

Description	**Mathematical symbol**	**Text (Matlab) symbol**
True value (from the Simulator)	x	`x`
Measured value	$\tilde{x}$	`x_m (measured)`
Computed value (e.g. from navigation equations)	$\hat{x}$	`x_c (computed)`
Updated value (from the Kalman filter)	$\hat{x}$	`x_u (updated)`
Predicted value (from the Kalman filter)	$\bar{x}$	`x_p (predicted)`
Derivative (in time)	$\dot{x}$	`x_d (derivative/dot)`
Value valid next time-step	x_{k+1}	`x_n (next/new)`
Value valid last/previous time-step	x_{k-1}	`x_l (last)`
Average of a value in the interval t_{k-1} to t_k	$\frac{x_{k-1}+x_k}{2}$	`x_a (average)`
Total error in a computed or measured version of the variable *x*. A subscript will indicate the source of this error.	δx	`dx`
A part of δx, with its own description, for instance the bias-part of a measurement error. A subscript will indicate what kind of error this is.	Δx	`Dx`

Table A.5 Text equivalents to mathematical symbols, used when programming NavLab. The general variable x is used as an example.

Different data sources have standard names, as shown in Table A.6.

Source	**Abbreviation**
Measurement from the **gyro**s	`gyro`
Measurement from the **acc**elerometers	`acc`
Measurement from GPS or other absolute position reference (**pos**ition **m**easurement)	`posm`
Measurement from the depth meter (**depth** **m**easurement)	`depthm`
Measurement from the Doppler Velocity Log (**DVL**) (or other velocity reference)	`DVL`
Measurement from the **c**o**mp**a**s**s	`cmps`
From the **nav**igation **eq**uations (strapdown navigator)	`naveq`
From the **K**alman **f**ilter	`KF`
From the **smooth**ing	`smooth`

Table A.6 Abbreviations used for the different data-sources in NavLab.

Examples of the notation given in appendix A:

roll_naveq_c	Roll, **c**alculated by **nav**igation **eq**uations
roll_KF_u	Roll, **u**pdated estimate from the **K**alman **f**ilter
sav_v_EB_B_DVL_m	The **sav**ed variant of v_EB_B **m**easured by the DVL.
DVL_tv	The **t**ime-**v**ector containing the times the DVL-measurements (sav_v_EB_B_DVL_m) are valid
Dz_depthm_bias_smooth	Bias error in the **depth** **m**easurement, estimate from the **smooth**ing

B ESTIMATING STANDARD DEVIATION OF A STOCHASTIC PROCESS FROM A SAMPLE

The error plots show true errors, estimates of errors, and the final estimation errors, and they are all expected to be zero. The graphs shown can be viewed as samples from stochastic processes.

Let x_i be sample number i. Assuming $\mu = E[x_i]$ is known, (B.1) is an unbiased estimator of the (underlying) true standard deviation σ, $E[\hat{\sigma}] = \sigma$.

$$\hat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2} \tag{B.1}$$

Since we assume that $E[x_i] = \mu = 0$, the estimated standard deviation is given by (B.2).

$$std = \hat{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^{\,2}} = RMS \tag{B.2}$$

This turns out to be equal to the RMS[1] of the sample.

If x_i is colored, and we have knowledge about the process model, estimators that are better on short intervals may exist. However RMS will converge towards the same (true) value for longer (ergodic) samples, and is the best to use when the model is unknown. For white Gaussian noise, RMS is also the maximum likelihood estimator.

Note1: If the sample mean is zero, RMS equals the "std" function in Matlab, with the flag set.

> The flag set means that we divide by n instead of $(n – 1)$. The $(n – 1)$ is used when we have to estimate the expected value (underlying mean) in addition to the standard deviation from the same data set. The estimated mean also depends on σ, and is subtracted, and this leads to the –1 (see (3) page 240 for a simple example showing this). In our case however, we assume we know the underlying mean (zero), and thus we should divide by n.

Note2: If the data in the sample is constant ("std" in Matlab is zero), RMS equals the sample mean.

C USER GUIDE WHEN POST-PROCESSING REAL DATA

Before starting: Make sure the NavLab directories (including the Preproc, Estimator and Export subdirectories) are in front of the Matlab path. If not, use Path Browser in Matlab, and add these 4 directories in front: `NavLab`, `NavLab\Preproc, NavLab\Estimator` and `NavLab\Export` (this can be done in one operation if using "Add with subdirectories").

[1] Root Mean Square, which is the square root of the empirical second moment of the sample

Figure C.1 shows the dataflow when using real data from HUGIN.

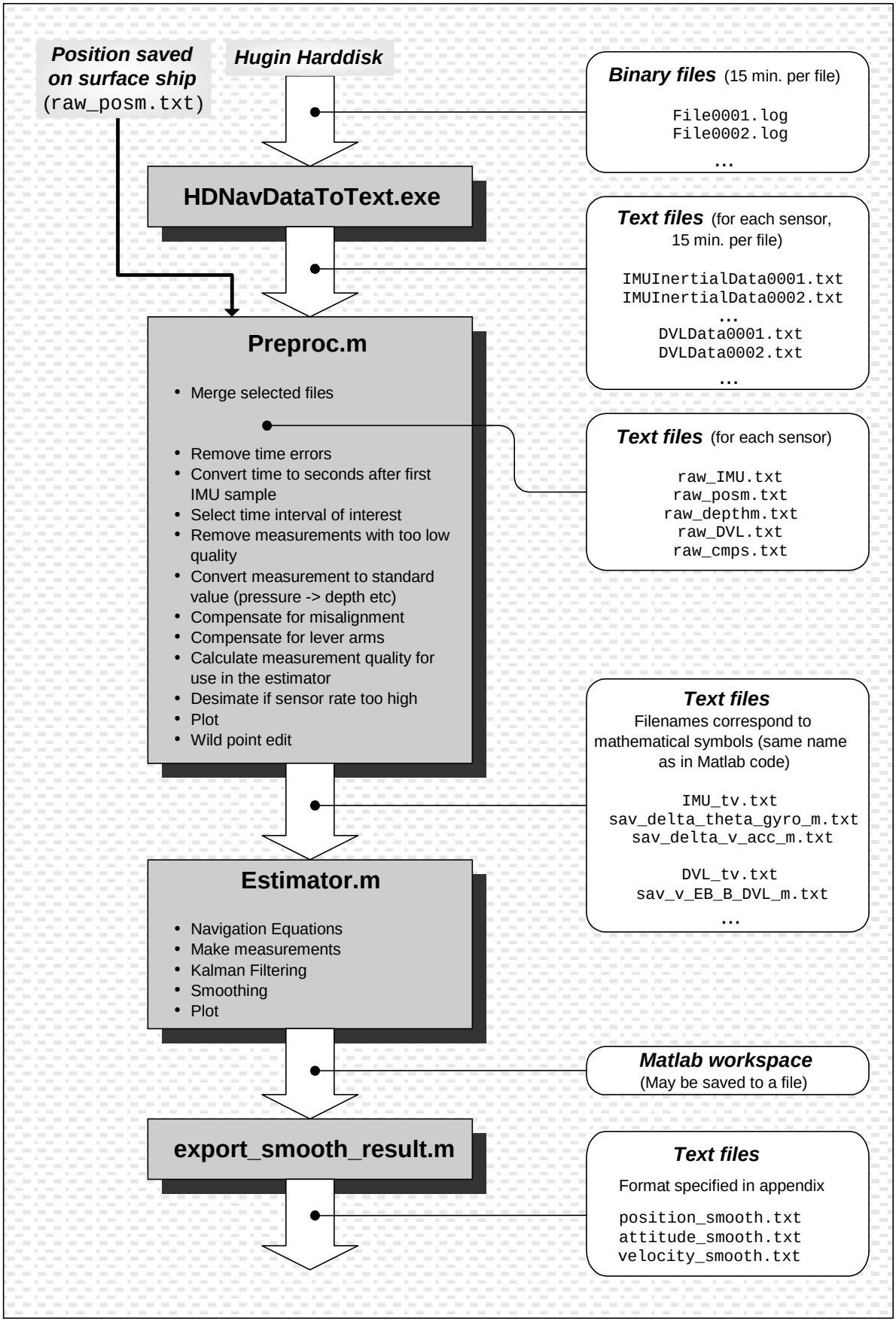

Figure C.1 Dataflow when processing real data from Hugin.

