Simulataneous localization and mapping with the extended Kalman filter

'A very quick guide... with Matlab code!'

Joan Solà

October 5, 2014

Contents

1 Simultaneous Localization and Mapping (SLAM) 2
1.1 Introduction . . . 2
1.2 Notes for the absolute beginners . . . 2
1.3 SLAM entities: map, robot, sensor, landmarks, observations, estimator. . . 4
1.3.1 Entities and their relationship . . . 4
1.3.2 Class structure in RTSLAM . . . 4
1.4 Motion and observation models . . . 5
1.4.1 Motion model . . . 5
1.4.2 Direct observation model . . . 5
1.4.3 Inverse observation model . . . 5

2 EKF-SLAM 6
2.1 Setting up an EKF for SLAM . . . 6
2.2 The map . . . 7
2.3 Operations of EKF-SLAM . . . 7
2.3.1 Map initialization . . . 7
2.3.2 Robot motion . . . 7
2.3.3 Observation of mapped landmarks . . . 9
2.3.4 Landmark initialization for full observations . . . 10
2.3.5 Landmark initialization for partial observations . . . 11
2.3.6 Partial landmark initialization from bearing-only measurements . . 13
2.4 Chaining the events . . . 14

A Geometry 16
A.1 Rotation matrix . . . 16
A.2 Reference frames . . . 16
A.3 Motion of a body in the plane . . . 16

A.4 Polar coordinates . . . 17
A.5 Useful combinations . . . 17

B Probability **18**
B.1 Generalities . . . 18
B.1.1 Probability density function . . . 18
B.1.2 Expectation operator . . . 18
B.1.3 Very useful examples . . . 18
B.2 Gaussian variables . . . 18
B.2.1 Introduction and definitions . . . 18
B.2.2 Linear propagation . . . 18
B.2.3 Nonlinear propagation and linear approximation . . . 19
B.3 Graphical representation . . . 19
B.3.1 The Mahalanobis distance and the n-sigma ellipsoid . . . 20
B.3.2 MATLAB examples . . . 21

C Matlab code **22**
C.1 Elementary geometric functions . . . 22
C.1.1 Frame transformations . . . 22
C.1.2 Project to sensor . . . 24
C.1.3 Back project from sensor . . . 25
C.2 SLAM level operations . . . 26
C.2.1 Robot motion . . . 26
C.2.2 Direct observation model . . . 27
C.2.3 Inverse observation model . . . 27
C.3 EKF-SLAM code . . . 28

1 Simultaneous Localization and Mapping (SLAM)

1.1 Introduction

Simultaneous localization and mapping (SLAM) is the problem of concurrently estimating in real time the structure of the surrounding world (the map), perceived by moving exteroceptive sensors, while simultaneously getting localized in it. The seminal solution to the problem by Smith and Cheeseman (1987) [2] employs an extended Kalman filter (EKF) as the central estimator, and has been used extensively.

This file is an accompanying document for a SLAM course I give at ISAE in Toulouse every winter. Please find all the Matlab code generated during the course at the end of this document.

1.2 Notes for the absolute beginners

SLAM is a simple and everyday problem: the problem of spatial exploration. You enter an unknown space, you observe it, you move inside it; you build a spatial model of it,

and you know where in this model you are located. Then, you can plan how to reach this or that part of the space, how to leave it, etc. It is truly an everyday problem. I mean, you do it all the time without even noticing it. And each time you get disoriented, confused or lost is because you did not do it right. Yet its solution, if it wants to be automated and executed by a robot, is complex and tricky, and many naive approaches to solve it literally fail.

SLAM involves a moving agent (for example a robot), which embarks at least one sensor able to gather information about its surroundings (a camera, a laser scanner, a sonar: these are called *exteroceptive sensors*). Optionally, the moving agent can incorporate other sensors to measure its own movement (wheel encoders, accelerometers, gyrometers: these are known as *proprioceptive sensors*). The minimal SLAM system consists of one moving exteroceptive sensor (for example, a camera in your hand) connected to a computer. Thus, it can be all included in your smartphone.

SLAM consists of three basic operations, which are reiterated at each time step:

The robot moves, reaching a new point of view of the scene. Due to unavoidable noise and errors, this motion **increases the uncertainty on the robot's localization**. An automated solution requires a mathematical model for this motion. We call this the *motion model*.

The robot discovers interesting features in the environment, which need to be incorporated to the map. We call these features *landmarks*. Because of errors in the exteroceptive sensors, the location of these landmarks will be uncertain. Moreover, as the robot location is already uncertain, these two uncertainties need to be properly composed. An automated solution requires a mathematical model to determine the position of the landmarks in the scene from the data obtained by the sensors. We call this the *inverse observation model*.

The robot observes landmarks that had been previously mapped, and uses them to correct both its self-localization and the localization of all landmarks in space. In this case, therefore, both **localization and landmarks uncertainties decrease**. An automated solution requires a mathematical model to predict the values of the measurement from the predicted landmark location and the robot localization. We call this the *direct observation model*.

With these three models plus an estimator engine we are able to build an automated solution to SLAM. The estimator is responsible for the proper propagation of uncertainties each time one of the three situations above occur. In the case of this course, an extended Kalman filter (EKF) is used.

Other than that, a solution to SLAM needs to chain all these operations together and to keep all data healthy and organized, making the appropriate decisions at every step.

This document covers all these aspects.

1.3 SLAM entities: map, robot, sensor, landmarks, observations, estimator...

1.3.1 Entities and their relationship

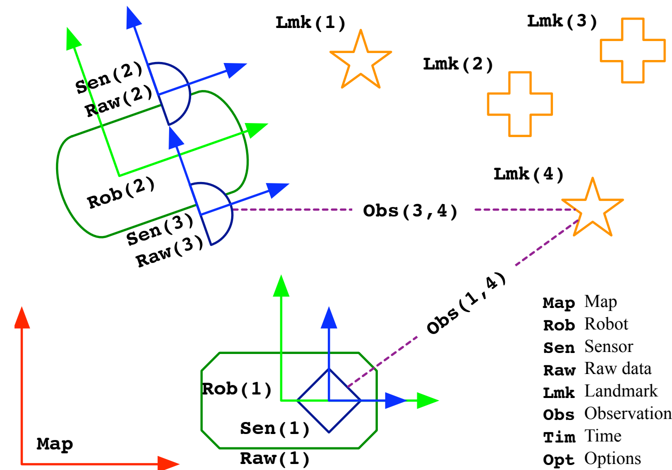

Figure 1: Typical SLAM entities

Fig. 1 is taken from the documentation of SLAMTB [3], a SLAM toolbox for Matlab that we built some years ago. In the figure we can see that

- The map has robots and landmarks.
- Robots have (exteroceptive) sensors.
- Each pair sensor-landmark defines an observation.

1.3.2 Class structure in RTSLAM

RTSLAM [1] is a C++ implementation of visual EKF-SLAM working in real-time at 60fps. Its structure of classes implements the scheme above, with the addition of two object managers, as follows,

In Fig. 2 we can see that

- The map has robots and landmarks. Landmarks are maintained by *map managers* owned by the map.
- Robots have (exteroceptive) sensors.
- Each pair sensor-landmark defines an observation. Observations are managed by the sensor with a *data manager*.

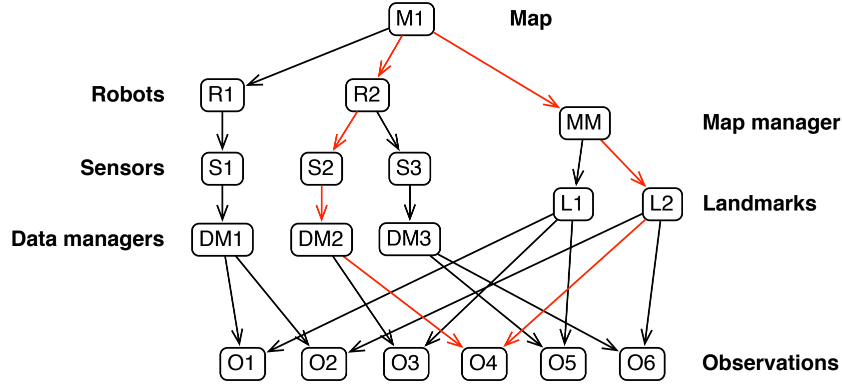

Figure 2: Ownship of classes in an object-oriented EKF-SLAM implementation

1.4 Motion and observation models

1.4.1 Motion model

The robot $\mathcal{R}$ moves according to a control signal $\mathbf{u}$ and a perturbation $\mathbf{n}$ and updates its state,

$$\mathcal{R} \leftarrow f(\mathcal{R}, \mathbf{u}, \mathbf{n}) \tag{1}$$

The control signal is often the data from the proprioceptive sensors. It can also be the control data sent by the computer to the robot's wheels. And it can also be void, in case the motion model does not take any control input.

See App. A.3 for an example of motion model.

See App. C.2.1 for a Matlab implementation.

1.4.2 Direct observation model

The robot $\mathcal{R}$ observes a landmark $\mathcal{L}_i$ that was already mapped by means of one of its sensors $\mathcal{S}$. It obtains a measurement $\mathbf{y}_i$,

$$\mathbf{y}_i = h(\mathcal{R}, \mathcal{S}, \mathcal{L}_i) \tag{2}$$

See App. A.5, Eq. (63) for an example of direct observation model.

See App. C.2.2 for a Matlab implementation.

1.4.3 Inverse observation model

The robot computes the state of a newly discovered landmark,

$$\mathcal{L}_j = g(\mathcal{R}, \mathcal{S}, \mathbf{y}_j) \tag{3}$$

See App. A.5, Eq. (64) for an example of inverse observation model.

See App. C.2.3 for a Matlab implementation.

