# Simultaneous Localization And Mapping via Extended Kalman Filter

*An informal tutorial for the* Intelligent Robotic Systems *course*

Giovanni Ciatto
giovanni.ciatto@studio.unibo.it

A.Y. 2015/2016

## 1 Introduction

In the context of robotics, Simultaneous Localization And Mapping (SLAM) is the problem of a robot dynamically constructing a map of an unknown environment and concurrently localizing itself within it, while exploring such environment. It may seem a chicken-or-egg problem, since a map is needed for localization and a frame of reference is needed for building a map, but the problem has been studied and solved both from a theoretical (probabilistic) and practical point of view in [TBF05].

SLAM involves a robotic agent being at least able to move within, and gather information about, the environment. Wheels and laser scanners are common choices to satisfy such requirements. Generally speaking, we will refer to them as *actuators* and *exteroceptive sensors*, respectively. Optionally, the robot may be equipped with one or more *odometric* sensors allowing it to actually measure its own movement, which should otherwise be inferred, e.g., by the speed imposed to the wheels. Generally speaking we call them *proprioceptive sensors*.

It is well understood in robotics that sensory data and inferred movement information are inherently noisy. From a probabilistic point of view this means that, if the robot is keeping track of its own position, the *uncertainty* of such position increases as the robot moves. Conversely, supposing the robot is able to detect some objects (*landmarks*, in jargon) within its surroundings, recognize them for a number of observations, and estimate the relative position between itself and each landmark, then it can use such information to reduce the uncertainty about its own position. Such a correlation

between position estimation and landmark measurement is clearly explained in [Bre14, Unit C].

The generic approach to SLAM requires the following models to be properly defined:

**Motion model:** describes how the robot updates the estimation of its own position and orientation according to the proprioceptive sensor data. It depends on the degrees of freedom of the robot and the nature of the available data. For instance, in this report, we consider the case of a differential robot constrained to move on a plane. So the robot pose variables are $x$, $y$ and $\theta$ (the *bearing*, in jargon) and the proprioceptive data consist of the last velocity values $v_l$ and $v_r$ imposed to the wheels motors.

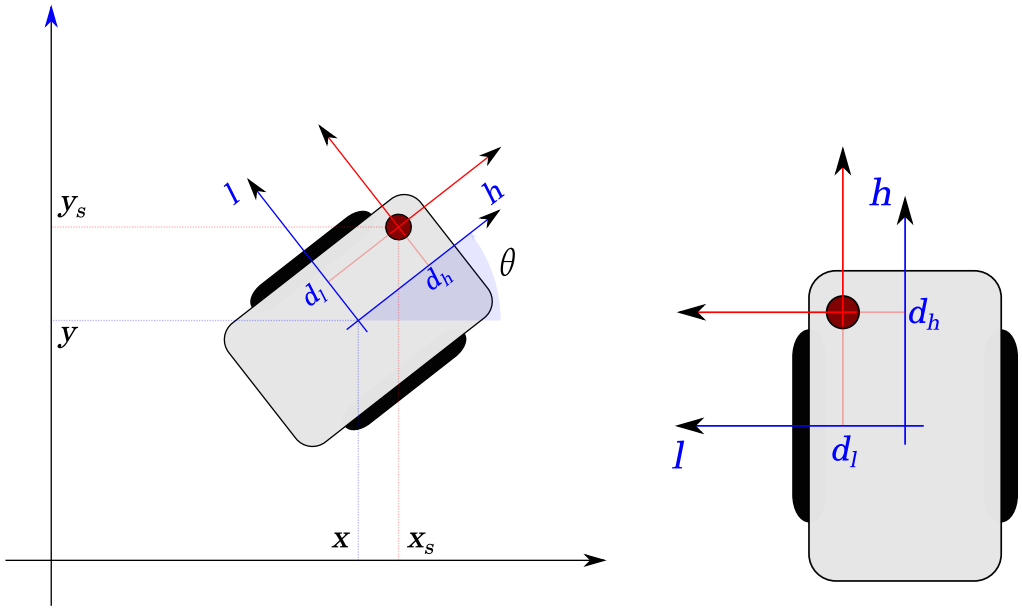
**Inverse Observation model:** describes how exteroceptive sensor data is used to deduce the landmarks positions, taking into account the current estimation of the robot position and orientation too. It depends on the nature of the data and the number of dimensions required to localize a landmark on the map. For example, in this report, we consider the case of a laser sensor providing, for each landmark, both its distance and angle w.r.t. the laser sensor. So the exteroceptive data consist of $(\rho, \alpha)$ pairs, which are used to deduce the landmark position $(x_m, y_m)$ on the map.