C.1 Preproc

1. First, create a **directory for the run** (*working directory*), and a subdirectory called `data`.
2. The **txt-file**s with inertial data, depth, DVL and compass are **copied into the data directory**. (Filenames like `IMUInertialData0001.txt` etc)
3. The **position file** from the surface vessel is **also copied** into this directory. (Filename, typically `pos.txt`)
4. **Rename** `pos.txt` **to** **`raw_posm.txt`** (or make a copy with this name).
5. **Copy 3 files** to the working directory: **`preproc.ini`, `estimator.ini`** and **`cov_matrix.ini`** (the two last files are for later use.)
6. Edit `preproc.ini`. If necessary: Select the interval of sensor files to load and your preferred level of automation (and make sure the lever arms and misalignment are correct).
7. **Change current directory in Matlab** to the working directory (but not the `data` directory).
8. **Run preproc.** The sensor files have names like `IMUInertialData0001.txt` etc, and there are several of each (0002, 0003 etc). Preproc is first merging these files, and renaming them to `raw_<sensor>.txt`.
9. Select time interval etc as requested in the Matlab command window (if not automated).
10. Examine all figures that are plotted on the screen and the text output in the Matlab command window. Note that these figures are updated as you select shorter intervals, remove wild-points etc.
11. If necessary, do wild-point editing for the sensors needed. Typically there are wild-points in the position measurement. A tip is to first remove those with poor HiPAP quality, and then auto-detect wild-points. For more details about the wild-point detection, see C.1.1.

Note that any dotted graphs are the sensor measurement before lever arm compensation.

After preproc has finished, new txt-files, containing time-vectors and measurements from the available sensors should be present in the `data` directory (their format is described in Appendix D.2).

Note that all text output from preproc (in the command window) is logged in: `preproc_log.txt`.

C.1.1 Position wild-point details

The wild-point detection algorithm simply evaluates the velocity from one position measurement to the next. If the velocity exceeds a user-specified limit, the next measurement

is assumed to be a wild-point. To avoid “being stuck outside the truth”, it is not allowed to reject more than a user-specified number of consecutive measurements.

A case where the algorithm has reduced performance is if there is a long time before the next measurement, and this measurement is a wild-point. If the time is long enough, the algorithm will assume that the measurement is OK. Additionally, if this wild-point is directly followed by good measurements, the first few of those might incorrectly be assumed to be wild-points.

To reduce such problems, the algorithm is also run backwards through the position measurements. The above example will be handled correctly by the algorithm running backwards.

The wild-points detected by the forward algorithm are marked with a red circle on the existing position figures. A green circle means that the measurement is accepted only because too many consecutive were rejected. The backward filter uses similar marking, but it uses squares instead of circles. Figure C.2 shows the above example, where the vehicle is heading south-east. The wild-point is not detected by the forward algorithm, and four consecutive correct measurements are assumed to be wild-points, before the fifth is accepted due to the limit. The backward algorithm correctly detects the wild-point.

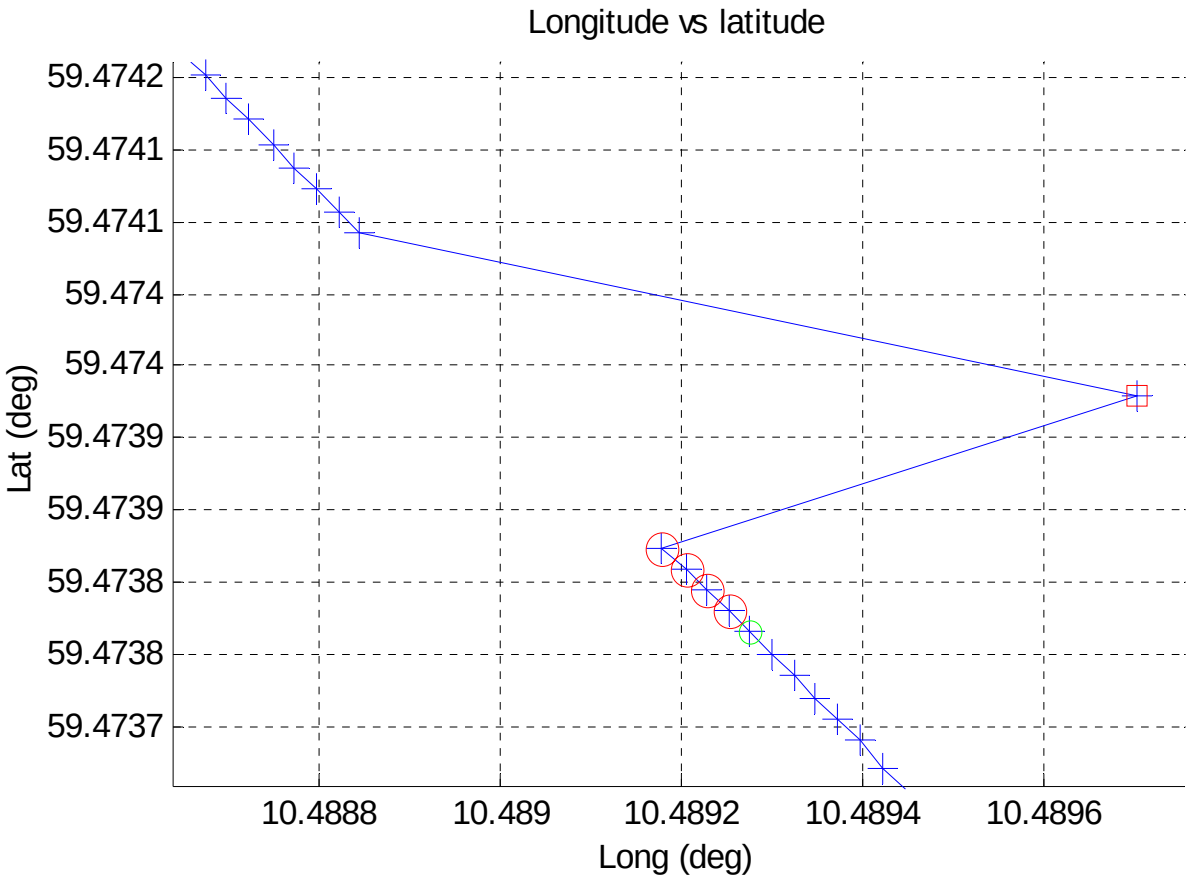

Figure C.2 Position dropout followed by a wild-point.

A wild-point status is calculated for each position measurement based on the two algorithms. The total status is the sum of the status from each of the two. The numbers used are as follows:

Measurement status:	**Forward filter**	**Backward filter**
OK	0	0
OK since too many were rejected	0.3	0.2
Wild-point	2	1.8

Consequently a total status 0 means OK according to both algorithms. 3.8 means wild-point according to both, etc (in general, a higher value means that the measurement is more unreliable).

The tuning of the detection algorithm is inside `preproc.ini`, and the parameters entered here will be read each time you press “Detect wild-points”. In this manner it is possible to tune the algorithm and watch the result until the tuning is OK.

C.2 Estimator

1. Open **`estimator.ini`** and **set the initial position, attitude and velocity** (see bullet below or explanation in C.2.2). Set the different sensor parameters to be used in the Kalman filter and the general Estimator info.

    a. Initial estimates of position, attitude and velocity can be set by using sensor measurements valid at the estimation_start_time (initialization time). To get the sensor measurements valid at the initialization time, the script `get_sensor_values_for_init.m` can be run. This script will get interpolated values for position, attitude and velocity based on the sensor measurements. If the requested initialization time is prior to the first measurement, the initialization will be less accurate (see below).

2. **Specify an initial covariance matrix** to use in the Kalman filter: *Make sure the estimator.ini has the correct values and is saved before continuing with this*. Open `cov_matrix.ini` and set the uncertainty in the initial position, attitude and velocity (if changes are necessary). Save this file, and run `make_and_save_cov_matrix`. A file called `initial_P_KF_u.txt` is created in the working-directory. This file is used by the Estimator.

    a. Tips for setting the initial uncertainty: If sensor measurements were used for initialization, the uncertainty should correspond to sensor uncertainties. For instance the uncertainty in initial position should account for AUV depth and the difference between init-time and the time of the position measurement used for init.

Note that running `make_and_save_cov_matrix.m` might be automated, by selecting `auto_create_cov_matrix = 1` in `estimator.ini`. The Estimator will then run it before starting the estimation.

3. **Run `estimator.m`.** For long runs, it is recommended to first run only about 30 to 60 seconds, to check that the initial values and standard deviations are ok. If there are several hours with data, a good solution may be to divide it into several parts, and estimate one part at a time (see C.2.3). Available memory can be increased by closing other applications and closing all Matlab figures (use: `close all`).

4. **Recommended: Save the Matlab workspace** for later use. Just exit the menu and type `save` (a file `matlab.mat` will be saved in the working directory). To get the menu back, run `plot_general`. The workspace can also be saved after finishing the plotting, but it will then be larger, due to all the new variables calculated by the plotting routine.

5. **Inspect the result.** Details about how to check that the data is OK, is found in C.2.1.