Ideally, the function $g()$ is the inverse of $h()$ with respect to the measurement. In cases where the measurement is rank-deficient (that is, the measurement does not contain information on all the DOF of the landmark's state), $h()$ is not invertible and $g()$ cannot be defined. This happens in *e.g.* monocular vision, where the images do not contain the distances to the perceived objects. The parameter $\mathbf{s}$ is then introduced as a *prior* of the lacking DOF in order to render $g()$ definible,

$$\mathcal{L}_j = g(\mathcal{R}, \mathcal{S}, \mathbf{y}_j, \mathbf{s}) \tag{4}$$

2 EKF-SLAM

2.1 Setting up an EKF for SLAM

In EKF-SLAM, the map is a large vector stacking sensors and landmarks states, and it is modeled by a Gaussian variable. This map, usually called the stochastic map, is maintained by the EKF through the processes of prediction (the sensors move) and correction (the sensors observe the landmarks in the environment that had been previously mapped).

In order to achieve true exploration, the EKF machinery is enriched with an extra step of landmark initialization, where newly discovered landmarks are added to the map. Landmark initialization is performed by inverting the observation function and using it and its Jacobians to compute, from the sensor pose and the measurements, the observed landmark state and its necessary co- and cross-variances with the rest of the map. These relations are then appended to the state vector and the covariances matrix.

The following table resumes the similarities and differences between EKF and EKF-SLAM.

Table 1: EKF operations for achieving SLAM

Event	SLAM	EKF
Robot moves	Robot motion	EKF prediction
Sensor detects new landmark	Landmark initialization	State augmentation
Sensor observes known landmark	Map correction	EKF correction
Mapped landmark is corrupted	Landmark deletion	State reduction

2.2 The map

The map is a large state vector stacking robot and landmark states,[1]

$$\mathbf{x} = \begin{bmatrix} \mathcal{R} \\ \mathcal{M} \end{bmatrix} = \begin{bmatrix} \mathcal{R} \\ \mathcal{L}_1 \\ \vdots \\ \mathcal{L}_n \end{bmatrix} \tag{5}$$

where $\mathcal{R}$ is the robot state and $\mathcal{M} = (\mathcal{L}_1, \cdots, \mathcal{L}_n)$ is the set of landmark states, with n the current number of mapped landmarks.

In EKF, this map is modeled by a Gaussian variable using the mean and the covariances matrix of the state vector, denoted respectively by $\overline{\mathbf{x}}$ and $\mathbf{P}$,

$$\overline{\mathbf{x}} = \begin{bmatrix} \overline{\mathcal{R}} \\ \overline{\mathcal{M}} \end{bmatrix} = \begin{bmatrix} \overline{\mathcal{R}} \\ \overline{\mathcal{L}}_1 \\ \vdots \\ \overline{\mathcal{L}}_n \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} \mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{M}} \\ \mathbf{P}_{\mathcal{M}\mathcal{R}} & \mathbf{P}_{\mathcal{M}\mathcal{M}} \end{bmatrix} = \begin{bmatrix} \mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{L}_1} & \cdots & \mathbf{P}_{\mathcal{R}\mathcal{L}_n} \\ \mathbf{P}_{\mathcal{L}_1\mathcal{R}} & \mathbf{P}_{\mathcal{L}_1\mathcal{L}_1} & \cdots & \mathbf{P}_{\mathcal{L}_1\mathcal{L}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{P}_{\mathcal{L}_n\mathcal{R}} & \mathbf{P}_{\mathcal{L}_n\mathcal{L}_1} & \cdots & \mathbf{P}_{\mathcal{L}_n\mathcal{L}_n} \end{bmatrix} \tag{6}$$

The goal of EKF-SLAM, therefore, is to keep the map $\{\overline{\mathbf{x}}, \mathbf{P}\}$ up to date at all times.

2.3 Operations of EKF-SLAM

2.3.1 Map initialization

The map starts with no landmarks, therefore $n = 0$ and $\mathbf{x} = \mathcal{R}$. Also, the initial robot pose is usually considered the origin of the map that is going to be constructed, with absolute certainty (or absolutely no uncertainty!). Therefore,

$$\overline{\mathbf{x}} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{7}$$

2.3.2 Robot motion

In regular EKF, if $\mathbf{x}$ is our state vector, $\mathbf{u}$ is the control vector and $\mathbf{n}$ is the perturbation vector, then we have the generic time-update function

$$\mathbf{x} \leftarrow f(\mathbf{x}, \mathbf{u}, \mathbf{n}) \tag{8}$$

The EKF prediction step is classically writen as

$$\overline{\mathbf{x}} \leftarrow f(\overline{\mathbf{x}}, \mathbf{u}, 0) \tag{9}$$

$$\mathbf{P} \leftarrow \mathbf{F_x} \mathbf{P} \mathbf{F_x}^\top + \mathbf{F_n} \mathbf{N} \mathbf{F_n}^\top \tag{10}$$

[1] The sensor state $\mathcal{S}$ appearing in the observation models is usually not part of the map because it is constituted of known constant parameters.

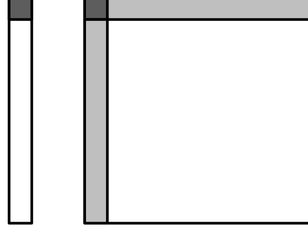

Figure 3: Updated parts of the map upon robot motion. The mean is represented by the bar on the left. The covariances matrix by the square on the right. The updated parts, in gray, correspond to the robot's state mean $\overline{\mathcal{R}}$ and covariance $\mathbf{P}_{\mathcal{R}\mathcal{R}}$ (dark gray), and the cross-variances $\mathbf{P}_{\mathcal{R}\mathcal{M}}$ and $\mathbf{P}_{\mathcal{M}\mathcal{R}}$ between the robot and the rest of the map (pale gray).

with the Jacobian matrices $\mathbf{F_x} = \frac{\partial f(\overline{\mathbf{x}},\mathbf{u})}{\partial \mathbf{x}}$ and $\mathbf{F_n} = \frac{\partial f(\overline{\mathbf{x}},\mathbf{u})}{\partial \mathbf{n}}$, and where $\mathbf{N}$ is the covariances matrix of the perturbation $\mathbf{n}$.

In SLAM, only a part of the state is time-variant: the robot, which moves. Therefore we have a different behavior for each part of the state vector,

$$\mathcal{R} \leftarrow f_{\mathcal{R}}(\mathcal{R}, \mathbf{u}, \mathbf{n}) \tag{11}$$

$$\mathcal{M} \leftarrow \mathcal{M} \tag{12}$$

where the first equation is precisely the *motion model*. Then, because the largest part of the map is invariant upon robot motion, we have sparse Jacobian matrices with the following structure,

$$\mathbf{F_x} = \begin{bmatrix} \frac{\partial f_{\mathcal{R}}}{\partial \mathcal{R}} & 0 \\ 0 & \mathbf{I} \end{bmatrix} \qquad \mathbf{F_n} = \begin{bmatrix} \frac{\partial f_{\mathcal{R}}}{\partial \mathbf{n}} \\ 0 \end{bmatrix} \tag{13}$$

If we avoid performing in (10) all trivial operations such as 'multiply by one', 'multiply by zero' and 'add zero', we obtain the EKF sparse prediction equations which are used for robot motion (see Fig. 3),

$$\overline{\mathcal{R}} \leftarrow f_{\mathcal{R}}(\overline{\mathcal{R}}, \mathbf{u}, 0) \tag{14}$$

$$\mathbf{P}_{\mathcal{R}\mathcal{R}} \leftarrow \frac{\partial f_{\mathcal{R}}}{\partial \mathcal{R}} \mathbf{P}_{\mathcal{R}\mathcal{R}} \frac{\partial f_{\mathcal{R}}}{\partial \mathcal{R}}^\top + \frac{\partial f_{\mathcal{R}}}{\partial \mathbf{n}} \mathbf{N} \frac{\partial f_{\mathcal{R}}}{\partial \mathbf{n}}^\top \tag{15}$$

$$\mathbf{P}_{\mathcal{R}\mathcal{M}} \leftarrow \frac{\partial f_{\mathcal{R}}}{\partial \mathcal{R}} \mathbf{P}_{\mathcal{R}\mathcal{M}} \tag{16}$$

$$\mathbf{P}_{\mathcal{M}\mathcal{R}} \leftarrow \mathbf{P}_{\mathcal{R}\mathcal{M}}^\top \tag{17}$$

Moreover, if we take care to store the covariances matrix as a triangular matrix, which is possible because it is symmetric, the operation (17) does not need to be performed.

The algorithmic complexity of this set of equations is $O(n)$ due to (16).

2.3.3 Observation of mapped landmarks

Similarly, we have in EKF the generic observation function

$$\mathbf{y} = h(\mathbf{x}) + \mathbf{v} \tag{18}$$

where $\mathbf{y}$ is the noisy measurement, $\mathbf{x}$ is the full state, $h()$ is the observation function and $\mathbf{v}$ is the measurement noise.

The EKF correction step is classically written as

$$\bar{\mathbf{z}} = \mathbf{y} - h(\bar{\mathbf{x}}) \tag{19}$$

$$\mathbf{Z} = \mathbf{H}_{\mathbf{x}} \mathbf{P} \mathbf{H}_{\mathbf{x}}^{\top} + \mathbf{R} \tag{20}$$

$$\mathbf{K} = \mathbf{P} \mathbf{H}_{\mathbf{x}}^{\top} \mathbf{Z}^{-1} \tag{21}$$

$$\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \mathbf{K} \bar{\mathbf{z}} \tag{22}$$

$$\mathbf{P} \leftarrow \mathbf{P} - \mathbf{K} \mathbf{Z} \mathbf{K}^{\top} \tag{23}$$

with the Jacobian $\mathbf{H}_{\mathbf{x}} = \frac{\partial h(\bar{\mathbf{x}})}{\partial \mathbf{x}}$ and where $\mathbf{R}$ is the covariances matrix of the measurement noise. In these equations, the first two are the innovation's mean and covariances matrix $\{\bar{\mathbf{z}}; \mathbf{Z}\}$; the third one is the Kalman gain $\mathbf{K}$; and the last two constitute the filter update.