**Direct Observation model:** describes how to predict the expected exteroceptive sensor data for a known landmark. From a conceptual point of view, it's the inverse function of the Inverse Observation model. E.g., for our concerns, the Direct Observation Model takes into account the current estimation of the robot position and orientation $(x, y, \theta)$ and some known landmark position $(x_m, y_m)$ and computes the expected sensor data $(\rho, \alpha)$.

Such models are exploited by the following conceptual flow in order to produce a new estimation of the current position and orientation. It takes into account the previous estimation, and the last available proprioceptive and exteroceptive data:

1. *Prediction* phase: a basic estimation of the new position and orientation is achieved by combining the previous estimation with the proprioceptive data according to the motion model.

2. *Recognition* phase: the information from the exteroceptive data is analyzed in order to understand if it corresponds to an already known




(a) The *sensor* F.o.R., inside the *robot* F.o.R., inside the *map* F.o.R..

(b) The *robot* F.o.R. and the sensor position within it.

Figure 1: Frames of references (F.o.R.) considered the SLAM problem.

landmark or a new one. This normally exploits the inverse observation model: if it's a new landmark it must be added to the map.

3. *Observation* phase: when an already-known landmark is found, the current exteroceptive data relative to such a landmark (the *measurement*) is compared to the estimation achieved through the direct observation model for the same landmark (the *expected* measurement).

4. *Correction* phase: the difference between the measurement and the expected measurement is used to improve the basic estimation from the prediction phase and the landmark position estimation.

Our contribution consists of a practical tutorial for solving the SLAM problem through the *Extended Kalman Filter* (EKF), which supposes the robot state and the landmarks positions to be random vectors having a multivariate normal distribution, whose principal moments are estimated according to the phases described above.

# 2 Models and Reference Frames

It is important to understand that several Frames of Reference (F.o.R.) are involved in SLAM (as shown in Figure 1):

- The *map* (or *global*) F.o.R., i.e., the one used for representing the map and the current robot orientation and position. It is arbitrary.
- The *robot* (or *local*) F.o.R., i.e., the one defined according to the reference center of the robot. For what concerns this report, we consider a frame having its origin into the robot geometry center, the abscissas axis always coinciding with the heading direction, and the ordinate axis coinciding with the lateral direction (increasing on the left, as shown in Figure 1b).
- The *sensor* (or *measurement*) F.o.R., which depends on the exteroceptive sensor position within the robot frame, and is in general different from the robot frame. E.g., the sensor may be in position $(d_h,\, d_l)^\top$ w.r.t. the robot frame, as shown in Figure 1b, and this translation should be taken into account when handling sensor data in order to understand obstacles position on the map. In this tutorial we consider, for simplicity, the sensor frame to be coincident with the robot frame i.e., $d_h = d_h = 0$.

Since part of the SLAM process is to build a map of the environment under the hypothesis that the robot has no prior knowledge about it, there is no constraint on how the robot should choose the initial global frame. In what follows, we (and the robot) assume that the global frame coincides with the initial robot frame, i.e., the map frame is defined by the initial pose of the robot into the environment: the origin is its *very first* position, the abscissas axis is its *initial* heading direction and, consequently, the ordinate axis is its *initial* lateral direction.

## 2.1 The motion model

The motion model is a function $g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ mapping the previous robot pose vector, $\mathbf{r}_{t-1} \in \mathbb{R}^n$, i.e., the vector containing the robot position and orientation (which are relative to the *map* frame), into the current one, $\mathbf{r}_t \in \mathbb{R}^n$, by taking into account the last *control* vector $\mathbf{u}_t \in \mathbb{R}^m$, i.e., the vector containing proprioceptive data (which is relative to the *robot* frame). More formally:

$$\mathbf{r}_t = g(\mathbf{r}_{t-1},\, \mathbf{u}_t)$$

The motion model aims to keep track the robot movements in order to provide an updated estimation of its pose vector.

