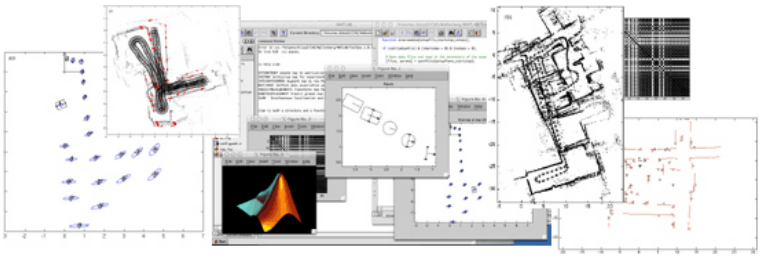
The CAS Robot Navigation Toolbox



Quick Guide

Kai O. Arras, CAS–KTH

Version 1.0, January 31st, 2004

Contents

1 About This Manual

This manual is a quick guide to the toolbox which gives an overview of its contents and functionality. Refer to the Users Guide and Reference and the Programmers Guide for more detailed information.

Note that to each method, m-file or experimental setup file has a help text which can be called by typing `>> help filename` (without the Matlab prompt and the `.m` extension). In most cases this gives a first and sometimes sufficient idea of what the file is about.

2 Introduction

2.1 What Does This Toolbox Do?

It is a tool to do localization and SLAM off-board and off-line with sensory data which have been previously recorded with a robot. The toolbox has been designed to be general with respect to the choice of the feature representation, the robot and the sensor models. This simplifies exchange of representations, methods and approaches and might help to encourage comparative experiments and studies. In detail, the toolbox...

- decouples features from algorithms. Add your own feature class and a predefined number of methods and do localization or SLAM without changing the localization or SLAM code
- decouples the robot model from algorithms. Add a robot, a predefined number of methods and a state prediction function which implements dead-reckoning (e.g. odometry) for your robot given some (proprioceptive) sensory data. Again, this does not affect the rest of the toolbox
- decouples sensor models from algorithms. Have an arbitrary number of sensors whose data are in an (almost) arbitrary file format
- allows to exchange a feature extraction or dead-reckoning method without changes in the code
- allows you to add you non-parametric approach to localization and SLAM
- keeps all experimental parameters at a single place. Add your own parameters there and reuse them where you want. This simplifies reproduction and management of experiments

2.2 What It Doesn't Do

The toolbox has currently some limitations:

- Version 1.0 is feature-based. No non-parametric approaches to localization and SLAM (e.g. particle filters) have been implemented although they could be integrated
- As the toolbox is written in Matlab and Matlab is an interpreted language, computational speed might be low on slow computers and/or big data sets
- Version 1.0 ships only with a differential drive robot class, a x,y-point feature class and a α,r-line feature class. If this does not match your robot setting, you have to add the respective robot and feature classes yourself.
- No 3D support. World, sensors and robots are assumed to be in a 2D-universe

2.3 Why Matlab?

Working with a navigation toolbox in Matlab has advantages and drawbacks. With the purpose of a tool for research and education, the advantages are breifly:

- Matlab is a wide-spread, standardized and mature programming environment with a big number of built-in functionalities and additions
- Programming in Matlab is easy
- Matlab runs on many platforms and operating systems
- The toolbox requires no installation or compilation (when Matlab is installed)

Whereas the drawback are mainly two points:

- Matlab is commercial and costly
- Although operations on matrices are very fast, overall speed of Matlab can be poor

2.4 Resources, License and Feedback

The toolbox is available under a GNU GPL license and therefore usable by anybody without warranty, charge, and on an "as is" basis. A GNU GPL license means however that you are not allowed to use this toolbox or parts of it with a commercial purpose. See details in the attached license file `License.txt` and the GNU GPL license text in the file `COPYING`.

The toolbox' homepage is at `http://www.cas.kth.se/toolbox`. It comes as a zipped tar-file and is accompanied by the mentioned licence files. The only

system requirement is Matlab version greater or equal than six. No additional Matlab toolboxes are required.