6. **Export the result.** Run `export_smooth_result.m` (or a special custom version like `save_result_to_CnC.m`). Result files (with format described in Appendix D.1) will be created in the working directory. To view these files at a later time `plot_exported_result.m` can be used.

Note: For long runs, some of the figures may take several minutes to plot. Please be patient if no response, a NavLab/Matlab crash is very unlikely.

C.2.1 How to check that the real data is OK?

Sometimes sensors degrade, timing is wrong or perhaps wrong initial values were given to the Estimator. Any of these problems will typically reduce the accuracy of the estimates. To verify that the real data and the estimation is OK, the following figures should be inspected:

Fig #	Figure contents	Figure group
1	Longitude vs Latitude	Plots of total states (including measurement and estimates)
2	Long, Lat, depth (vs time)	
3	Roll, pitch, yaw	
4	Yaw relative North (contains compass measurement)	
11	Velocity decomposed in *B* (contains DVL measurement. **Note:** *May take a long time to plot!*)	
45	General summary, including IMU	Numerical summary
46	Summary for posm, depthm, DVL and cmps	
53	Gyro error	Estimate of sensor bias errors versus error models (for Quality Control).
54	Accelerometer error	
55	Position measurement error	
56	Depth measurement error	
57	DVL error	
58	Compass error	

The figure numbers shown in bold should always be inspected. It is recommended to check all figures listed, especially if irregularities are found. What to look for depends on the figure group:

C.2.1.1 Plots of total states

Check that the measurements are not too far from the estimates. If the real-time (green) and smooth (red) estimates are far apart at the start, the initial values given in `estimator.ini` are probably wrong.

C.2.1.2 Numerical summary

Look for bold red numbers. If a number is bold and red it means that the standard deviation of this error estimate is above 3 times the model. This indicates that the sensor has a much larger error than assumed by the model. Note that for parameters that are varying (the text “varying” has replaced the number in the summary), the Quality Control menu should be used, see below.

C.2.1.3 Sensor errors

The usage of red numbers in the numerical summary is based on comparison of the total standard deviation with the model. The quality control graphs show the estimates and limits as a function of time. This makes it possible to detect sensor problems present in a small interval only, such that the total standard deviation for the entire run is still under the limit (and no numbers are red).
The thick red line shows the bias estimate, and the green lines show the Kalman filter model, ±1 and 3 sigma. The bias estimate should ideally be inside the 1-sigma limit 68% of the time, and inside the 3-sigma limit 99.7% of the time (if fully observable). If it is outside the 3-sigma limit much of the time, the sensor error is probably larger than modeled.

The figures (except gyro and accelerometer) also have a dotted red graph showing the sensor measurement minus the estimate. Thus, this graph shows the total estimated error (with both bias (colored) and white-noise). This graph is included in the plot to detect wild-points that was not removed during preproc. Figure C.3 shows a remaining position wild-point of 22 meters. To evaluate the impact the wild-point has on the estimate, the full state plot with the measurement can be used. If the smooth (red) estimate makes a jump/shift towards the wild-point that is too big, preproc should be re-run to remove the wild-point (or the wild-point could be removed by editing the measurement files directly).

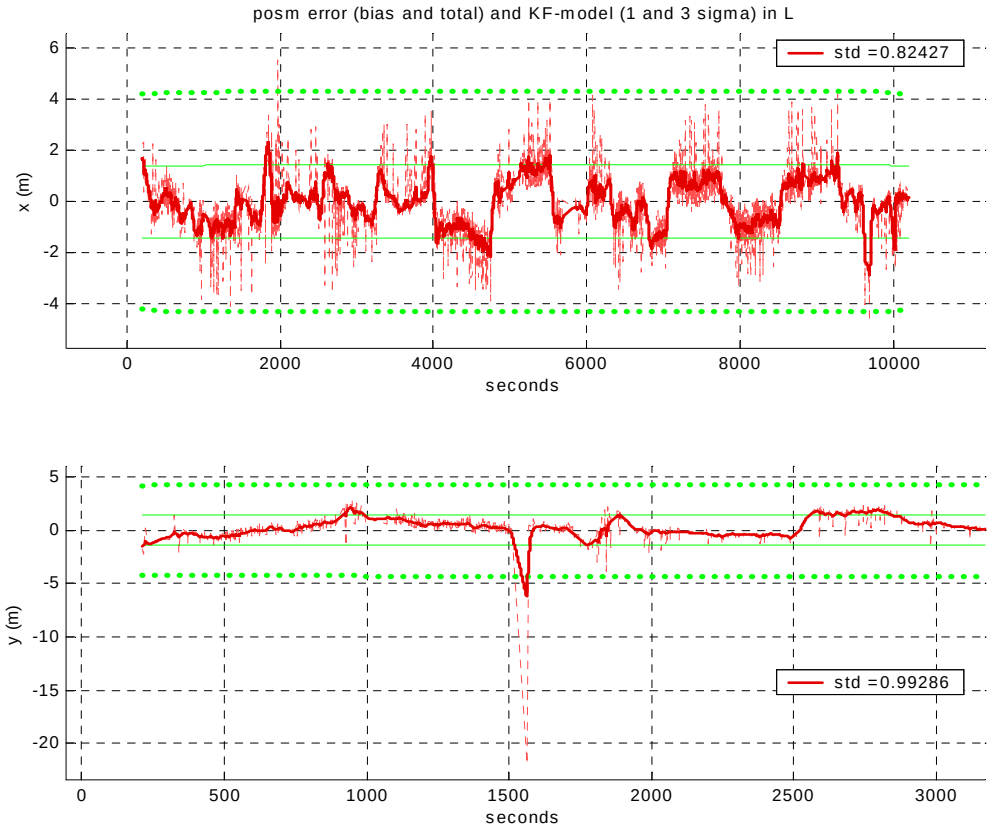

Figure C.3 Quality control of the position measurement

C.2.1.4 Checking estimation accuracy

In addition to the above-mentioned figures, it is often interesting to take a look at the Kalman filter estimation uncertainty. Figure 21, 23 and 25 in NavLab shows the accuracy of the attitude, velocity and position estimates.

Note that the accuracy shown is the theoretical number from the Kalman filter, assuming all sensors behave exactly as modeled. When using real (non-ideal) sensors, the actual estimation error is usually somewhat larger. However, if the sensor errors are modeled larger than they really are, the real estimation error might also be smaller than the theoretical standard deviation.

C.2.2 How to find the initial estimate?

An alternative to running `get_sensor_values_for_init.m` is to use the procedure described below:

Yaw: Use the compass measurement valid at the estimation_start_time, by looking at the figure from preproc.

Depth: Use the depth measurement valid at the estimation_start_time, by looking at the figure from preproc.

Velocity in B: Use the DVL measurement valid at the estimation_start_time, by looking at the figure from preproc.

Roll and pitch: Roll and pitch from a motion sensor (if available) may be used, - by looking at the graph from preproc with the attitude reference used in posm or depthm lever arm compensation. However they may not always be available. **Solution:** Run the Estimator with smoothing for about 30 seconds or more using zero (or a better guess) for both initial roll and pitch and an uncertainty of several degrees. The smoothing will detect (most of) the error and suggest a more correct initial roll and pitch. Use these values as initial values. You may have to repeat this procedure to get acceptable initial roll and pitch. Initial roll and pitch might also be found from the direction of the specific force measurements (assuming the acceleration is significantly below g).

Horizontal position: Use the posm long and lat measurement valid at the estimation_start_time, by looking at the figure from preproc if possible. Sometimes it is not possible to zoom enough on this figure to get the required accuracy. These Matlab lines will pick out the first posm measurement, and display the values needed:

```
%Get the first position measurement:
[long_posm,lat_posm]=n_E2long_lat(sav_n_E_posm_m(:,1));
format long
deg(long_posm)
deg(lat_posm)
```

If there is a long time from the estimation_start_time to the first position measurement, the first measurement will not be valid at estimation_start_time (have a too big error). **Solution:** Run the Estimator with smoothing for about 30 seconds or more using the first (incorrect) measurement. The smoothing will detect (most of) the error and suggest a more correct position valid at estimation_start_time. Use this estimate as the initial position by means of these lines (if it is not possible to zoom enough on the figure):

```
% Get the first smoothed position:
[long_smooth,lat_smooth]=n_E2long_lat(sav_n_E_smooth(:,1));
deg(long_smooth)
deg(lat_smooth)
```

You may have to repeat this procedure to get an ok initial position.

C.2.3 How much data can be processed by NavLab in one run?

NavLab itself has no limit when it comes to the endurance of the run or amount of data. However, after the run is processed, there is a lot of data in memory to be plotted. On a 1.4 GHz computer with 512 MB RAM, the plotting might get pretty slow if the run is longer than 5 - 6 hours (assuming IMU-rate is 100 Hz, Kalman filter rate is about 4 Hz and smoothing is included).