In SLAM, observations occur when a measure of a particular landmark is taken by any of the robot's embarked sensors. This usually requires a more or less significant amount of raw data processing (especially in vision and other high-bandwidth sensors), which is obviated by now. The outcome of this processing is a geometric parametrization of the landmark in the measurement space: the vector $\mathbf{y}_i$.

For example, in vision, a measurement of a point landmark corresponds to the coordinates of the pixel where this landmark is projected in the image: $\mathbf{y}_i = (u_i, v_i)$.

Landmark observations are processed in the EKF usually one-by-one. The observation depends only on the robot position or state $\mathcal{R}$, the sensor state $\mathcal{S}$ and the particular landmark's state $\mathcal{L}_i$. Assuming that landmark i is observed, we have the individual observation function (the *observation model*)

$$\mathbf{y}_i = h_i(\mathcal{R}, \mathcal{S}, \mathcal{L}_i) + \mathbf{v} \tag{24}$$

which does not depend on any other landmark than $\mathcal{L}_i$. Therefore the structure of the Jacobian $\mathbf{H}_{\mathbf{x}}$ in EKF-SLAM is also sparse,

$$\mathbf{H}_{\mathbf{x}} = \begin{bmatrix} \mathbf{H}_{\mathcal{R}} & 0 & \cdots & 0 & \mathbf{H}_{\mathcal{L}_i} & 0 & \cdots & 0 \end{bmatrix} \tag{25}$$

with $\mathbf{H}_{\mathcal{R}} = \frac{\partial h_i(\overline{\mathcal{R}}, \mathcal{S}, \overline{\mathcal{L}}_i)}{\partial \mathcal{R}}$ and $\mathbf{H}_{\mathcal{L}_i} = \frac{\partial h_i(\overline{\mathcal{R}}, \mathcal{S}, \overline{\mathcal{L}}_i)}{\partial \mathcal{L}_i}$. Thanks to this sparsity the equations (20) and (21) can be reduced to only the products involving the non-zero elements (see Fig.

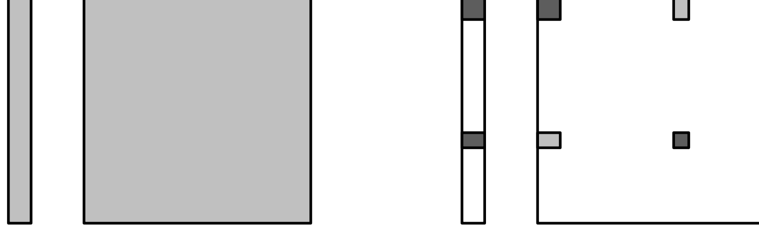

Figure 4: *Left:* Updated parts of the map upon landmark observation. The updated parts (in gray) correspond to the full map because the Kalman gain matrix $\mathbf{K}$ affects the full state. *Right:* However, the computation of the innovation (26, 27) is sparse: it only involves (in dark gray) the robot state $\overline{\mathcal{R}}$, the concerned landmark state $\overline{\mathcal{L}}_i$ and their covariances $\mathbf{P}_{\mathcal{R}\mathcal{R}}$ and $\mathbf{P}_{\mathcal{L}_i\mathcal{L}_i}$, and (in pale gray) their cross-variances $\mathbf{P}_{\mathcal{R}\mathcal{L}_i}$ and $\mathbf{P}_{\mathcal{L}_i\mathcal{R}}$.

4), and the set of correction equations becomes

$$\bar{\mathbf{z}} = \mathbf{y}_i - h_i(\overline{\mathcal{R}}, \mathcal{S}, \overline{\mathcal{L}}_i) \tag{26}$$

$$\mathbf{Z} = \begin{bmatrix}\mathbf{H}_{\mathcal{R}} & \mathbf{H}_{\mathcal{L}_i}\end{bmatrix} \begin{bmatrix}\mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{L}_i} \\ \mathbf{P}_{\mathcal{L}_i\mathcal{R}} & \mathbf{P}_{\mathcal{L}_i\mathcal{L}_i}\end{bmatrix} \begin{bmatrix}\mathbf{H}_{\mathcal{R}}^\top \\ \mathbf{H}_{\mathcal{L}_i}^\top\end{bmatrix} + \mathbf{R} \tag{27}$$

$$\mathbf{K} = \begin{bmatrix}\mathbf{P}_{\mathcal{R}\mathcal{R}} & \mathbf{P}_{\mathcal{R}\mathcal{L}_i} \\ \mathbf{P}_{\mathcal{M}\mathcal{R}} & \mathbf{P}_{\mathcal{M}\mathcal{L}_i}\end{bmatrix} \begin{bmatrix}\mathbf{H}_{\mathcal{R}}^\top \\ \mathbf{H}_{\mathcal{L}_i}^\top\end{bmatrix} \mathbf{Z}^{-1} \tag{28}$$

$$\bar{\mathbf{x}} \leftarrow \bar{\mathbf{x}} + \mathbf{K}\bar{\mathbf{z}} \tag{29}$$

$$\mathbf{P} \leftarrow \mathbf{P} - \mathbf{K}\mathbf{Z}\mathbf{K}^\top \tag{30}$$

The complexity of this set of equations is $O(n^2)$ due to (30). Such set of equations is applied each time a landmark is measured and updated. The total complexity for a total of k landmark updates is therefore $O(kn^2)$. It is worth noticing, for those who are used to standard EKF, that in our case the inversion of the innovation matrix $\mathbf{Z}$ is done in constant time $\mathcal{O}(1)$ (as opposed to cubic time $\mathcal{O}(n^3)$ in EKF!). Then, the Kalman gain $\mathbf{K}$ is computed in linear time $\mathcal{O}(n)$.

2.3.4 Landmark initialization for full observations

Landmark initialization happens when the robot discovers landmarks that are not yet mapped and decides to incorporate them in the map. As such, this operation results in an increase of the state vector's size. The EKF becomes then a filter of a state of dynamic size. This is why this operation is not usually known by users of regular EKF.

Landmark initialization is simple in cases where the sensor provides information about all the degrees of freedom of the new landmark. When this happens, we only need to invert the observation function $h()$ to compute the new landmark's state $\mathcal{L}_{n+1}$ from the robot state $\mathcal{R}$, the sensor state $\mathcal{S}$ and the observation $\mathbf{y}_{n+1}$,

$$\mathcal{L}_{n+1} = g(\mathcal{R}, \mathcal{S}, \mathbf{y}_{n+1}), \tag{31}$$

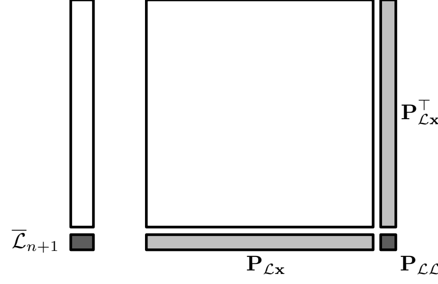

Figure 5: Appended parts to the map upon landmark initialization. The appended parts, in gray, correspond to the landmark's mean and covariance (dark gray), and the cross-variances between the landmark and the rest of the map (pale gray). Each gray block in the figure is identified with the results of equations (32) and (35–36), and the update is as shown in (37–38).

which constitutes the *inverse observation model* of one landmark.

We proceed as follows. First compute the landmark's mean and the function's Jacobians[2]

$$\overline{\mathcal{L}}_{n+1} = g(\overline{\mathcal{R}}, \mathcal{S}, \mathbf{y}_{n+1}) \tag{32}$$

$$\mathbf{G}_{\mathcal{R}} = \frac{\partial g(\overline{\mathcal{R}}, \mathcal{S}, \mathbf{y}_{n+1})}{\partial \mathcal{R}} \tag{33}$$

$$\mathbf{G}_{\mathbf{y}_{n+1}} = \frac{\partial g(\overline{\mathcal{R}}, \mathcal{S}, \mathbf{y}_{n+1})}{\partial \mathbf{y}_{n+1}} \tag{34}$$

Then compute the landmark's co-variance $\mathbf{P}_{\mathcal{L}\mathcal{L}}$, and its cross-variance with the rest of the map $\mathbf{P}_{\mathcal{L}\mathbf{x}}$,

$$\mathbf{P}_{\mathcal{L}\mathcal{L}} = \mathbf{G}_{\mathcal{R}}\mathbf{P}_{\mathcal{R}\mathcal{R}}\mathbf{G}_{\mathcal{R}}^\top + \mathbf{G}_{\mathbf{y}_{n+1}}\mathbf{R}\mathbf{G}_{\mathbf{y}_{n+1}}^\top \tag{35}$$

$$\mathbf{P}_{\mathcal{L}\mathbf{x}} = \mathbf{G}_{\mathcal{R}}\mathbf{P}_{\mathcal{R}\mathbf{x}} = \mathbf{G}_{\mathcal{R}}\left[\mathbf{P}_{\mathcal{R}\mathcal{R}}\ \mathbf{P}_{\mathcal{R}\mathcal{M}}\right] \tag{36}$$

Finally append these results to the state mean and covariances matrix (see Fig. 5)

$$\overline{\mathbf{x}} \leftarrow \begin{bmatrix} \overline{\mathbf{x}} \\ \overline{\mathcal{L}}_{n+1} \end{bmatrix} \tag{37}$$

$$\mathbf{P} \leftarrow \begin{bmatrix} \mathbf{P} & \mathbf{P}_{\mathcal{L}\mathbf{x}}^\top \\ \mathbf{P}_{\mathcal{L}\mathbf{x}} & \mathbf{P}_{\mathcal{L}\mathcal{L}} \end{bmatrix} \tag{38}$$

The complexity of this set of operations is $O(n)$.

2.3.5 Landmark initialization for partial observations

In cases where the sensor does not provide enough degrees of freedom for the function $h()$ to be invertible, we need to introduce this lacking information as a prior to the system

[2] See App. C.2.3 for a Matlab implementation.

[4]. This is the case when using bearing only sensors such as a monocular camera or range-only sensors such as a sonar.