As any piece of software, this toolbox grows by its use in practice in a variety of different setting. When you encounter a problem, a bug, or you would like to make a suggestion for future versions, do so by writing an email to `navtoolbox@nada.kth.se`. Please add information on your computer, OS, Matlab version etc. We do appreciate your feedback!

3 Getting Started

Before we dive into any details, let us first have a look on the general structure of the toolbox. It will be helpful to understand where information flows and in what form.

3.1 Overall Structure

Figure 1 shows the three-layered structure of the toolbox. On the bottom is the data generating layer whose output are streams of raw sensory data. They are the content of the specified sensor data files.

The second layer is the dead reckoning and feature extraction layer. Given the respective stream of raw sensory data (e.g. encoder or IMU values for dead reckoning, range data for feature extraction), the task of this layer is to produce the robot state prediction $x(k+1|k), P(k+1|k)$ and the observation $z(k+1), R(k+1)$. The latter is sometimes called the *local map*.

The top layer is the localization and SLAM method. In case of a localization experiment, an a priori map is an additional input to this layer. Given the observation $z(k+1), R(k+1)$ and the robot state prediction $x(k+1|k), P(k+1|k)$ of each discrete time step k, the algorithm in this layer builds a map or localizes a robot using the implemented method. The toolbox' default method is an extended Kalman filter with a nearest neighbor matching.

The "heartbeat" of the toolbox is the discrete time index k. The sensor which is defined to be the master sensor triggers a new step. At each new step, the data importer reads the raw data from all sensor files between the last step k and the new step $k+1$. The timesteps associated to the data guarantee temporal consistency across all sensors. The master sensor, which can be any sensor, is defined in the set up file of the experiment. Typically it is one of the exteroceptive sensors such as a laser scanner.

Future releases of the toolbox might contain an additional simulation part (gray) in the data generating layer.

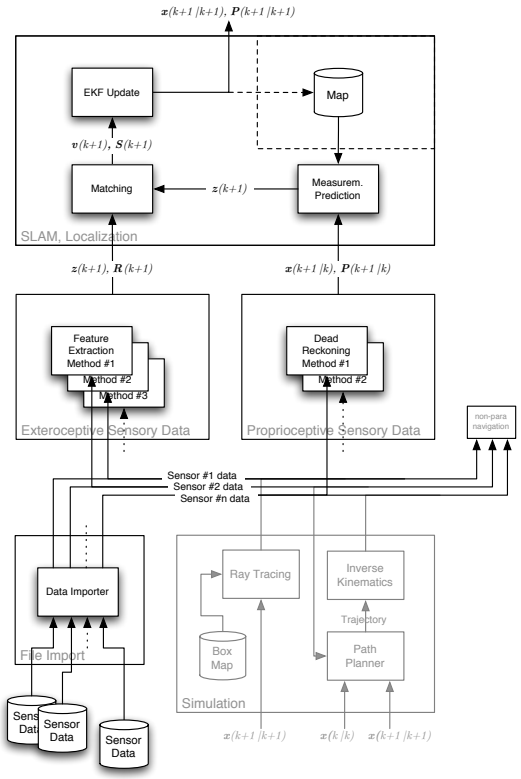


Figure 1: The three layered structure of the toolbox: the data generating layer (bottom), the feature extraction and dead reckoning layer (middle) and the localization and slam layer (top)

3.2 Initializing the Matlab Session

After decompression and unpacking, put the directory `toolbox` at a place of your choice. As long as the Matlab path is properly set, it does not matter where. Start up Matlab from `toolbox` or set `toolbox` as the current working directory. The current directory window of Matlab should look like figure 2. Then execute the initialization script `inittoolbox.m` by typing `inittoolbox` (without the `.m` extension). The script adds all toolbox-related paths and reads in a lookup table. This has to be done only once per Matlab session.

The subdirectories in `toolbox` reflect the structure of the toolbox. All m-files related to data import are in the folder `import` and e.g. all m-files related to slam are in the `slam` directory. Feature extraction has the folder `featureextr` which contains subdirectories for each extraction method. Class directories in Matlab start with a `@`. They contain all methods of that class.