D DATA FORMATS

D.1 Standard export format.

After the estimation is finished, data is resident in memory. Running `export_smooth_result.m` will save the result to files with position, attitude and velocity in the working directory (assuming smooth estimates have been made). The format of these files are given in tables D.1 to D.3

Filename:	**`position_smooth.txt`**		
Contents:	Vehicle position relative Earth ($\boldsymbol{p}_{EB}^{E}$)		
Column 1:	**Column 2:**	**Column 3:**	**Column 4:**
time since 1970.01.01 [sec]	**geodetic latitude** [rad]	**longitude** [rad]	**depth**/height (from ellipsoid surface[a], positive down) [m]

Table D.1 Exported position file

[a] *Note: NavLab assumes that the depth measurement is relative to the ellipsoid surface, using the same ellipsoid as the latitude and longitude measurements (e.g. WGS-84). However, if the depth input is relative to the mean sea level (geoid) or other reference, such that it has a relatively fixed error compared to the ellipsoid, no significant error is introduced in NavLab. However, in such cases the depth output from NavLab will be relative to that same reference.*

Filename:	**attitude_smooth.txt**		
Contents:	Vehicle attitude relative local (*L*) North-East-Down (NED) system ($\boldsymbol{R}_{LB}$)		
Column 1:	**Column 2:**	**Column 3:**	**Column 4:**
time since 1970.01.01 [sec]	**roll** [rad]	**pitch** [rad]	**yaw**/heading (relative north) [rad]

Table D.2 Exported attitude file

Filename:	**velocity_smooth.txt**		
Contents:	Vehicle velocity relative Earth (*E*), decomposed in the body (*B*) system ($\boldsymbol{v}_{EB}^{B}$)		
Column 1:	**Column 2:**	**Column 3:**	**Column 4:**
time since 1970.01.01 [sec]	$v_{EB,x}^{B}$ [m/s]	$v_{EB,y}^{B}$ [m/s]	$v_{EB,z}^{B}$ [m/s]

Table D.3 Exported velocity file

The rate of the data corresponds to the rate of the Kalman filter (will vary according to the sensor measurement rates during the run).

If smoothing was not included in the estimation, or the result from the real-time Kalman filter is preferred, a similar script, `export_realtime_result.m`, can be used. The resulting file-formats are the same (but the file names are now `position_realtime.txt`, `attitude_realtime.txt` and `velocity_realtime.txt`.)

All exported files can be loaded and plotted (typically useful for verification) by running `plot_exported_results.m`. This script looks for the files at the current directory and plots all files available.

D.2 Format of files into the Estimator

When running the Estimator it will look for files containing sensor measurements. The sensor measurements found will be used in the estimation. IMU measurements must be present for the estimation to take place, all other sensors are optional.

The measurement files are located in the `/data` directory, and are usually produced by the Simulator or Preproc. However advanced users might want to make or modify these files, and thus their format is given below.

Note that the file-names and content directly[1] correspond to variables in the Estimator. Thus, the names are explained in Appendix A.

The Estimator assumes that the timing of the measurements is in seconds relative to a common time, `start_time`.

D.2.1 IMU measurements

Filename:	**`IMU_tv.txt`**
Contents:	The time of each IMU delta measurement in seconds after `start_time` (one column). The delta measurements are valid for an interval, and the timing is at the end of that interval.

Table D.4 *IMU time-vector*

Filename:	**`sav_delta_theta_gyro_m.txt`**		
Contents:	The delta theta angular increment from the gyros ($\Delta\tilde{\boldsymbol{\theta}}_{gyro}$). It is assumed to be the angle-axis product expressing the rotation during the interval after the previous sample, decomposed in the body (*B*) system.		
Column 1:	**Column 2:**	**Column 3:**	
$\Delta\tilde{\theta}_{gyro,x}$ [rad]	$\Delta\tilde{\theta}_{gyro,y}$ [rad]	$\Delta\tilde{\theta}_{gyro,z}$ [rad]	

Table D.5 *Delta theta measurements*

Filename:	**`sav_delta_v_acc_m.txt`**		
Contents:	The delta v velocity increment from the accelerometers ($\Delta\tilde{\boldsymbol{v}}_{acc}$) in the interval after last sample. It is assumed to be decomposed in the body (*B*) system at the beginning of the interval.		
Column 1:	**Column 2:**	**Column 3:**	
$\Delta\tilde{v}_{acc,x}$ [m/s]	$\Delta\tilde{v}_{acc,y}$ [m/s]	$\Delta\tilde{v}_{acc,z}$ [m/s]	

Table D.6 *Delta v measurements*

[1] The only difference is a transpose. NavLab follows the standard convention that column vectors are used to represent physical vectors. Saving many vectors (valid at different times) in a matrix thus means a new column for each time-step. When representing the same matrix in a text file it is transposed to make the file more readable.

D.2.2 Position measurements (posm)

Filename:	**`posm_tv.txt`**
Contents:	The time of each position measurement in seconds after `start_time` (one column)

Table D.7 Position measurement time-vector

Filename:	**`sav_n_E_posm_m.txt`**				
Contents:	The horizontal position measurement expressed as *n*-vector decomposed in the Earth (*E*) system ($\tilde{\boldsymbol{n}}^E_{posm}$).				
Column 1:		**Column 2:**		**Column 3:**	
$\tilde{n}^E_{posm,x}$	[no unit]	$\tilde{n}^E_{posm,y}$	[no unit]	$\tilde{n}^E_{posm,z}$	[no unit]

Table D.8 Horizontal position measurements

The *n*-vector is described briefly in Table A.4. To get the *n*-vector from longitude and geodetic latitude, use `long_lat2n_E.m` (included in NavLab). This function calculates a 3D *n*-vector from each pair of longitude/latitude (in radians).

To convert *n*-vector to longitude and geodetic latitude, use `n_E2long_lat.m`.

D.2.3 Depth measurements (depthm)

Filename:	**`depthm_tv.txt`**
Contents:	The time of each depth measurement in seconds after `start_time` (one column).

Table D.9 Depth measurement time-vector

Filename:	**sav_z_depthm_m.txt**
Contents:	The depth measurement, which equals the *z*-component of the vector from the local (*L*) system to the body (*B*) system ($\tilde{z}_{depthm} = \tilde{p}^{L}_{LB,depthm,z}$), thus positive direction is down.
Column 1:	
$\tilde{z}_{depthm}$	[m]

*Table D.*10 *Depth measurements*

NavLab assumes that the depth measurement is relative to the ellipsoid surface, using the same ellipsoid as the *n*-vector (or longitude/latitude) measurements (e.g. WGS-84). However, if the depth input is relative to the mean sea level (geoid) or other reference, such that it has a relatively fixed error compared to the ellipsoid, no significant error is introduced in NavLab. However, in such cases the depth output from NavLab will be relative to that same reference.

D.2.4 DVL measurements

Filename:	**DVL_tv.txt**
Contents:	The time of each DVL measurement in seconds after `start_time` (one column).

*Table D.*11 *DVL measurement time-vector*

Filename:	**sav_v_EB_B_DVL_m.txt**				
Contents:	The vehicle velocity relative Earth (*E*) decomposed in the body (*B*) system ($\tilde{\boldsymbol{v}}^{B}_{EB,DVL}$).				
Column 1:		**Column 2:**		**Column 3:**	
$\tilde{v}^{B}_{EB,DVL,x}$	[m/s]	$\tilde{v}^{B}_{EB,DVL,y}$	[m/s]	$\tilde{v}^{B}_{EB,DVL,z}$	[m/s]

*Table D.*12 *Velocity measurements*

D.2.5 Compass measurements (cmps)

Filename:	**`cmps_tv.txt`**
Contents:	The time of each compass measurement in seconds after `start_time` (one column).

*Table D.*13 *Compass measurement time-vector*

Filename:	**`sav_yaw_north_cmps_m.txt`**
Contents:	The compass measurement, which equals the yaw/heading angle relative north ($\tilde{\psi}_{north,cmps}$).
Column 1:	
$\tilde{\psi}_{north,cmps}$	[rad] in the interval [0 2π)

*Table D.*14 *Compass measurements*

D.2.6 Other files read by the Estimator

In addition to the sensor measurements in the `/data` directory the Estimator also looks for/reads other files in the working directory:

- ***sensor*`_est_quality.txt` (sensor quality files):** If the quality in any of the sensor measurements is changing, the quality of each single measurement is specified in files called *sensor*`_est_quality.txt`, where *sensor* represents any of the aiding sensors. Otherwise, the constant quality is specified in `estimator.ini`. For format and more details, see section 5.2.

- **`estimator.ini`:** General Estimator parameters, initial estimate and Kalman filter tuning.