With a prior, the inverse observation model is augmented to

$$\mathcal{L}_{n+1} = g(\mathcal{R}, \mathcal{S}, \mathbf{y}_{n+1}, \mathbf{s}) \tag{39}$$

where $\mathbf{s}$ is the prior. This prior is Gaussian with mean $\bar{\mathbf{s}}$ and covariances matrix $\mathbf{S}$.

From this on, the way to proceed is analogous to the fully observable case with just the addition of the prior term. First compute landmark's mean and all Jacobians

$$\overline{\mathcal{L}}_{n+1} = g(\overline{\mathcal{R}}, \mathcal{S}, \mathbf{y}_{n+1}, \bar{\mathbf{s}}) \tag{40}$$

$$\mathbf{G}_{\mathcal{R}} = \frac{\partial g(\overline{\mathcal{R}}, \mathcal{S}, \mathbf{y}_{n+1}, \bar{\mathbf{s}})}{\partial \mathcal{R}} \tag{41}$$

$$\mathbf{G}_{\mathbf{y}_{n+1}} = \frac{\partial g(\overline{\mathcal{R}}, \mathcal{S}, \mathbf{y}_{n+1}, \bar{\mathbf{s}})}{\partial \mathbf{y}_{n+1}} \tag{42}$$

$$\mathbf{G}_{\mathbf{s}} = \frac{\partial g(\overline{\mathcal{R}}, \mathcal{S}, \mathbf{y}_{n+1}, \bar{\mathbf{s}})}{\partial \mathbf{s}} \tag{43}$$

Then compute the landmark's co-variance and the cross-variance with the rest of the map

$$\mathbf{P}_{\mathcal{L}_{n+1}\mathcal{L}_{n+1}} = \mathbf{G}_{\mathcal{R}}\mathbf{P}_{\mathcal{R}\mathcal{R}}\mathbf{G}_{\mathcal{R}}^\top + \mathbf{G}_{\mathbf{y}_{n+1}}\mathbf{R}\mathbf{G}_{\mathbf{y}_{n+1}}^\top + \mathbf{G}_{\mathbf{s}}\mathbf{S}\mathbf{G}_{\mathbf{s}}^\top \tag{44}$$

$$\mathbf{P}_{\mathcal{L}\mathbf{x}} = \mathbf{G}_{\mathcal{R}}\mathbf{P}_{\mathcal{R}\mathbf{x}} = \mathbf{G}_{\mathcal{R}}\left[\mathbf{P}_{\mathcal{R}\mathcal{R}}\ \mathbf{P}_{\mathcal{R}\mathcal{M}}\right] \tag{45}$$

And finally augment the map as before.

Important note on Partial Landmark Initialization This trick of just introducing an invented prior seems trivial but it is not. Being $\mathbf{s}$ an unknown parameter, its associated covariance must ideally be infinite. And then this is what happens if we do not take important precautions:

1. EKF expects reasonable linearizations. This means that the function Jacobians must be valid approximations of the function derivatives inside all the probability concentration region (PCR) of the state variable.

2. If one DOF of our state has infinite uncertainty, it is required by the above rule that the functions manipulating it have a fairly linear behavior along the whole unbounded PCR of the prior. This is usually not the case.

3. As a consequence, setting up a naive system with a naive prior will most probably break the linearity condition and make EKF fail.

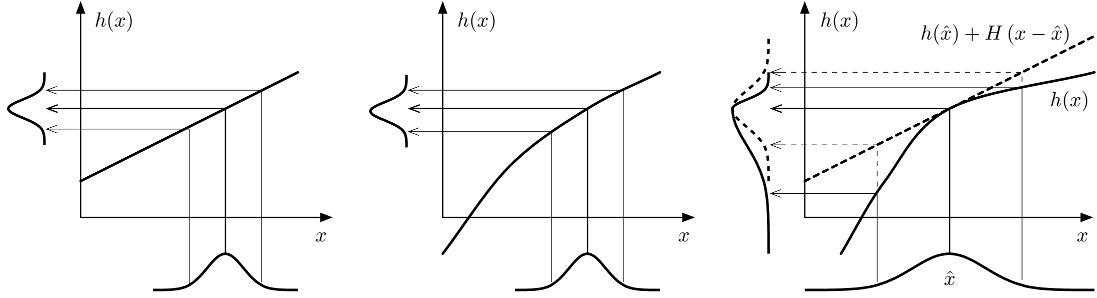

Figure 6: Linearization quality as a function of the probability concentration region (PCR). *Left*: linear case. *Center*: good linearization: the function is reasonably linear inside the PCR. *Right*: bad linearization: the derivatives vary very much within the PCR.

2.3.6 Partial landmark initialization from bearing-only measurements

The first key for a proper EKF performance is linearity. Linearity is defined as the opposite to non-linearity. Trivial. And non-linearity is defined as the change in the function derivatives inside the probability concentration region (PCR) of the input variables. Therefore non-linearity depends on both the function and the PCR (Fig. 6).

Then, if one of the input variables is completely unknown, its PCR is unbounded (it reaches the infinity) and the non-linearity should be small over this unbounded PCR.

In a bearing-only sensor such as a video camera, the unmeasured distance has an unbounded PCR, which reaches the infinity.

Because the observation function is nonlinear with respect to distance, we cannot assure a proper linearization inside the unbounded PCR. Can we do something about it? The answer is YES.

The first thing we can do is to define a new variable ρ as the inverse of the distance d

$$\rho \triangleq 1/d \tag{46}$$

and define the prior in this new variable. Assuming that the PCR of d spans from a certain minimum distance d_{min} to infinity, the PCR of ρ becomes bounded

$$d \in [d_{min}\ ,\ \infty] \quad \rightarrow \quad \rho \in [0\ ,\ 1/d_{min}] \tag{47}$$

We can define the (Cartesian) landmark position as a function of this new parameter, as follows

$$\mathbf{p} = \mathbf{p}_0 + 1/\rho \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix} \tag{48}$$

where $\mathbf{p}_0$ is the position of the sensor at the time of initialization, and α is an angle representing a direction in space. It is then convenient to parametrize the landmark as

follows

$$\mathcal{L} = \begin{bmatrix} \mathbf{p}_0 \\ \alpha \\ \rho \end{bmatrix} \in \mathbb{R}^4 \tag{49}$$

The power of such construction becomes visible when we express the bearing of a new measurement taken from another robot position $\mathbf{t}$. Consider the vector $\mathbf{v} = \mathbf{p} - \mathbf{t}$ corresponding to the new line of sight,

$$\mathbf{v} \triangleq \mathbf{p} - \mathbf{t} = (\mathbf{p}_0 - \mathbf{t}) + 1/\rho \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix}. \tag{50}$$

Because the sensor cannot observe distances, the measurement of this landmark is insensitive to the magnitude of (or norm of) $\mathbf{v}$. Therefore rescaling the expression above by a factor ρ yields

$$\mathbf{v} \propto \rho(\mathbf{p}_0 - \mathbf{t}) + \begin{bmatrix} \cos(\alpha) \\ \sin(\alpha) \end{bmatrix}, \tag{51}$$

which is **linear in** ρ.

> KEY FACT: *The observation function in homogeneous coordinates is linear in ρ, and the PCR is bounded in ρ. This means that EKF is probably going to work!*

The direct observation function is the bearing of $\mathbf{v}$,

$$\mathbf{y} = h(\mathcal{R}, \mathcal{L}) = \arctan(v_2/v_1) - \theta \tag{52}$$

The inverse observation function for this landmark parametrization, knowing the robot state $\mathcal{R} = [\mathbf{t}, \theta]^\top$ at initialization time and the bearing-only measurement $\mathbf{y} = \phi$, is simply

$$\mathcal{L} = g(\mathcal{R}, \mathbf{y}, \rho) = \begin{bmatrix} \mathbf{t} \\ \theta + \phi \\ \rho \end{bmatrix} \tag{53}$$

2.4 Chaining the events

A basic but functioning algorithm performing SLAM needs to chain all these operations in a meaningful way. The following pseudocode is a valuable example,

```
% INITIALIZATION
initialize_map()
time = 0

% TIME LOOP
while (execution() == true) do

    % LOOP ROBOTS
```

```
    for each robot in list_of_robots
        control = acquire_control_signal()
        move_robot(robot, control)

        % LOOP SENSORS IN EACH ROBOT
        for each sensor in robot->list_of_sensors
            raw = sensor->acquire_raw_data()

            % LOOP OBSERVATIONS IN EACH SENSOR
            for each observation in sensor->feasible_observations()

                % MEASURE LANDMARK AND CORRECT MAP
                measurement = find_known_feature(raw, observation)
                update_map(robot, sensor, landmark, observation, measurement)
            end

            % DISCOVER NEW LANDMARKS WITH THE CURRENT SENSOR
            measurement = detect_new_feature(raw)

            % INITIALIZE LANDMARK
            landmark = init_new_landmark(robot, sensor, measurement)
            create_new_observation(sensor, landmark)
        end
    end
    time ++
end
```

This course is devoted to expand this pseudo code to a full functional algorithm that implements a 2-dimensional SLAM system. The full code is collected in App. C.

Appendices

A Geometry

A.1 Rotation matrix

$$\mathsf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{54}$$

A.2 Reference frames

Let $\mathcal{W}$ be the World frame. Let $\mathcal{F}$ be a cartesian frame defined with respect to the world frame by a translation vector $\mathbf{t}$ and a rotation angle θ.

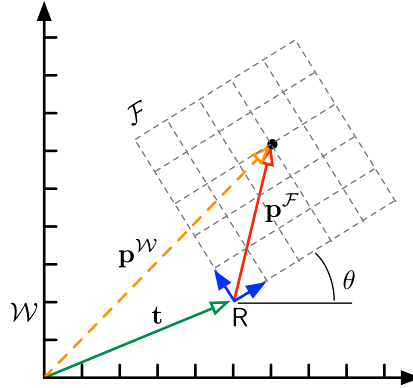

Figure 7: Frame transformation in the 2D plane. The two blue arrows are the two column vectors of the rotation matrix R, corresponding to the orientation θ of the local frame $\mathcal{F}$.