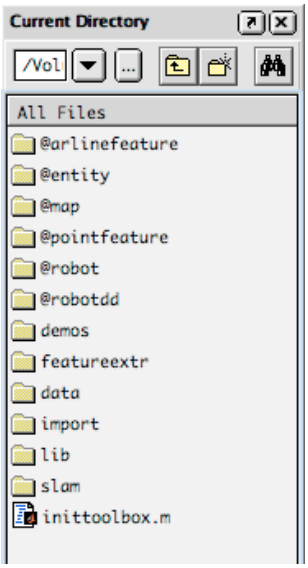


Figure 2: The working directory

3.3 Obtaining Help and Information on Objects

If the command window is cleared and the initialization script terminates by `inittoolbox: Paths added, lookup table in memory`, you are ready to go. Remember that you can always type `help filename` (without the `.m` extension) to obtain the help text of a file and `help directoryname` to get the first lines of all m-files in a directory. As method names can be ambiguous, they must be called together with their class name using a slash, i.e. `help map/set` or `help robotdd/draw`.

Maps, features and robots are implemented as objects. They have at least a constructor method, a set, get and a display method. If you have an object `obj`, just type `obj` without semicolon to invoke the display method and see the contents of the object. The same information in a different format is obtained by typing `get(obj)`. The call `set(obj)` displays all property names and their possible values for the object `obj`. In order to access object properties, use the set and get methods.

Example:
Let us create and draw a map with a robot, a point feature, and an α, r-line feature. We first call the constructor method of the map object. The method takes a name and a timestamp.

```
G = map('global map',0);
```

Then we create a differential drive robot and add it to the map using the `addentity` method. The robot construction method takes a name, an identifier, a x, y, θ-pose, a 3×3 pose covariance matrix, the radii of the left and right wheel, the wheel base (all in meters) and a timestamp.

```
r = robotdd('Piggy',[1;2;pi/6],0.001*eye(3),0.2,0.2,0.5,4,0);
G = addentity(G,r);
```

Now type `G` without semicolon to see the contents of the map (which is merely the robot). Or type `r` to see the properties of the robot. Let us create the point feature and add it to the map. The constructor method takes an identifier, an x, y-position and a 2×2 position covariance matrix.

```
p = pointfeature(101,[-2;2],[0.01 0.002; 0.002 0.03]);
G = addentity(G,p)
```

where we left the semicolon away to see the new `G` immediately. Creating and adding the line feature is similar. Again, we specify an identifier, an α, r position and the associated covariance matrix

```
l = arlinefeature(201,[-pi/3;2],0.001*eye(2));
G = addentity(G,l)
```

Handling objects in this way is a Matlab-wide standard. You might have encountered this when editing Matlab figures using set/get.

Finally we want to plot the map using its `draw` method. As we are not sure about the arguments, we type `help map/draw`. Let us decide to draw a blue map including identifiers and uncertainty ellipses into the prepared figure window number one. As the map contains an infinite line, the result might be somewhat unpredictable and we choose to limit the window's axes to ± 3 m:

```
figure(1); clf; hold on; axis equal;
draw(G,1,1,0,'b');
axis([-3 3 -3 3]);
```

The result is shown in figure 3.

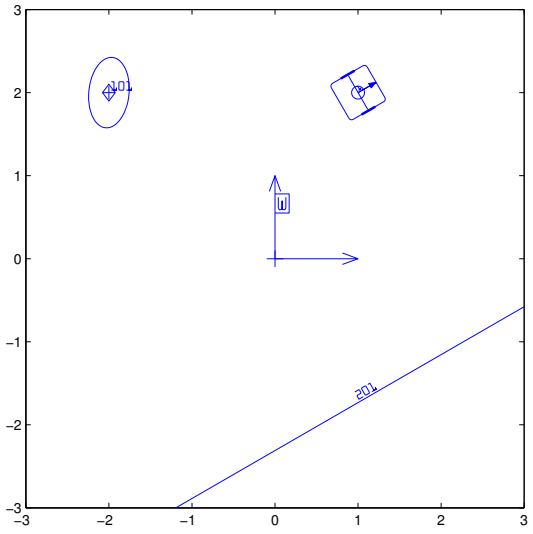


Figure 3: The resulting map