- **`initial_P_KF_u.txt`:** A file containing the initial covariance matrix used by the Estimator.

- **`cov_matrix.ini`:** This ini-file is actually used by the script `make_and_save_cov_matrix.m`, to produce `initial_P_KF_u.txt`, but if selected the script is automatically run in the start of the Estimator.

E EXAMPLES OF INI-FILES

In the following, examples of the different ini-files are listed.

E.1 Simulator

E.1.1 General Simulator parameters

Filename: **Simulator.ini**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% simulator.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file contains simulator parameters, and is read by:
% Simulator.m
% Traj_sim_init.m

% Syntax:
variable = value


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        General parameters:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Choose the time interval to simulate (seconds):
sim_starttime = 0    % sec
sim_stoptime  = 240  % sec
h             = 0.1  % simulator time-step length, seconds


% Simulate IMU errors and make measurements (0 or 1):
IMU_available    = 1

% Simulate errors and make measurements from the aiding sensors (0 or 1):
posm_available   = 1
depthm_available = 1
DVL_available    = 1
cmps_available   = 1


% This variable decides if the simulation results should be saved to
% files (0 or 1). If 1, the Estimator can read the data later:
save_sim_data_to_files = 1


% Select a random seed (integer) to produce all measurement and process noise.
% Choose random_seed=0 to let the internal clock produce the seed:
random_seed = 1977


% By default the plot menu (plot_general.m) is called at the end of the
% simulation, alternatively this menu can be skipped:
skip_simulation_plot = 0


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        Trajectory initialization:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Initial horizontal position (and wander azimuth) (degrees):
Latitude_deg       = 60       % deg
Longitude_deg      = 10       % deg
Wander_azimuth_deg = 0        % deg

% Initial depth (note: z = p_LB_L_z) (meters):
z = 50                        % m, positive below sea level
```

```
% Initial attitude (degrees):
roll_deg  = 0                  % deg
pitch_deg = 0                  % deg
yaw_deg   = 90                 % deg

% Initial velocity (meters/second):
v_EB_B_x = 2                   % m/s
v_EB_B_y = 0                   % m/s
v_EB_B_z = 0                   % m/s
```

E.1.2 IMU simulator

Filename: **IMU_sim.ini**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% IMU_sim.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file contains IMU simulation parameters, and is read by:
% IMU_sim_init.m

% Syntax:
variable = value


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  Measurement rate of the sensor:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If sensor_tv_from_file==0, then the given constant rate is used, otherwise an
% (arbitrary) user specified time-vector saved on file is used:

% NOTE: Currently, only constant time-steps are allowed for IMU.

IMU_tv_from_file = 0
h_IMU            = 0.1   % IMU time-step length, seconds


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                           Gyros:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Current values: Honeywell HG 1700 Inertial measurement unit

%%%%%%%%%%%%% Gyro continous measurement noise, Angular Random Walk (mn_gyro):
% Magnitude ( power density, rad/sqrt(s) )
% ( 0.1 deg/sqrt(h) = 2.91e-5 rad/sqrt(s) )
pd_mn_gyro_x = 2.91e-5        % rad/sqrt(s)
pd_mn_gyro_y = 2.91e-5        % rad/sqrt(s)
pd_mn_gyro_z = 2.91e-5        % rad/sqrt(s)


%%%%%%%%%%%%% Gyro bias (Domg_gyro_bias):
%  Magnitude (rad/s)
% (1 deg/h = 4.85e-6 rad/s)
std_Domg_gyro_bias_x = 4.85e-6       % rad/s
std_Domg_gyro_bias_y = 4.85e-6       % rad/s
std_Domg_gyro_bias_z = 4.85e-6       % rad/s

% Timeconstant (seconds):
T_Domg_gyro_bias_x   = 600    % s
T_Domg_gyro_bias_y   = 600    % s
T_Domg_gyro_bias_z   = 600    % s


%%%%%%%%%%%%% Gyro scale factor error (Domg_gyro_sf, no unit):
% (100 ppm = 1e-4)
std_Domg_gyro_sf_x = 1e-4
std_Domg_gyro_sf_y = 1e-4
std_Domg_gyro_sf_z = 1e-4
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                              Accelerometers:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Current values: Honeywell HG 1700 Inertial measurement unit

%%%%%%%%%%%%% Accelerometer continous measurement noise (mn_acc):
% Magnitude ( power density, m/s^(3/2) )
% ( 10 micro g/sqrt(Hz) = 9.81e-5 m/s^(3/2) )
pd_mn_acc_x = 9.81e-5         % m/s^(3/2)
pd_mn_acc_y = 9.81e-5         % m/s^(3/2)
pd_mn_acc_z = 9.81e-5         % m/s^(3/2)

%%%%%%%%%%%%% Accelerometer bias (Df_acc_bias):
% Magnitude (m/s^2):
% (1 milli g = (1e-3)*9.81 m/s^2, 9.81e-3 m/s^2 + contribution from
% misalignment = 1.1e-2 m/s^2
std_Df_acc_bias_x = 1.1e-2    % m/s^2
std_Df_acc_bias_y = 1.1e-2    % m/s^2
std_Df_acc_bias_z = 1.1e-2    % m/s^2

% Timeconstant (seconds):
T_Df_acc_bias_x   = 600       % s
T_Df_acc_bias_y   = 600       % s
T_Df_acc_bias_z   = 600       % s


%%%%%%%%%%%%% Accelerometer scale factor error (Df_acc_sf, no unit):
% (200 ppm = 2e-4)
std_Df_acc_sf_x = 2e-4
std_Df_acc_sf_y = 2e-4
std_Df_acc_sf_z = 2e-4
```

E.1.3 Position measurement simulator

Filename: **posm_sim.ini**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% posm_sim.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file contains position measurement (posm) simulation parameters, and is
% read by:
posm_sim_init.m

% Syntax:
variable = value


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    Measurement rate of the sensor:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If sensor_tv_from_file==0, then the given constant rate is used, otherwise an
% (arbitrary) user specified time-vector saved on file is used:

posm_tv_from_file = 0
h_posm            = 5     % posm time-step length, seconds


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  posm white measurement noise (w_posm_dp):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Standard deviation (in L, meters):

std_w_posm_dp_x = 1.5  % m
std_w_posm_dp_y = 1.5  % m


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        posm bias (Dp_posm_bias):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This variable decides if the standard deviation of the bias in
% posm is depth dependent (which is typical for acoustic position, HPR/HiPAP)
std_posm_bias_is_depth_dependent = 0;
```

```
This is used if depth dependent:
% Magnitude of angular uncertainty (degrees):
std_Dp_posm_bias_angle_deg = 0.3 % deg


This is used if not depth dependent:
% Magnitude (in L, meters):
std_Dp_posm_bias_x = 2        % m
std_Dp_posm_bias_y = 2        % m


% Timeconstant (in L, T_Dp_posm_bias=T_e_posm_bias) (seconds):
T_Dp_posm_bias_x = 60         % s
T_Dp_posm_bias_y = 60         % s
```

E.1.4 Depth measurement simulator

Filename: **depthm_sim.ini**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% depthm_sim.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file contains depth measurement (depthm) simulation parameters, and is
% read by:
% depthm_sim_init.m

% Syntax:
variable = value


% Current values: Pharoscientific Digiquartz FS = 3000 m

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                  Measurement rate of the sensor:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If sensor_tv_from_file==0, then the given constant rate is used, otherwise an
% (arbitrary) user specified time-vector saved on file is used:

depthm_tv_from_file = 0
h_depthm            = 0.5     % depthm time-step length, seconds


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                 depthm white measurement noise (w_depthm):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Standard deviation (meters):
std_w_depthm = 0.02           % m


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                      depthm bias (Dz_depthm_bias):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Magnitude (meters):
std_Dz_depthm_bias = 0.15     % m

% Timeconstant (seconds):
T_Dz_depthm_bias   = 100      % s
```

E.1.5 DVL simulator

Filename: **DVL_sim.ini**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%% DVL_sim.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file contains DVL simulation parameters, and is read by:
% DVL_sim_init.m

% Syntax:
variable = value
```

```
% The current values: RDI WorkHorse Navigator DVL, 300 kHz

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    Measurement rate of the sensor:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If sensor_tv_from_file==0, then the given constant rate is used, otherwise an
% (arbitrary) user specified time-vector saved on file is used:

DVL_tv_from_file = 0
h_DVL            = 1   %DVL time-step length, sec


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                       DVL white measurement noise (w_DVL):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Standard deviation (m/s):
% (0.6 cm/s = 6e-3 m/s)
std_w_DVL_x = 6e-3            % m/s
std_w_DVL_y = 6e-3            % m/s
std_w_DVL_z = 6e-3            % m/s