A point in space $\mathbf{p}$ can be expressed in World frame or in the local frame $\mathcal{F}$. Both expressions are related by the frame-transformation equations,

$$\mathbf{p}^{\mathcal{W}} = \mathsf{R}\mathbf{p}^{\mathcal{F}} + \mathbf{t} \quad \leftarrow \text{from frame } \mathcal{F} \tag{55}$$

$$\mathbf{p}^{\mathcal{F}} = \mathsf{R}^\top(\mathbf{p}^{\mathcal{W}} - \mathbf{t}) \leftarrow \text{to frame } \mathcal{F} \tag{56}$$

where R is the rotation matrix associated with the angle θ. The first expression is known as the '*from frame*' transformation. The second one is known as '*to frame*'.

A.3 Motion of a body in the plane

Let a Robot move in the plane. Let its state be the position and orientation, defined by

$$\mathcal{R} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} \mathbf{t} \\ \theta \end{bmatrix} \tag{57}$$

Let this robot receive a control signal $\mathbf{u}$ in the form of a vector specifying linear and angular pose increments,

$$\mathbf{u} = \begin{bmatrix} \delta x \\ \delta y \\ \delta\theta \end{bmatrix} = \begin{bmatrix} \delta\mathbf{t} \\ \delta\theta \end{bmatrix} \tag{58}$$

After the motion step, the robot state $\mathcal{R}$ is updated according to

$$\mathbf{t} \leftarrow \mathsf{R}\delta\mathbf{t} + \mathbf{t} \tag{59}$$

$$\theta \leftarrow \theta + \delta\theta \tag{60}$$

where the first equation corresponds to a '*from frame*' transform, while the second one is trivial.

A.4 Polar coordinates

Let a point in the plane be expressed by its two cartesian coordinates, $\mathbf{p} = [x, y]^\top$. Its polar representation is

$$\widehat{\mathbf{p}} = \begin{bmatrix} \rho \\ \phi \end{bmatrix} = \text{polar}(\mathbf{p}) = \begin{bmatrix} \sqrt{x^2 + y^2} \\ \arctan(y, x) \end{bmatrix}. \tag{61}$$

This operation can be inverted as follows

$$\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix} = \text{rectangular}(\widehat{\mathbf{p}}) = \begin{bmatrix} \rho\cos\phi \\ \rho\sin\phi \end{bmatrix}. \tag{62}$$

A.5 Useful combinations

Suitable combinations of frame transforms and polar transforms are very handy. Many onboard sensors used in mobile robotics such as laser rangers, sonars, video cameras, etc., provide information of the external landmarks in the form of range and/or bearing with respect to the local sensor frame. Range-only sensors perform only the first row of (61). Bearing-only sensors perform only the second row. Range-and-bearing sensors perform both rows.

Let the point $\mathbf{p}$ above be expressed in World frame. The polar representation of this point in frame $\mathcal{F}$ is obtained by composing a '*to frame*' transformation with the polar transform above,

$$\widehat{\mathbf{p}}^{\mathcal{F}} = \text{polar}(\text{toFrame}(\mathcal{F}, \mathbf{p}^{\mathcal{W}})). \tag{63}$$

The opposite situation requires composing the inverse functions in reversed order,

$$\mathbf{p}^{\mathcal{W}} = \text{fromFrame}(\mathcal{F}, \text{rectangular}(\widehat{\mathbf{p}}^{\mathcal{R}})) \tag{64}$$

Equations (63) and (64) are used as the direct and inverse range-and-bearing observation functions in SLAM. We can call **`observe()`** the first function, in which the robot is obtaining a range-and-bearing measurement of a point, and **`invObserve()`** the second one, with the operation of obtaining a point from a range-and-bearing measurement. See Appendices C.2.2 and C.2.3 for their Matlab implementation.

Table 2: Range and/or bearing information provided by popular sensors

sensor	range	bearing
Laser range finder	YES	YES
Sonar	YES	poor
Camera	NO	YES
RGBD (*e.g.* Kinect)	YES	YES
ARVA	poor	poor
RFID antenna	poor	poor

B Probability

B.1 Generalities

B.1.1 Probability density function

$$p_X(\mathbf{x}) \triangleq \lim_{d\mathbf{x}\to 0} \frac{P(\mathbf{x} \leq X < \mathbf{x} + d\mathbf{x})}{d\mathbf{x}} \tag{65}$$

B.1.2 Expectation operator

$$\mathbb{E}[f(\mathbf{x})] \triangleq \int_{-\infty}^{\infty} f(\mathbf{x})p(\mathbf{x})d\mathbf{x} \tag{66}$$

B.1.3 Very useful examples

Mean and covariances matrix

$$\overline{\mathbf{x}} = \mathbb{E}[\mathbf{x}] \tag{67}$$

$$\mathbf{P} = \mathbb{E}[(\mathbf{x} - \overline{\mathbf{x}})(\mathbf{x} - \overline{\mathbf{x}})^\top] \tag{68}$$

B.2 Gaussian variables

B.2.1 Introduction and definitions

$$\mathcal{N}(\mathbf{x}, \overline{\mathbf{x}}, \mathbf{P}) = \frac{1}{\sqrt{(2\pi)^n|\mathbf{P}|}} \exp\Big(-\frac{1}{2}(\mathbf{x} - \overline{\mathbf{x}})^\top \mathbf{P}^{-1}(\mathbf{x} - \overline{\mathbf{x}})\Big) \tag{69}$$

$$\tag{70}$$

B.2.2 Linear propagation

$$\mathbf{y} = \mathbf{F}\mathbf{x} \tag{71}$$

$$\overline{\mathbf{y}} = \mathbf{F}\overline{\mathbf{x}} \tag{72}$$

$$\mathbf{Y} = \mathbf{F}\mathbf{P}\mathbf{F}^\top \tag{73}$$

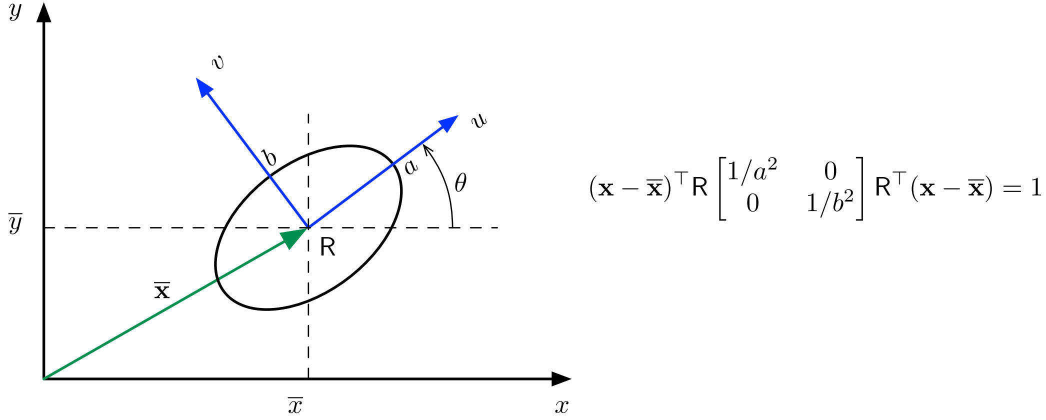

Figure 8: Ellipsoidal representation of multivariate Gaussian variables (2D). Ellipse dimensions, position and orientation are governed by the SVD decomposition of the covariances matrix $\mathbf{P}$.

B.2.3 Nonlinear propagation and linear approximation

We make use of the first-order Taylor approximation of the function, with $\mathbf{x}_0 = \overline{\mathbf{x}}$ as the linearization point,

$$f(\mathbf{x}) = f(\overline{\mathbf{x}}) + \mathbf{F_x}(\mathbf{x} - \overline{\mathbf{x}}) + O(\|\mathbf{x} - \overline{\mathbf{x}}\|^2) \tag{74}$$

with $\mathbf{F_x} = \frac{\partial f(\overline{\mathbf{x}})}{\partial \mathbf{x}}$ the Jacobian of $f(\mathbf{x})$ with respect to $\mathbf{x}$ around $\overline{\mathbf{x}}$. Then,

$$\mathbf{y} = f(\mathbf{x}) \tag{75}$$

$$\overline{\mathbf{y}} \approx f(\overline{\mathbf{x}}) \tag{76}$$

$$\mathbf{Y} \approx \mathbf{F_x} \mathbf{P} \mathbf{F_x}^\top \tag{77}$$

B.3 Graphical representation

From the multivariate Gaussian definition, we have that the part depending on $\mathbf{x}$ is only the exponent (at the right of the exp!). A curve of constant probability density can therefore be found as the locus of points $\mathbf{x}$ satisfying

$$(\mathbf{x} - \overline{\mathbf{x}})^\top \mathbf{P}^{-1} (\mathbf{x} - \overline{\mathbf{x}}) = const \tag{78}$$

When $const = 1$, this corresponds (see Fig. 8) to an ellipsoid centered at $\mathbf{x} = \overline{\mathbf{x}}$, with semiaxes oriented as the eigenvectors of $\mathbf{P}$ and of equal length as the square root of the singular values of $\mathbf{P}$. [3]

[3] **Proof:** Consider the ellipse in axes (u, v) in Fig. 8, with semiaxes a and b. Express it as $\frac{u^2}{a^2} + \frac{v^2}{b^2} = 1$. Write this expression in matrix form as

$$\mathbf{v}^\top \mathbf{D}^{-1} \mathbf{v} = 1 \tag{79}$$

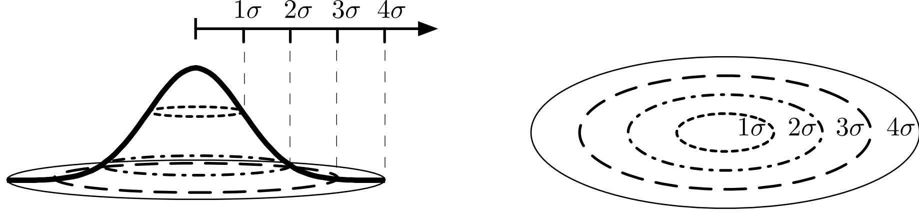

Figure 9: Ellipsoidal representation of multivariate Gaussian variables (2D). Different sigma-value ellipses can be defined for the same covariances matrix. The most useful ones are 2-sigma and 3-sigma.