**Example: Differential robot on the plane.** Suppose we only own a two-wheeled, differential robot having no odometric sensor, which is often the case in experimental setups. The distance between the wheels is $2L$. Suppose we want to experiment the SLAM problem on planar arena containing some obstacles. Finally, suppose that each step of the robot controller is executed after $\Delta T_t$ seconds since the end of the previous step and that the duration of each step is negligible. Under such assumptions, movements relative to the $z$-axis are not so relevant: we are only interested in the robot to solve the SLAM problem on the 2D plane. In this setting we can assume the robot pose vector to be the triplet $\mathbf{r}_t = (x_t,\, y_t,\, \theta_t)^\top$, i.e., the position and bearing of the robot w.r.t. the global frame, and the control vector to be the triplet $\mathbf{u}_t = (v_{l,t},\, v_{r,t},\, \Delta T_t)^\top$, i.e., the velocities imposed to the wheels actuators at the end of the previous control step and the time elapsed since that moment. Notice that, if we assume $\Delta T_t$ to be constant, the control vector is the pair $\mathbf{u}_t = (v_{l,t},\, v_{r,t})^\top$. Under such hypotheses, the motion model could be defined as follows*:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \leftarrow \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \Delta T \cdot I \cdot \begin{pmatrix} \frac{v_l+v_r}{2} \\ 0 \\ \frac{v_r-v_l}{2L} \end{pmatrix} + \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.1}$$

where $\Delta T \cdot I$ is a diagonal $3 \times 3$ matrix having each element on the diagonal equal to $\Delta T$. Equation 2.1 computes the translation and rotation the robot performed within its reference frame during the last control step as a function of the velocity imposed to the wheels. Such values are converted into the map frame and added to the previous position and orientation in order to achieve the new ones.

Please consider reading Appendix A.1 and A.3 in order to recall conversions between reference frames and differential drive.

**Example: Robot equipped with odometric sensor on the plane.** Suppose our robot is embodied with an odometric sensor, which periodically provides a triplet $(dh,\, dl, d\theta)^\top$, i.e., the variations in the heading and lateral directions and the angular variation since the last update. Such variations are of course relative to the robot frame. We assume such a sensor frequency

*Here we write $\alpha \leftarrow f(\alpha)$ as a simplified notation for $\alpha_t = f(\alpha_{t-1})$, where $\alpha$ is any variable to be updated according to the value it had during the previous computational step and $t$ ranges over computational steps.

to be high. In this case the motion model could be simply defined as follows*:

$$\begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \leftarrow \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} dh \\ dl \\ d\theta \end{pmatrix} + \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.2}$$

This means the variations relative to the robot frame are simply converted into the map frame.

*Important.* Always take care of normalizing the angle $\theta + d\theta$, since it's a sum of angles.

## 2.2 The inverse observation model

The inverse observation model is a function $\eta : \mathbb{R}^n \times \mathbb{R}^p \to \mathbb{R}^q$ mapping a measurement vector, $\mathbf{s}_t \in \mathbb{R}^p$, i.e., the vector containing the data identifying an obstacle from the exteroceptive sensor (w.r.t. the *sensor* frame), into the position vector, $\mathbf{m}_t \in \mathbb{R}^q$, of the corresponding landmark (which is relative to the *global* frame), by taking into account the current *robot* pose vector $\mathbf{r}_t \in \mathbb{R}^n$. More formally:

$$\mathbf{m}_t = \eta(\mathbf{r}_t, \mathbf{s}_t)$$

The inverse observation model aims to produce an estimation of the position vector for some maybe-unknown landmark taking into account the measurement vector received by the exteroceptive sensor and the current estimation of the robot pose. Such estimation is compared with the known landmarks positions in order to understand if it is relative to one of them.