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                              DVL bias (Dv_DVL_bias):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Magnitude (m/s):
% (0.3 cm/s = 3e-3 m/s
std_Dv_DVL_bias_x = 3e-3      % m/s
std_Dv_DVL_bias_y = 3e-3      % m/s
std_Dv_DVL_bias_z = 3e-3      % m/s

% Timeconstant (seconds):
T_Dv_DVL_bias_x = 800         % s
T_Dv_DVL_bias_y = 800         % s
T_Dv_DVL_bias_z = 800         % s


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    DVL scale factor error (Dv_DVL_sf, no unit):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
std_Dv_DVL_sf_x = 100e-6
std_Dv_DVL_sf_y = 100e-6
std_Dv_DVL_sf_z = 100e-6
```

E.1.6 Compass simulator

Filename: **cmps_sim.ini**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%% cmps_sim.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file contains compass simulation parameters, and is read by:
% cmps_sim_init.m

% Syntax:
variable = value



% Current values: Octans gyrocompass

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    Measurement rate of the sensor:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% If sensor_tv_from_file==0, then the given constant rate is used, otherwise an
% (arbitrary) user specified time-vector saved on file is used:

cmps_tv_from_file = 0
h_cmps            = 0.4   % cmps time-step length, seconds
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                      cmps white measurement noise (w_cmps):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Standard deviation (degrees):
std_w_cmps_deg         = 0.01 % deg


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        cmps bias (Dyaw_cmps_bias):
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Magnitude (degrees):
% Specification: 0.7 deg * sec(lat)
std_Dyaw_cmps_bias_deg = 0.9  % deg

% Timeconstant (seconds):
T_Dyaw_cmps_bias       = 600  % s
```

E.1.7 Change-rate of the trajectory

Filename: **Trajrate.m**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Trajrate.m %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% function [omg_LB_B_a,v_EB_B_d_a]=Trajrate(t,h)
%
% The rate of change in attitude and velocity is returned, as function
% of the time (t).
%
% In:
% t: (seconds) Current time
% h: (seconds) The amount of time since last time-step
%
% Out:
% omg_LB_B_a: (rad/s) Average omg_LB_B in the interval from t-h to t
% v_EB_B_d_a: (m/s^2) Average change in v_EB_B in the interval from t-h to t
%                     _d means dot (= time derivative)
%
%
% Note: This is a local copy of the file, located in the users working directory.
% The original is located in the NavLab directory, under Simulator.

% Originated: 1999.01.24 Kenneth Gade, FFI
% Modified:
%
% Possible modifications:
%

function [omg_LB_B_a,v_EB_B_d_a]=Trajrate(t,h)


% Default values:
omg_LB_B_a_deg=[0,0,0]';
v_EB_B_d_a=[0,0,0]';


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% User selectable. Specify average v_EB_B_d and omg_LB_B for the interval t-h to t.
% omg_LB_B_a_deg: deg/s
% v_EB_B_d_a:     m/s^2


%%%%%% Example:
if t>=50  & t<60,     v_EB_B_d_a=[0.21,0,0]';    end; % accelerate
if t>=100 & t<106,    omg_LB_B_a_deg=[0,0,15]';  end; % turn right
if t>=120 & t<124.5,  omg_LB_B_a_deg=[0,-10,0]'; end; % pitch down
if t>=140 & t<144.5,  omg_LB_B_a_deg=[0,10,0]';  end; % pitch up
if t>=160 & t<166,    omg_LB_B_a_deg=[0,0,15]';  end; % turn right
%%%%%%%%%%


% End of user selectable variables
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
% Converts the angular rate in deg/s to radians/s
omg_LB_B_a=rad(omg_LB_B_a_deg);
```

E.2 Estimator

E.2.1 General Estimator parameters

Filename: Estimator.ini

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%% estimator.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file contains Estimator parameters, and is read by:
% Estimator.m
% naveq_init.m
% make_and_save_cov_matrix.m
% + several others

% Syntax:
variable = value



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        General parameters:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% By default, the entire IMU interval is used by the Estimator.
% If this variable is 0, a smaller interval may be selected:
use_entire_IMU_interval = 1

% The desired start and stop times (nearest IMU time-steps will be used).
% These values will be used if use_entire_IMU_interval = 0:
est_starttime_desired = 45    % s
est_stoptime_desired  = 200   % s


% Making smoothed estimates requires a lot of memory. This variable decides
% if smoothed estimates should be made (0 or 1):
make_smoothed_estimates = 1


% The navigation equations should be reset every k_KF (Kalman filter time-step),
% but longer intervals could be chosen to better view the drift etc:
n_of_k_KF_per_reset     = 5   % number of KF time-steps per reset


% Only the navigation equations can be run (no estimation):
naveq_only = 0


% The Estimator needs an initial covariance matrix. This can be manually created
% and saved to a file (initial_P_KF_u.txt) by running make_and_save_cov_matrix.m
% before running the Estimator. Alternatively the Estimator can automatically
% run make_and_save_cov_matrix.m itself:
auto_create_cov_matrix = 1

% The initial covariance matrix (P_KF_u) is by default read from a file
% (initial_P_KF_u.txt), if not initialization is assumed to be perfect:
initial_P_KF_from_file  = 1


% By default the plot menu (plot_general.m) is called at the end of the
% estimation, alternatively this menu can be skipped:
skip_estimation_plot = 0


% This variable decides if the complete set of estimation results should automatically
% be saved to txt-files (0 or 1). Alternatively, save the workspace manually by typing
% save in the command window. (The latter is often preferred.)
save_est_data_to_files  = 0
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        Naveq initialization:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initial horizontal position (and wander azimuth) (degrees)
Latitude_deg       = 60       % deg
Longitude_deg      = 10       % deg
Wander_azimuth_deg = 0        % deg

% Initial depth (note: z = p_LB_L_z) (meters)
z         = 50               % m, positive below sea level

% Initial attitude (degrees):
roll_deg  = 0                % deg
pitch_deg = 0                % deg
yaw_deg   = 90               % deg

% Initial velocity (meters/second):
v_EB_B_x = 2                 % m/s
v_EB_B_y = 0                 % m/s
v_EB_B_z = 0                 % m/s



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        Kalman filter parameters:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Gyros %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Current values: Honeywell HG 1700 Inertial measurement unit

%%%%%%%%%%%%% Gyro continous measurement noise, Angular Random Walk (mn_gyro):
% Magnitude ( power density, rad/sqrt(s) )
% ( 0.1 deg/sqrt(h) = 2.91e-5 rad/sqrt(s) )
pd_mn_gyro_x         = 2.91e-5       % rad/sqrt(s)
pd_mn_gyro_y         = 2.91e-5       % rad/sqrt(s)
pd_mn_gyro_z         = 2.91e-5       % rad/sqrt(s)


%%%%%%%%%%%%% Gyro bias (Domg_gyro_bias):
%  Magnitude (rad/s)
% (1 deg/h = 4.85e-6 rad/s)
std_Domg_gyro_bias_x = 4.85e-6       % rad/s
std_Domg_gyro_bias_y = 4.85e-6       % rad/s
std_Domg_gyro_bias_z = 4.85e-6       % rad/s

% Timeconstant (seconds):
T_Domg_gyro_bias_x   = 600           % s
T_Domg_gyro_bias_y   = 600           % s
T_Domg_gyro_bias_z   = 600           % s


%%%%%%%%%%%%%%%%%%%%%%%%%%%% Accelerometers %%%%%%%%%%%%%%%%%%%%%%%%%%%%
Current values: Honeywell HG 1700 Inertial measurement unit

%%%%%%%%%%%%% Accelerometer continous measurement noise (mn_acc):
% Magnitude ( power density, m/s^(3/2) )
% ( 10 micro g/sqrt(Hz) = 9.81e-5 m/s^(3/2) )
pd_mn_acc_x        = 9.81e-5  % m/s^(3/2)
pd_mn_acc_y        = 9.81e-5  % m/s^(3/2)
pd_mn_acc_z        = 9.81e-5  % m/s^(3/2)

%%%%%%%%%%%%% Accelerometer bias (Df_acc_bias):
% Magnitude (m/s^2):
% (1 milli g = (1e-3)*9.81 m/s^2, 9.81e-3 m/s^2 + contribution from
% misalignment = 1.1e-2 m/s^2
std_Df_acc_bias_x  = 1.1e-2   % m/s^2
std_Df_acc_bias_y  = 1.1e-2   % m/s^2
std_Df_acc_bias_z  = 1.1e-2   % m/s^2

% Timeconstant (seconds):
T_Df_acc_bias_x    = 600      % s
T_Df_acc_bias_y    = 600      % s
T_Df_acc_bias_z    = 600      % s
```

```
%%%%%%%%%%%%%%%%%%%%%% Position measurement (posm) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%% posm white measurement noise (w_posm_dp):
% Standard deviation (in L, meters):
std_w_posm_dp_x    = 1.5      % m
std_w_posm_dp_y    = 1.5      % m

%%%%%%%%%%%%% posm bias (Dp_posm_bias):
% This variable decides if the standard deviation of the bias in
% posm is depth dependent (which is typical for acoustic position, HPR/HiPAP)
std_posm_bias_is_depth_dependent = 0

This is used if depth dependent:
% Magnitude of angular uncertainty (degrees):
std_Dp_posm_bias_angle_deg = 0.3     % deg

This is used if not depth dependent:
% Magnitude (in L, meters):
std_Dp_posm_bias_x = 2        % m
std_Dp_posm_bias_y = 2        % m

% Timeconstant (in L, T_Dp_posm_bias=T_e_posm_bias) (seconds):
T_Dp_posm_bias_x   = 60       % s
T_Dp_posm_bias_y   = 60       % s


%%%%%%%%%%%%%%%%%%%%%% Depth measurement (depthm) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Current values: Pharoscientific Digiquartz FS = 3000 m

%%%%%%%%%%%%% depthm white measurement noise (w_depthm):
% Standard deviation (meters):
std_w_depthm       = 0.02     % m

%%%%%%%%%%%%% depthm bias (Dz_depthm_bias):
%  Magnitude (meters):
std_Dz_depthm_bias = 0.15     % m

% Timeconstant (seconds):
T_Dz_depthm_bias   = 100      % s


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% DVL %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Current values: RDI workhorse navigator, 300 kHz:

%%%%%%%%%%%%% DVL white measurement noise (w_DVL):
% Standard deviation (m/s):
% (0.6 cm/s = 6e-3 m/s)
std_w_DVL_x       = 6e-3      % m/s
std_w_DVL_y       = 6e-3      % m/s
std_w_DVL_z       = 6e-3      % m/s

%%%%%%%%%%%%% DVL bias (Dv_DVL_bias):
% Magnitude (m/s):
% (0.3 cm/s = 3e-3 m/s
std_Dv_DVL_bias_x = 3e-3      % m/s
std_Dv_DVL_bias_y = 3e-3      % m/s
std_Dv_DVL_bias_z = 3e-3      % m/s

% Timeconstant (seconds):
T_Dv_DVL_bias_x   = 800       % s
T_Dv_DVL_bias_y   = 800       % s
T_Dv_DVL_bias_z   = 800       % s


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Compass (cmps) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Current values: Octans gyrocompass

%%%%%%%%%%%%% cmps measurement noise (w_cmps):
% Standard deviation (degrees):
std_w_cmps_deg    = 0.01      % deg

%%%%%%%%%%%%% cmps bias (Dyaw_cmps_bias):
% Magnitude (degrees):
% Specification: 0.7 deg * sec(lat)
std_Dyaw_cmps_bias_deg = 0.9  % deg

% Timeconstant (seconds):
T_Dyaw_cmps_bias       = 600  % s
```