Table 3: Percent probabilities of a random variable being inside its n-sigma ellipsoid.

	1σ	2σ	3σ	4σ
2D	$39,4\%$	$86,5\%$	$98,9\%$	$99,97\%$
3D	$19,9\%$	$73,9\%$	$97,1\%$	$99,89\%$

B.3.1 The Mahalanobis distance and the n-sigma ellipsoid

We can define the Mahalanobis distance as the normalized quadratic distance operator

$$MD(\mathbf{x}, \overline{\mathbf{x}}, \mathbf{P}) \triangleq \sqrt{(\mathbf{x}-\overline{\mathbf{x}})^\top \mathbf{P}^{-1}(\mathbf{x}-\overline{\mathbf{x}})} \tag{83}$$

With this, the ellipsoid above is the locus of points at a Mahalanobis distance of 1 from the point $\overline{\mathbf{x}}$.

We can also draw the ellipsoids at Mahalanobis distances other than one. These are called the n-sigma ellipsoids and are defined as a function of n by the locus $MD(\mathbf{x}, \overline{\mathbf{x}}, \mathbf{P}) = n$, or more explicitly

$$(\mathbf{x}-\overline{\mathbf{x}})^\top \mathbf{P}^{-1}(\mathbf{x}-\overline{\mathbf{x}}) = n^2 \tag{84}$$

In SLAM we make extensive use of the 2- and 3-sigma ellipsoids because they enclose probability concentrations of 97% to 99% (see Fig. 9 and Table 3).

---

with $\mathbf{v} = [u\ v]^\top$ and $\mathbf{D} = \operatorname{diag}(a^2, b^2)$. Then define the local reference frame $\mathbf{v}$ with respect to the global frame $\mathbf{x}$ by means of a rotation matrix R and a translation vector $\overline{\mathbf{x}}$. The following *to-frame* relation holds,

$$\mathbf{v} = \mathsf{R}^\top(\mathbf{x}-\overline{\mathbf{x}}). \tag{80}$$

Now inserting (80) into (79),

$$(\mathbf{x}-\overline{\mathbf{x}})^\top \mathsf{R}\mathbf{D}^{-1}\mathsf{R}^\top(\mathbf{x}-\overline{\mathbf{x}}) = 1 \tag{81}$$

and identifying terms in (78) we have that

$$\mathbf{P} = (\mathsf{R}\mathbf{D}^{-1}\mathsf{R}^\top)^{-1} = \mathsf{R}^{-\top}\mathbf{D}\mathsf{R}^{-1} = \mathsf{R}\mathbf{D}\mathsf{R}^\top \tag{82}$$

which corresponds to the singular value decomposition of $\mathbf{P}$.

Drawing the n-sigma ellipsoid is easy. Start by performing the SVD of $\mathbf{P}$,

$$[\mathsf{R}, \mathbf{D}] = \mathrm{svd}(\mathbf{P}) \tag{85}$$

Also, build the 2-by-$(n+1)$ matrix containing a set of points representing the unit circle in (u, v) axes,

$$\mathcal{C}_{uv} = \begin{bmatrix} u_0 & \cdots & u_n \\ v_0 & \cdots & v_n \end{bmatrix} \tag{86}$$

with $u_i = \cos(2\pi i/n)$ and $v_i = \sin(2\pi i/n)$, for $i = \{0, \ldots, n\}$. Then build the ellipse with semi axes na and nb, and rotate and translate it according to R and $\bar{\mathbf{x}} = [\bar{x}, \bar{y}]^\top$

$$\mathcal{E}_{xy} = n\mathsf{R}\sqrt{\mathbf{D}}\mathcal{C}_{uv} + \begin{bmatrix} \bar{x} & \cdots & \bar{x} \\ \bar{y} & \cdots & \bar{y} \end{bmatrix} \tag{87}$$

A plot of the contents of $\mathcal{E}_{xy}$ completes the drawing process.

B.3.2 MATLAB examples

To draw the 2-sigma ellipsoid of a 2D Gaussian with mean $\bar{\mathbf{x}}$ and covariances matrix $\mathbf{P}$, type the code

```
x = [1;2];                          % for example
P = [2 3;3 2];                       % for example
[xx,yy] = cov2elli(x, P, 3);         % 3-sigma ellipse's coordinates
plot(xx,yy);
axis equal
```

The key in this code is the function **cov2elli()** which returns two sets of coordinate values to be plotted as a line. This line draws the 2-sigma ellipse.

The code for **cov2elli** follows

```
function [X,Y] = cov2elli(x,P,n,NP)

% COV2ELLI  Ellipse contour from Gaussian mean and covariances matrix.
%   [X,Y] = COV2ELLI(X0,P) returns X and Y coordinates of the contour of
%   the 1-sigma ellipse of the Gaussian defined by mean X0 and covariances
%   matrix P. The contour is defined by 16 points, thus both X and Y are
%   16-vectors.
%
%   [X,Y] = COV2ELLI(X0,P,n,NP) returns the n-sigma ellipse and defines the
%   contour with NP points instead of the default 16 points.
%
%   The ellipse can be plotted in a 2D graphic by just creating a line
%   with 'line(X,Y)' or 'plot(X,Y)'.

%   Copyright 2008-2009 Joan Sola @ LAAS-CNRS.

if nargin < 4
```

```
    NP = 16;
    if nargin < 3
        n = 1;
    end
end

alpha  = 2*pi/NP*(0:NP);           % NP angle intervals for one turn
circle = [cos(alpha);sin(alpha)];  % the unit circle

% SVD method, P = R*D*R' = R*d*d*R'
[R,D]=svd(P);
d = sqrt(D);
% n-sigma ellipse <- rotated 1-sigma ellipse <- aligned 1-sigma ellipse <- unit circle
ellip = n * R * d * circle;

% output ready for plotting (X and Y line vectors)
X = x(1)+ellip(1,:);
Y = x(2)+ellip(2,:);
```

C Matlab code

Proof-ready functions to perform 2D EKF-SLAM with a range-and-bearing sensor are given below. They implement most of the material presented in this brief guide to EKF-SLAM. You can directly copy-paste them into your Matlab editor.

We differentiate between elementary function blocks and the functions SLAM really needs. The SLAM functions are often compositions of elementary functions.

All functions are able to return the Jacobian matrices of the output variables with respect to each one of the input variables.

Also, some of the functions here include a foot section with Matlab symbolic code for either constructing Jacobian matrices or testing if the function's code for the Jacobians is correct. To execute this code, put Matlab in cell mode and execute the foot cell.

Finally, we give a simple but sufficient example of a fully working SLAM algorithm, with simulation, estimation and graphics output. The program utilizes only 102 lines of code. With the functions it adds up to around 200 lines of code.

C.1 Elementary geometric functions

C.1.1 Frame transformations

Express a global point in a local frame:

```
function [pf, PF_f, PF_p] = toFrame(F , p)
%   TOFRAME transform point P from global frame to frame F
%
%   In:
%       F :     reference frame          F = [f_x ; f_y ; f_alpha]
%       p :     point in global frame    p = [p_x ; p_y]
```

```
%   Out:
%       pf:     point in frame F
%       PF_f:   Jacobian wrt F
%       PF_p:   Jacobian wrt p

%   (c) 2010, 2011, 2012 Joan Sola

t = F(1:2);
a = F(3);

R = [cos(a) -sin(a) ; sin(a) cos(a)];

pf = R' * (p - t);

if nargout > 1 %  Jacobians requested
    px = p(1);
    py = p(2);
    x = t(1);
    y = t(2);

    PF_f = [...
        [ -cos(a), -sin(a),   cos(a)*(py - y) - sin(a)*(px - x)]
        [  sin(a), -cos(a), - cos(a)*(px - x) - sin(a)*(py - y)]];

    PF_p = R';
end
end

function f()
%% Symbolic code below -- Generation and/or test of Jacobians
% - Enable 'cell mode' to use this section
% - Left-click once on the code below - the cell should turn yellow
% - Type ctrl+enter (Windows, Linux) or Cmd+enter (MacOSX) to execute
% - Check the Jacobian results in the Command Window.
syms x y a px py real
F = [x y a]';
p = [px py]';
pf = toFrame(F, p);
PF_f = jacobian(pf, F)
end
```

Express a local point in the global frame:

```
function [pw, PW_f, PW_pf] = fromFrame(F, pf)
%   FROMFRAME Transform a point PF from local frame F to the global frame.
%
%   In:
%       F :     reference frame     F  = [f_x ; f_y ; f_alpha]
%       pf:     point in frame F    pf = [pf_x ; pf_y]
%   Out:
%       pw:     point in global frame
%       PW_f:   Jacobian wrt F
```

```
%       PW_pf:  Jacobian wrt pf

%   (c) 2010, 2011, 2012 Joan Sola

t = F(1:2);
a = F(3);

R = [cos(a) -sin(a) ; sin(a) cos(a)];

pw = R*pf + repmat(t,1,size(pf,2));  % Allow for multiple points

if nargout > 1 %  Jacobians requested

    px = pf(1);
    py = pf(2);

    PW_f =  [...
        [ 1, 0, - py*cos(a) - px*sin(a)]
        [ 0, 1,   px*cos(a) - py*sin(a)]];

    PW_pf = R;

end
end

function f()
%% Symbolic code below -- Generation and/or test of Jacobians
% - Enable 'cell mode' to use this section
% - Left-click once on the code below - the cell should turn yellow
% - Type ctrl+enter (Windows, Linux) or Cmd+enter (MacOSX) to execute
% - Check the Jacobian results in the Command Window.
syms x y a px py real
F = [x;y;a];
pf = [px;py];
pw = fromFrame(F,pf);
PW_f = jacobian(pw,F)
PW_pf = jacobian(pw,pf)
end
```