**Example: Laser scanner on top of a robot.** Suppose a laser scanner is located in position $(d_h, d_l)^\top$ within the robot frame†. In the real case, the sensor will probably be on the top of the robot. The laser scanners usually provide distance and angle (w.r.t. the sensor frame) for each obstacle which is closer than a maximum distance $D_{max} > 0$. Moreover, they are usually characterized by $W_V, W_H \subseteq [-\pi, \pi]$: vertical and horizontal angle windows, respectively. Using spheric coordinates, $\rho$, $\theta$ and $\phi$, we say that the sensor can only perceive obstacles such that $\rho \leq D_{max}$ and $\theta \in W_H$ and $\phi \in W_V$. For what concerns this example, we suppose that $W_H = (-\pi, \pi]$ and $W_V = \{\pi/2\}$, i.e., the robot can perceive obstacles in any direction on the horizontal plane. In the real case, one may take into account that the sensor be in an higher position than the robot and therefore not being able to perceive shorter obstacles.

†coordinates are expressed w.r.t. the *h*eading and *l*ateral axes

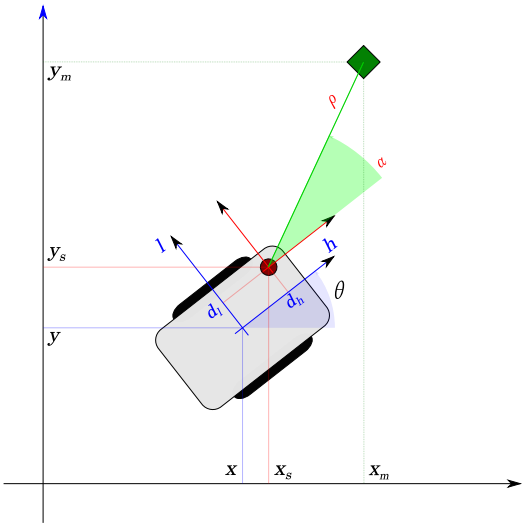



Figure 2: An obstacle on the 2D plane ant its coordinates both in the global and sensor frame.

Under such hypotheses[‡], the measurement vector is the distance-and-angle pair $\mathbf{s} = (\rho, \alpha)^\top$, the landmark position is the pair $\mathbf{m} = (x_m, y_m)^\top$, and the inverse observation model can be simply defined as follows:

$$\begin{pmatrix} x_m \\ y_m \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \left( \begin{pmatrix} d_x \\ d_y \end{pmatrix} + \rho \cdot \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \right) + \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.3}$$

Please note that, if $d_x = d_y = 0$, i.e., sensor is located into the robot center, Equation 2.3 can reduces to:

$$\begin{pmatrix} x_m \\ y_m \end{pmatrix} = \rho \cdot \begin{pmatrix} \cos(\theta + \alpha) & -\sin(\theta + \alpha) \\ \sin(\theta + \alpha) & \cos(\theta + \alpha) \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix}$$

## 2.3 The (direct) observation model

The observation model is a function $h : \mathbb{R}^n \times \mathbb{R}^q \to \mathbb{R}^p$ mapping the position vector $\mathbf{m}_t \in \mathbb{R}^q$ of some landmark (which is relative to the *global* frame), into the measurement vector, $\mathbf{s}_t \in \mathbb{R}^p$, (w.r.t. the *sensor* frame) expected for that landmark. It takes into account the current *robot* pose vector $\mathbf{r}_t \in \mathbb{R}^n$. More formally:

$$\mathbf{s}_t = h(\mathbf{r}_t, \mathbf{m}_t)$$

[‡]the subscript indicating the computational step is omitted in order to keep the notation readable

The observation model aims to produce an estimation of the measurement vector for some known landmark taking into account the current estimation of the robot pose vector and the landmark position vector. Such estimation is compared with the actual measurement vector relative to the recognized landmark and their difference is used to reduce the current uncertainty about the robot pose.