E.2.2 Initial covariance matrix

Filename: **cov_matrix.ini**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% cov_matrix.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file contains values for the initial Kalman filter covariance matrix,
% and is read by:
% make_and_save_cov_matrix.m


IMPORTANT: Make sure estimator.ini has the correct values before running
          make_and_save_cov_matrix.m


% Syntax:
variable = value


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                    Initial naveq uncertainty:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%% Attitude uncertainty
% Uncertainty in initial attitude estimate (in degrees):
% (The initial attitude estimate is set in estimator.ini)

std_e_LB_L_x_deg = 0.2   % deg, (Roll when yaw and pitch is 0)
std_e_LB_L_y_deg = 0.2   % deg, (Pitch when yaw and roll is 0)
std_e_LB_L_z_deg = 2     % deg, (Yaw (Heading) when roll and pitch is 0)



%%%%%%%%%%%%% Velocity uncertainty
% Uncertainty in initial velocity estimate (in m/s):
% (The initial velocity estimate is set in estimator.ini)

std_dv_EB_B_x = 0.1      % m/s, Vehicle fore-aft velocity
std_dv_EB_B_y = 0.1      % m/s, Vehicle starboard-port velocity
std_dv_EB_B_z = 0.1      % m/s, Vehicle down-up velocity



%%%%%%%%%%%%% Position uncertainty
% Uncertainty in initial position estimate (in meters):
% (The initial position estimate is set in estimator.ini)

std_dp_L_x = 2           % m, North
std_dp_L_y = 2           % m, East
std_dp_L_z = 0.3         % m, Down


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                         Correlations:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% If the aiding sensors were used to get the initial estimate, errors are correlated
% (which should be reflected by the off-diagonal elementes in the cov matrix).