C.1.2 Project to sensor

```
function [y, Y_p] = scan (p)
%   SCAN perform a range-and-bearing measure of a 2D point.
%
%   In:
%       p :     point in sensor frame   p = [p_x ; p_y]
%   Out:
%       y :     measurement             y = [range ; bearing]
%       Y_p:    Jacobian wrt p
```

```
%   (c) 2010, 2011, 2012 Joan Sola

px = p(1);
py = p(2);

d = sqrt(px^2+py^2);
a = atan2(py,px);
% a = atan(py/px);  % use this line if you are in symbolic mode.

y = [d;a];

if nargout > 1 %  Jacobians requested

    Y_p = [...
        px/sqrt(px^2+py^2)          , py/sqrt(px^2+py^2)
        -py/(px^2*(py^2/px^2 + 1)), 1/(px*(py^2/px^2 + 1)) ];

end
end

function f()
%% Symbolic code below -- Generation and/or test of Jacobians
% - Enable 'cell mode' to use this section
% - Left-click once on the code below - the cell should turn yellow
% - Type ctrl+enter (Windows, Linux) or Cmd+enter (MacOSX) to execute
% - Check the Jacobian results in the Command Window.
syms px py  real
p = [px;py];
y = scan(p);
Y_p = jacobian(y,p)
[y,Y_p] = scan(p);
simplify(Y_p - jacobian(y,p))
end
```

C.1.3 Back project from sensor

```
function [p, P_y] = invScan(y)
%   INVSCAN Backproject a range-and-bearing measure into a 2D point.
%
%   In:
%       y :     range-and-bearing measurement y = [range ; bearing]
%   Out:
%       p :     point in sensor frame   p = [p_x ; p_y]
%       P_y:    Jacobian wrt y

%   (c) 2010, 2011, 2012 Joan Sola

d = y(1);
a = y(2);
```

```
px = d*cos(a);
py = d*sin(a);

p = [px;py];

if nargout > 1 %  Jacobians requested

   P_y = [...
       cos(a) , -d*sin(a)
       sin(a) , d*cos(a)];

end
```

C.2 SLAM level operations

C.2.1 Robot motion

```
function [ro, RO_r, RO_n] = move(r, u, n)
%   MOVE Robot motion, with separated control and perturbation inputs.
%
%   In:
%       r: robot pose       r = [x ; y ; alpha]
%       u: control signal   u = [d_x ; d_alpha]
%       n: perturbation, additive to control signal
%   Out:
%       ro: updated robot pose
%       RO_r: Jacobian   d(ro) / d(r)
%       RO_n: Jacobian   d(ro) / d(n)

a = r(3);
dx = u(1) + n(1);
da = u(2) + n(2);

ao = a + da;

if ao > pi
    ao = ao - 2*pi;
end
if ao < -pi
    ao = ao + 2*pi;
end

% build position increment dp=[dx;dy], from control signal dx
dp = [dx;0];

if nargout == 1 % No Jacobians requested

    to = fromFrame(r, dp);

else %  Jacobians requested
```

```
    [to, TO_r, TO_dt] = fromFrame(r, dp);
    AO_a = 1;
    AO_da = 1;

    RO_r = [TO_r ; 0 0 AO_a];
    RO_n = [TO_dt(:,1) zeros(2,1) ; 0 AO_da];

end

ro = [to;ao];
```

C.2.2 Direct observation model

```
function [y, Y_r, Y_p] = observe(r, p)
%   OBSERVE Transform a point P to robot frame and take a
%   range-and-bearing measurement.
%
%   In:
%       r :     robot frame             r = [r_x ; r_y ; r_alpha]
%       p :     point in global frame   p = [p_x ; p_y]
%   Out:
%       y:      range-and-bearing measurement
%       Y_r:    Jacobian wrt r
%       Y_p:    Jacobian wrt p

%   (c) 2010, 2011, 2012 Joan Sola

if nargout == 1 % No Jacobians requested

    y = scan(toFrame(r,p));

else %  Jacobians requested

    [pr, PR_r, PR_p] = toFrame(r, p);
    [y, Y_pr] = scan(pr);

    % The chain rule!
    Y_r = Y_pr * PR_r;
    Y_p = Y_pr * PR_p;

end
```

C.2.3 Inverse observation model

```
function [p, P_r, P_y] = invObserve(r, y)
%   INVOBSERVE Backproject a range-and-bearing measurement and transform
%   to map frame.
```

```
%
%   In:
%       r :     robot frame     r = [r_x ; r_y ; r_alpha]
%       y :     measurement     y = [range ; bearing]
%   Out:
%       p :     point in sensor frame
%       P_r:    Jacobian wrt r
%       P_y:    Jacobian wrt y

%   (c) 2010, 2011, 2012 Joan Sola

if nargout == 1 % No Jacobians requested

    p   = fromFrame(r, invScan(y));

else %  Jacobians requested

    [p_r, PR_y]     = invScan(y);
    [p, P_r, P_pr] = fromFrame(r, p_r);

    % here the chain rule !
    P_y = P_pr * PR_y;

end
end

function f()
%% Symbolic code below -- Generation and/or test of Jacobians
% - Enable 'cell mode' to use this section
% - Left-click once on the code below - the cell should turn yellow
% - Type ctrl+enter (Windows, Linux) or Cmd+enter (MacOSX) to execute
% - Check the Jacobian results in the Command Window.
syms rx ry ra yd ya real
r = [rx;ry;ra];
y = [yd;ya];
[p, P_r, P_y] = invObserve(r, y); % We extract also the coded Jacobians P_r and P_y
% We use the symbolic result to test the coded Jacobians
simplify(P_r - jacobian(p,r))  % zero-matrix if coded Jacobian is correct
simplify(P_y - jacobian(p,y))  % zero-matrix if coded Jacobian is correct
end
```

C.3 EKF-SLAM code

It follows a 102-lines-of-code m-file performing SLAM. This code uses all the files above, plus the helper function **cloister.m** (also given below) which is just used to define the set of landmarks for the simulation.