**Example: Laser scanner on top of a robot.** Here we consider the same hypotheses of the previous example[‡]. Let $(x_s,\, x_s)^\top$ be the position of the sensor w.r.t. the *global* frame: the same position is $(d_h,\, d_l)^\top$ within the *robot* frame. The equation relating the two coordinate frames is the following:

$$\begin{pmatrix} x_s \\ y_s \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \cdot \begin{pmatrix} d_h \\ d_l \end{pmatrix} + \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.4}$$

Notice that $(x_s,\, y_s)^\top$ is the origin of the *sensor* frame and that the measurement vector $\mathbf{s} = (\rho,\, \alpha)^\top$ consists of the polar coordinates of the landmark position $\mathbf{m} = (x_m,\, y_m)^\top$ according to such frame. So, the observation model is simply:

$$\begin{pmatrix} \rho \\ \alpha \end{pmatrix} = \begin{pmatrix} \sqrt{(x_m - x_s)^2 + (y_m - y_s)^2} \\ \text{atan2}(y_m - y_s,\ x_m - x_s) - \theta \end{pmatrix}$$

where $x_s$ and $y_s$ are both functions of $x$, $y$ and $\theta$.

Please note that, if $d_x = d_y = 0$, then $(x_s,\, y_s)^\top \equiv (x,\, y)^\top$, so the motion model reduces to:

$$\begin{pmatrix} \rho \\ \alpha \end{pmatrix} = \begin{pmatrix} \sqrt{(x_m - x)^2 + (y_m - y)^2} \\ \text{atan2}(y_m - y,\ x_m - x) - \theta \end{pmatrix}$$

which is indeeds a far simpler expression.

*Important.* Always take care of normalizing the angle $\alpha$, since it's a subtraction of angles.

# 3 Extended Kalman Filter

In this section we describe and motivate the EKF algorithm allowing us to face the SLAM problem.

## 3.1 The system state

Here we define the concept of (system) *state*, which will be pervasively used in the following.

The system state $\mathbf{x}_t$ consist of the robot pose vector $\mathbf{r}_t$ and the position vector $\mathbf{m}_1$, $\mathbf{m}_2$, ..., $\mathbf{m}_M$ of each known landmark. It is indeed defined as the concatenation of such vectors:

$$\mathbf{x}_t \stackrel{def}{=} \begin{pmatrix} \mathbf{r}_t \\ \mathbf{m}_1 \\ \mathbf{m}_2 \\ \vdots \\ \mathbf{m}_M \end{pmatrix} = \begin{pmatrix} \mathbf{r}_t \\ \mathbf{m} \end{pmatrix} \in \mathbb{R}^{n+M\cdot q}$$

where $\mathbf{m} \in \mathbb{R}^{M\cdot q}$ is a compact way to express the concatenation of all the known landmarks position vectors.

The purpose of the EKF-SLAM algorithm is to produce and update an estimation of the state vector exploiting the exteroceptive and proprioceptive data. It is important to understand that the robot *cannot* know its exact state because of the noise afflicting sensors and actuators. Hence, we assume the state to be a multi-normal random vector (please refer to Appendix A.2 for details) for which we assume to know the initial mean $\bar{\mathbf{x}}_0$ and covariances matrix $\Sigma_0$. The EKF-SLAM algorithm allows to compute the mean $\bar{\mathbf{x}}_t$ and covariances matrix $\Sigma_t$ of the current state as a function of the mean $\bar{\mathbf{x}}_{t-1}$ and covariances matrix $\Sigma_{t-1}$ of the previous state. The mean $\bar{\mathbf{x}}_t$ represents the most recent estimation of the robot pose and landmarks positions, while the covariances matrix $\Sigma_t$ stores the uncertainty of such estimation.

Generally speaking, the mean and covariances are in the following form:

$$\bar{\mathbf{x}}_t = \begin{pmatrix} \bar{\mathbf{r}}_t \\ \bar{\mathbf{m}}_{1,t} \\ \bar{\mathbf{m}}_{2,t} \\ \vdots \\ \bar{\mathbf{m}}_{M,t} \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{r}}_t \\ \bar{\mathbf{m}}_t \end{pmatrix} \in \mathbb{R}^{n+M\cdot q}$$