posm_was_used_for_init   = 1  % 0 = false, 1 = true
depthm_was_used_for_init = 1  % 0 = false, 1 = true
DVL_was_used_for_init    = 1  % 0 = false, 1 = true
cmps_was_used_for_init   = 1  % 0 = false, 1 = true
```

E.3 Preproc

Filename: **preproc.ini**

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%% preproc.ini %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% This file contains parameters for preproc.m
%
%
% Syntax:
variable = value


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% File numbers:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Hugin stores several sequential files with limited length (e.g. 15 min) containing
% sensor data.
%
% Example: DVL-data is stored in files called:
% C:\data\DVLData0001.txt
% C:\data\DVLData0002.txt
%   etc...

% If this variable is 1, all files that are found are loaded for each sensor:
load_all_files = 1

% If a smaller interval should be loaded, select the number of the first and last
% file to be included:
file_start_no = 9
file_stop_no  = 10


%%% Advanced:
% By default the sequential files described above are located in the \data-directory
% under the current directory. In special cases, these files may be located
% somewhere else (perhaps at a read-only location).
%
% Note that only sequential files are read from the alternative location, thus a copy
% of the raw_posm.txt (pos.txt) from the ship (if used) must still be in the \data-directory

use_default_sequentialfile_location = 1     % Default is 1

% If not default is used, this location is used:
alternative_sequentialfile_location = G:\RecordedData\Run20010710_2\data\

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        Automated user input:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Normally preproc prompts the user to do different selections. However, the
% user input can also be set in advance:

% After the IMU data is loaded, it is possible to reduce the interval found in
% the IMU data:
use_entire_interval_first_time = 0    % 0 : No (let user decide)
                                      % 1 : Yes



%%%%%% Depth decimation:
% Often, the depth measurement is available at a higher rate than needed by the Estimator.
% Decimating the depth measurement will increase estimation speed.
decimate_depthm = 2  % 0 : No
                     % 1 : Yes
                     % 2 : Let user decide

% If decimate_depthm == 1, then this rate will be used automatically:
desired_depthm_rate = 1 % Hz
```

```
%%%%% Compass decimation:
% Often, the compass is available at a higher rate than needed by the Estimator.
% Desimating the compass measurement will increase estimation speed.
decimate_cmps = 2  % 0 : No
                   % 1 : Yes
                   % 2 : Let user decide

% If decimate_cmps == 1, then this rate will be used automatically:
desired_cmps_rate = 2 % Hz



% After all the sensor graphs are plotted, it is possible to reduce the interval
% a second time:
use_entire_interval_second_time = 0    % 0 : No (let user decide)
                                       % 1 : Yes

% The wildpoint edit menu may be skipped:
skip_wildpoint_edit = 0  % 0 : No
                         % 1 : Yes



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                          Tuning of automatic wildpoint detection:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% For some sensors wildpoints can be detected automatically. The following parameters
% are used to tune the detection algorithm.
%
% NOTE: These parameters are read each time the algorithm is run, and thus you can
% repeatedly tune the algorithm and press "Detect wildpoints" until the tuning is fine.
%

%%%% posm wildpoints:
% If the new horizontal position measurement indicates a velocity (based on the
% established position) exceeding this limit, then the measurement is assumed to be a
% wildpoint:
wp_posm_velocity_limit = 5  % m/s

% To avoid "being stuck outside the truth" it is not allowed to reject more than a
% limited number of consecutive measurements:
wp_posm_n_of_consecutive_wp_limit = 4



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                          Which position measurement to use:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Normally there are two possible position files to use. The best is the one stored on the
% surface ship, because it has a higher rate. However it is also possible to make a file
% from DGPSHiPAPData0001.txt etc which are the position measurements stored on HUGIN
% (with low rate). Using the latter will overwrite raw_posm.txt from the surface ship
% (if it exists).

use_posm_from_surface_ship = 1   % 1 is default



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                          Lever arms:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Measurements from the sensors are assumed to be valid in B. However, their physical
% distribution gives us several B systems: BIMU, Bposm, Bdepthm, BDVL and Bcmps.
%
%
% The physical location of the different sensors, relative a reference point (Bref),
% should be given. The Estimator will give its results in IMUs position.

% All lever arms (vector from Bref to Bsensor) are decomposed in the body system:
% x: forward (towards the nose)
% y: starboard (right)
% z: down

% The numbers are given in meters.
```

```
% Hugin3000 parameters
% From Odd Arild Pedersen 27th of October 2000.
% Bref = the tip of HUGIN 3000 nose cone

% IMU:
p_Bref_BIMU_Bx     = -3.620   % m
p_Bref_BIMU_By     =  0       % m
p_Bref_BIMU_Bz     =  0.076   % m


% posm:
% NOTE: TWO different transducers are used, depending on depth!
%
% 324: Is used when depth is 0 to 1000 m.
% 331: narrow-beam, is typically used when deeper than 1000 m
%
% 324: (above 1000 m)
p_Bref_Bposm_Bx    = -3.4747  % m
p_Bref_Bposm_By    = -0.215   % m
p_Bref_Bposm_Bz    = -0.3967  % m

% 331: (below 1000 m)
% p_Bref_Bposm_Bx  = -3.3437  % m
% p_Bref_Bposm_By  = -0.215   % m
% p_Bref_Bposm_Bz  = -0.4474  % m



% depthm:
p_Bref_Bdepthm_Bx  = -3.620   % m
p_Bref_Bdepthm_By  = -0.145   % m
p_Bref_Bdepthm_Bz  =  0.284   % m

% DVL:
p_Bref_BDVL_Bx     = -3.355   % m
p_Bref_BDVL_By     =  0       % m
p_Bref_BDVL_Bz     =  0.456   % m

% cmps (Not relevant when used only as orientation sensor):
p_Bref_Bcmps_Bx    = 0        % m
p_Bref_Bcmps_By    = 0        % m
p_Bref_Bcmps_Bz    = 0        % m




%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                            Misalignment:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Measurements from the sensors are assumed to be decomposed in B. However,
% if not mounted properly the misalignment must be compensated for.
%
% Assume that the sensor first has a nonzero roll, pitch and yaw angle, relative
% to the true B. Enter the values that describe the misalignment (in deg):
% (Note that the sign is from true to error, i.e. the angles describe the
% misalignment, not the correction)

% DVL:
DVL_roll_deg   = 0   % deg
DVL_pitch_deg  = 0   % deg
DVL_yaw_deg    = 0   % deg

% cmps:
cmps_roll_deg  = 0   % deg
cmps_pitch_deg = 0   % deg
cmps_yaw_deg   = 0   % deg



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                            Known constant sensor errors:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Due to bad calibration or other reasons, constant (repeatable) errors may be
% present in sensors. If these errors are known, the values can be entered below
% and will be compensated for by preproc.

% Constant gyro error (deg/h):
Domg_gyro_constant_deg_pr_h_x = 0 % deg/h
Domg_gyro_constant_deg_pr_h_y = 0 % deg/h
Domg_gyro_constant_deg_pr_h_z = 0 % deg/h
```

```
% Constant accelerometer error (milli g):
Df_acc_constant_mg_x          = 0 % milli g
Df_acc_constant_mg_y          = 0 % milli g
Df_acc_constant_mg_z          = 0 % milli g



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                        Varying sensor quality:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% For some sensors we have knowledge of how their accuracy are varying, e.g. by
% means of numbers reported by the sensor. For these sensors the quality in
% each single measurement should be sent to the Kalman filter. To calculate
% this quality based on the reported numbers and other variables, the parameters
% below are used.


% Position measurement may be received from different sources, and different
% models should be used depending on the source. In the following parameters
% for 3 different sources are listed:
%
% 1: DGPS-HiPAP (standard AUV position measurement)
% 2: GPS (from own receiver when vehicle at surface)
% 3: ShipGPS (received when AUV on deck of a ship)


%%%%%%%%%%%%%%%%%% posm: Combined DGPS-HiPAP %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Usually, two numbers are received from the sensor: assumed standard deviation
% in DGPS (DGPS_qlty) and in HiPAP (HiPAP_qlty), in addition we might have a
% depth dependency.
%
%


% Factor to multiply reported DGPS_qlty:
DGPS_qlty2std_fctr  = 1   % default is 1


% Factor to multiply reported HiPAP_qlty:
HiPAP_qlty2std_fctr = 1   % default is 1

% Factor to calculate horizontal position accuracy from depth (in degrees):
depth2std_fctr_deg  = 0.1 % deg


%%%%%%%%%%%%% White measurement noise (w_posm_dp):

% fixed part (in L, meters):
std_w_DGPS_HiPAP_fxd_x = 0      % m
std_w_DGPS_HiPAP_fxd_y = 0      % m


% White part of DGPS_qlty (in the interval [0 1] ):
wht_part_DGPS  = 0.2 % Assuming that 20% of the DGPS error is white-noise


% White part of HiPAP_qlty (in the interval [0 1] ):
wht_part_HiPAP = 0.9 % Assuming that 90% of the HiPAP error is white-noise


% White part of depth dependent error (in the interval [0 1] ):
wht_part_depth = 0 % White-noise does not include the depth dependent part



%%%%%%%%%%%%% Colored/bias error (Dp_posm_bias):

% fixed part (in L, meters):
std_DGPS_HiPAP_bias_fxd_x = 1.4    % m
std_DGPS_HiPAP_bias_fxd_y = 1.4    % m


% Colored part of DGPS_qlty (in the interval [0 1] ):
col_part_DGPS  = 0 %NOTE: To avoid jumps in the colored error, this number is usually 0


% Colored part of HiPAP_qlty (in the interval [0 1] ):
col_part_HiPAP = 0 %NOTE: To avoid jumps in the colored error, this number is usually 0
```

```
% Colored part of depth dependent error (in the interval [0 1] ):
col_part_depth = 1 % Assuming that 100% of the depth dependent error is colored



%%%%%%%%%%%% Timeconstant (in L, T_Dp_posm_bias) (seconds):
T_Dp_DGPS_HiPAP_bias_x   = 120       % s
T_Dp_DGPS_HiPAP_bias_y   = 120       % s




%%%%%%%%%%%%%%%%%% posm: GPS from own Hugin receiver %%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Factor to multiply reported GPS_qlty:
GPS_qlty2std_fctr  = 1   % default is 1

%%%%%%%% White measurement noise:
std_w_GPS_fxd_x    = 0   % m
std_w_GPS_fxd_y    = 0   % m

% White part of GPS_qlty (in the interval [0 1] ):
wht_part_GPS       = 0.2


%%%%%%%% Colored/bias error:
std_GPS_bias_fxd_x = 0   % m
std_GPS_bias_fxd_y = 0   % m

% Colored part of GPS_qlty (in the interval [0 1] ):
col_part_GPS       = 0.8


%%%%%%%% Timeconstant (seconds):
T_Dp_GPS_bias_x    = 60  % s
T_Dp_GPS_bias_y    = 60  % s




%%%%%%%%%%%%%%%%%% posm: Position received onboard ship %%%%%%%%%%%%%%%%%%%%%%%
%
%
% Factor to multiply reported ShipGPS_qlty:
ShipGPS_qlty2std_fctr  = 0    % default is 1

%%%%%%%% White measurement noise:
std_w_ShipGPS_fxd_x    = 1.0  % m
std_w_ShipGPS_fxd_y    = 1.0  % m

% White part of ShipGPS_qlty (in the interval [0 1] ):
wht_part_ShipGPS       = 0.0


%%%%%%%% Colored/bias error:
std_ShipGPS_bias_fxd_x = 5.0  % m
std_ShipGPS_bias_fxd_y = 5.0  % m

% Colored part of ShipGPS_qlty (in the interval [0 1] ):
col_part_ShipGPS       = 0.0

%%%%%%%% Timeconstant (seconds):
T_Dp_ShipGPS_bias_x    = 60   % s
T_Dp_ShipGPS_bias_y    = 60   % s
```

F ABBREVIATIONS AND ACRONYMS

AUV	Autonomous Underwater Vehicle
DVL	Doppler Velocity Log
GPS	Global Positioning System
HiPAP	High Precision Acoustic Positioning
IMU	Inertial Measurement Unit
KF	Kalman Filter
NavLab	Navigation Laboratory
NED	North-East-Down
QC	Quality Control
RMS	Root Mean Square
WGS	World Geodetic System
ZUPT	Zero velocity update

References

(1) Gade Kenneth (1997): Integrering av treghetsnavigasjon i en autonom undervannsfarkost (in Norwegian), FFI/RAPPORT-97/03179, Ugradert

(2) Gelb Arthur (1974): Applied Optimal Estimation, The M.I.T. Press, Cambridge.

(3) Larsen R J and Marx M L (1986): Mathematical Statistics and Its Applications, Prentice-Hall, New Jersey.