Please read all help notes in this file carefully. They explain a number of important things not covered (because they relate to implementation issues) in the main body of this document.

```
% SLAM2D  A 2D EKF-SLAM algorithm with simulation and graphics.
%
%  HELP NOTES:
%  1. The robot state is defined by [xr;yr;ar] with [xr;yr] the position
%     and [ar] the orientation angle in the plane.
%  2. The landmark states are simply Li=[xi;yi]. There are a number of N
%     landmarks organized in a 2-by-N matrix W=[L1 L2 ... Ln]
%     so that Li = W(:,i).
%  3. The control signal for the robot is U=[dx;da] where [dx] is a forward
%     motion and [da] is the angle of rotation.
%  4. The motion perturbation is additive Gaussian noise n=[nx;na] with
%     covariance Q, which adds to the control signal.
%  5. The measurements are range-and-bearing Yi=[di;ai], with [di] the
%     distance from the robot to landmark Li, and [ai] the bearing angle from
%     the robot's x-axis.
%  6. The simulated variables are written in capital letters,
%       R: robot
%       W: set of landmarks or 'world'
%       Y: set of landmark measurements Y=[Y1 Y2 ... YN]
%  7. The true map is [xr;yr;ar;x1;y1;x2;y2;x3;y3; ... ;xN;yN]
%  8. The estimated map is Gaussian, defined by
%       x: mean of the map
%       P: covariances matrix of the map
%  9. The estimated entities (robot and landmarks) are extracted from {x,P}
%     via pointers, denoted in small letters as follows:
%       r:  pointer to robot state. r=[1,2,3]
%       l:  pointer to landmark i. We have for example l=[4,5] if i=1,
%       l=[6,7] if i=2, and so on.
%       m:  pointers to all used landmarks.
%       rl: pointers to robot and one landmark.
%       rm: pointers to robot and all landmarks (the currently used map).
%       Therefore:  x(r)     is the robot state,
%                   x(l)     is the state of landmark i
%                   P(r,r)   is the covariance of the robot
%                   P(l,l)   is the covariance of landmark i
%                   P(r,l)   is the cross-variance between robot and lmk i
%                   P(rm,rm) is the current full covariance -- the rest is
%                   unused.
%       NOTE: Pointers are always row-vectors of integers.
% 10. Managing the map space is done through the variable mapspace.
%     mapspace is a logical vector the size of x. If mapspace(i) = false,
%     then location i is free. Oterwise mapspace(i) = true. Use it as
%     follows:
%       * query for n free spaces: s = find(mapspace==false, n);
%       * block positions indicated in vector s: mapspace(s) = true;
%       * liberate positions indicated in vector s: mapspace(s) = false;
% 11. Managing the existing landmarks is done through the variable landmarks.
%     landmarks is a 2-by-N matrix of integers. l=landmarks(:,i) are the
%     pointers of landmark i in the state vector x, so that x(l) is the
%     state of landmark i. Use it as follows:
%       * query 1 free space for a new landmark: i = find(landmarks(1,:)==0,1)
%       * associate indices in vector s to landmark i: landmarks(:,i) = s
%       * liberate landmark i: landmarks(:,i) = 0;
```

```
% 12. Graphics objects are Matlab 'handles'. See Matlab doc for information.
% 13. Graphic objects include:
%       RG:  simulated robot
%       WG:  simulated set of landmarks
%       rG:  estimated robot
%       reG: estimated robot ellipse
%       lG:  estimated landmarks
%       leG: estimated landmark ellipses

%    (c) 2010, 2011, 2012 Joan Sola.

% I. INITIALIZE

% I.1 SIMULATOR -- use capital letters for variable names
%   W: set of external landmarks
W = cloister(-4,4,-4,4,7);    % Type 'help cloister' for help
%   N: number of landmarks
N = size(W,2);
%   R: robot pose [x ; y ; alpha]
R = [0;-2;0];
%   U: control [d_x ; d_alpha]
U = [0.1 ; 0.05]; % fixing advance and turn increments creates a circle
%   Y: measurements of all landmarks
Y = zeros(2, N);

% I.2 ESTIMATOR
% Map: Gaussian {x,P}
%   x: state vector's mean
x = zeros(numel(R)+numel(W), 1);
%   P: state vector's covariances matrix
P = zeros(numel(x),numel(x));

% System noise: Gaussian {0,Q}
q = [.01;.02];      % amplitude or standard deviation
Q = diag(q.^2);     % covariances matrix

% Measurement noise: Gaussian {0,S}
s = [.1;1*pi/180];  % amplitude or standard deviation
S = diag(s.^2);     % covariances matrix

% Map management
mapspace = false(1,numel(x));   % See Help Note #10 above.

% Landmarks management
landmarks = zeros(2, N);        % See Help Note #11 above

% Place robot in map
r           = find(mapspace==false, numel(R) ); % set robot pointer
mapspace(r) = true; % block map positions
x(r)        = R;    % initialize robot states
P(r,r)      = 0;    % initialize robot covariance

% I.3 GRAPHICS -- use the variable names of simulated and estimated
```

```
%   variables, followed by a capital G to indicate 'graphics'.
%   NOTE: the graphics code is long but absolutely necessary.

% Set figure and axes for Map
mapFig = figure(1); % create figure
cla                 % clear axes
axis([-6 6 -6 6])   % set axes limits
axis square         % set 1:1 aspect ratio

% Simulated World -- set of all landmarks, red crosses
WG = line(...
    'linestyle','none',...
    'marker','+',...
    'color','r',...
    'xdata',W(1,:),...
    'ydata',W(2,:));

% Simulated robot, red triangle
Rshape0 = .2*[...
    2 -1 -1  2; ...
    0  1 -1  0];                % a triangle at the origin
Rshape = fromFrame(R, Rshape0);  % a triangle at the robot pose
RG = line(...
    'linestyle','-',...
    'marker','none',...
    'color','r',...
    'xdata',Rshape(1,:),...
    'ydata',Rshape(2,:));

% Estimated robot, blue triangle
rG = line(...
    'linestyle','-',...
    'marker','none',...
    'color','b',...
    'xdata',Rshape(1,:),...
    'ydata',Rshape(2,:));

% Estimated robot ellipse, magenta
reG = line(...
    'linestyle','-',...
    'marker','none',...
    'color','m',...
    'xdata',[ ],...
    'ydata',[ ]);

% Estimated landmark means, blue crosses
lG = line(...
    'linestyle','none',...
    'marker','+',...
    'color','b',...
    'xdata',[ ],...
    'ydata',[ ]);
```

```matlab
% Estimated landmark ellipses, green
leG = zeros(1,N);
for i = 1:numel(leG)
leG(i) = line(...
    'linestyle','-',...
    'marker','none',...
    'color','g',...
    'xdata',[ ],...
    'ydata',[ ]);
end

% II. TEMPORAL LOOP

for t = 1:200

    % II.1 SIMULATOR
    % a. motion
    n = q .* randn(2,1);           % perturbation vector
    R = move(R, U, zeros(2,1) ); % we will perturb the estimator
                                 % instead of the simulator

    % b. observations
    for i = 1:N                      % i: landmark index
        v      = s .* randn(2,1);   % measurement noise
        Y(:,i) = observe(R, W(:,i)) + v;
    end

    % II.2 ESTIMATOR
    % a. create dynamic map pointers to be used hereafter
    m  = landmarks(landmarks≠0)'; % all pointers to landmarks
    rm = [r , m];                    % all used states: robot and landmarks
                                     % ( also OK is  rm = find(mapspace); )

    % b. Prediction -- robot motion
    [x(r), R_r, R_n] = move(x(r), U, n); % Estimator perturbed with n
    P(r,m)           = R_r * P(r,m);   % See PDF notes 'SLAM course.pdf'
    P(m,r)           = P(r,m)';
    P(r,r)           = R_r * P(r,r) * R_r' + R_n * Q * R_n';

    % c. Landmark correction -- known landmarks
    lids = find( landmarks(1,:) ); % returns all indices of existing landmarks
    for i = lids

        % expectation: Gaussian {e,E}
        l              = landmarks(:, i)';      % landmark pointer
        [e, E_r, E_l] = observe(x(r), x(l) ); % this is h(x) in EKF
        rl             = [r , l];              % pointers to robot and lmk.
        E_rl           = [E_r , E_l];          % expectation Jacobian
        E              = E_rl * P(rl, rl) * E_rl';

        % measurement of landmark i
        Yi = Y(:, i);
```

```
    % innovation: Gaussian {z,Z}
    z = Yi - e; % this is z = y - h(x) in EKF
    % we need values around zero for angles:
    if z(2) > pi
        z(2) = z(2) - 2*pi;
    end
    if z(2) < -pi
        z(2) = z(2) + 2*pi;
    end
    Z = S + E;

    % Individual compatibility check at Mahalanobis distance of 3-sigma
    % (See appendix of documentation file 'SLAM course.pdf')
    if z' * Z^-1 * z < 9

        % Kalman gain
        K = P(rm, rl) * E_rl' * Z^-1; % this is K = P*H'*Z^-1 in EKF

        % map update (use pointer rm)
        x(rm)     = x(rm)     + K*z;
        P(rm,rm) = P(rm,rm) - K*Z*K';

    end

end

% d. Landmark Initialization -- one new landmark only at each iteration
lids = find(landmarks(1,:)==0);    % all non-initialized landmarks
if ~isempty(lids)                  % there are still landmarks to initialize
    i = lids(randi(numel(lids)));  % pick one landmark randomly, its index is i
    l = find(mapspace==false, 2);  % pointer of the new landmark in the map
    if ~isempty(l)                 % there is still space in the map
        mapspace(l)     = true;    % block map space
        landmarks(:,i) = l;        % store landmark pointers

        % measurement
        Yi = Y(:,i);

        % initialization
        [x(l), L_r, L_y] = invObserve(x(r), Yi);
        P(l,rm)          = L_r * P(r,rm);
        P(rm,l)          = P(l,rm)';
        P(l,l)           = L_r * P(r,r) * L_r' + L_y * S * L_y';
    end
end

% II.3 GRAPHICS

% Simulated robot
Rshape = fromFrame(R, Rshape0);
set(RG, 'xdata', Rshape(1,:), 'ydata', Rshape(2,:));

% Estimated robot
```

```
    Rshape = fromFrame(x(r), Rshape0);
    set(rG, 'xdata', Rshape(1,:), 'ydata', Rshape(2,:));

    % Estimated robot ellipse
    re      = x(r(1:2));             % robot position mean
    RE      = P(r(1:2),r(1:2));      % robot position covariance
    [xx,yy] = cov2elli(re,RE,3,16); % x- and y- coordinates of contour
    set(reG, 'xdata', xx, 'ydata', yy);

    % Estimated landmarks
    lids = find(landmarks(1,:)); % all indices of mapped landmarks
    lx   = x(landmarks(1,lids)); % all x-coordinates
    ly   = x(landmarks(2,lids)); % all y-coordinates
    set(lG, 'xdata', lx, 'ydata', ly);

    % Estimated landmark ellipses -- one per landmark
    for i = lids
        l       = landmarks(:,i);
        le      = x(l);
        LE      = P(l,l);
        [xx,yy] = cov2elli(le,LE,3,16);
        set(leG(i), 'xdata', xx, 'ydata', yy);
    end

    % force Matlab to draw all graphic objects before next iteration
    drawnow
%     pause(1)

end
```

```
function f = cloister(xmin,xmax,ymin,ymax,n)

% CLOISTER  Generates features in a 2D cloister shape.
%   CLOISTER(XMIN,XMAX,YMIN,YMAX,N) generates a 2D cloister in the limits
%   indicated as parameters.
%
%   N is the number of rows and columns; it defaults to N = 9.

%   Copyright 2008-2009-2010 Joan Sola @ LAAS-CNRS.
%   Copyright 2011-2012-2013 Joan Sola

if nargin < 5
    n = 9;
end

% Center of cloister
x0 = (xmin+xmax)/2;
y0 = (ymin+ymax)/2;

% Size of cloister
hsize = xmax-xmin;
vsize = ymax-ymin;
```

```
tsize = diag([hsize vsize]);

% Integer ordinates of points
outer = (-(n-3)/2 : (n-3)/2);
inner = (-(n-3)/2 : (n-5)/2);

% Outer north coordinates
No = [outer; (n-1)/2*ones(1,numel(outer))];
% Inner north
Ni = [inner ; (n-3)/2*ones(1,numel(inner))];
% East (rotate 90 degrees the North points)
E = [0 -1;1 0] * [No Ni];
% South and West are negatives of N and E respectively.
points = [No Ni E -No -Ni -E];

% Rescale
f = tsize*points/(n-1);

% Move
f(1,:) = f(1,:) + x0;
f(2,:) = f(2,:) + y0;
```

References

[1] Cyril Roussillon, Aurélien Gonzalez, Joan Solà, Jean Marie Codol, Nicolas Mansard, Simon Lacroix, and Michel Devy. RT-SLAM: a generic and real-time visual SLAM implementation. In *Int. Conf. on Computer Vision Systems (ICVS)*, Sophia Antipolis, France, Sept. 2011.

[2] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *Int. Journal of Robotics Research*, 5(4):56–68, 1987.

[3] Joan Solà, David Marquez, Jean Marie Codol, and Teresa Vidal-Calleja. An EKF-SLAM toolbox for MATLAB, 2009.

[4] Joan Solà, Teresa Vidal-Calleja, Javier Civera, and José María Martínez Montiel. Impact of landmark parametrization on monocular EKF-SLAM with points and lines. *Int. Journal of Computer Vision*, 97(3):339–368, September 2012. Available online at Springer's: http://www.springerlink.com/content/5u5176nj521kl3h0/.