(notice that the expected position $\bar{\mathbf{m}}_{i,t}$ of any landmark may vary — i.e., be corrected — as time progresses, while the actual position $\mathbf{m}_i$ is supposed not to change)

$$\Sigma_t = \begin{pmatrix} \sigma_{\mathbf{r},\mathbf{r},t} & \sigma_{\mathbf{r},\mathbf{m},t} \\ \sigma_{\mathbf{m},\mathbf{r},t} & \sigma_{\mathbf{m},\mathbf{m},t} \end{pmatrix} \in \mathcal{M}_{(n+M\cdot q)\times(n+M\cdot q)}(\mathbb{R})$$

where $\sigma_{\mathbf{r},\mathbf{r},t} \in \mathcal{M}_{n\times n}(\mathbb{R})$ is the sub-matrix containing the covariances of the pose, $\sigma_{\mathbf{r},\mathbf{m},t} \in \mathcal{M}_{n\times M\cdot q}(\mathbb{R})$ is the sub-matrix containing the covariances of the pose w.r.t. the landmark positions, $\sigma_{\mathbf{m},\mathbf{r},t} = \sigma^{\top}_{\mathbf{r},\mathbf{m},t}$, since $\Sigma_t$ must be symmetric, and $\sigma_{\mathbf{m},\mathbf{m},t} \in \mathcal{M}_{M\cdot q\times M\cdot q}(\mathbb{R})$ is the sub-matrix containing the covariances of the landmark positions.

Please notice that $M$, i.e., the number of currently known and mapped landmarks, may actually vary as the robot moves, since more landmarks may be discovered.

### 3.1.1 The initial state

As stated before, we (and the robot) assume that the global frame coincides with the robot frame at the very first step of the SLAM algorithm. We also assume that the robot has no prior knowledge about the landmarks positions, i.e., $M_0 = 0$. This implies that:

$$\bar{\mathbf{x}}_0 = \bar{\mathbf{r}}_0 = \mathbf{0} \qquad\qquad \Sigma_0 = \mathbf{0}$$

meaning that the robot is supposed to have *null uncertainty* about its initial position to be the origin of the global frame.

In what follows we will explain:

- How $\bar{\mathbf{x}}_t$ and $\Sigma_t$ can be extended when a measurement vector $\mathbf{s}$ detects an unknown landmark
- How $\bar{\mathbf{x}}_t$ and $\Sigma_t$ can be computed as functions of $\bar{\mathbf{x}}_{t-1}$ and $\Sigma_{t-1}$, taking into account the last control vector $\mathbf{u}_t$ and some measurement vector $\mathbf{s}$ relative to some known landmark

## 3.2 Handling the noise

Here we discuss about how to model the intrinsic noise afflicting the robot sensors and actuators.

### 3.2.1 The system noise

Whenever the control software imposes (resp. receives) a control vector $\mathbf{u}_t$ to the robot actuators (resp. from the proprioceptive sensor), we assume such data to be inherently altered by the *system* noise $\boldsymbol{\varepsilon}_t \in \mathbb{R}^m$, i.e., the actual control vector perceived by the actuators (resp. proprioceptive sensor) is:

$$\mathbf{u'}_t = \mathbf{u}_t + \boldsymbol{\varepsilon}_t$$

For what concerns the EKF, it is important for $\boldsymbol{\varepsilon}_t$ to be modeled as a multi-normal random vector with null mean and covariances matrix $E \in \mathcal{M}_{m\times m}(\mathbb{R})$.

We can therefore consider the control vector $\bar{\mathbf{u}}_t$ provided to the EKF to be the expected value of the multi-normally distributed random vector $\mathbf{u'}_t$, having covariances matrix $E$.

**Example: Differential robot on the plane.** In this case we assume the left and right wheel actuators to be affected by a normally distributed error with null mean and standard deviations $\sigma_l$ and $\sigma_r$ meters per second, respectively. This means that, whenever the velocity $v_l$ (resp. $v_r$) is imposed to the left (resp. right) wheel actuator, the real speed of the left (resp. right) wheel will be $v_l + \varepsilon_l$ (resp. $v_r + \varepsilon_r$), where $\varepsilon_l$ (resp. $\varepsilon_r$) will be in $[-3\sigma_l, +3\sigma_l]$ (resp. $[-3\sigma_r, +3\sigma_r]$) with about 99% probability.

Under such hypotheses, the covariances matrix of $\boldsymbol{\varepsilon} = (\varepsilon_l, \varepsilon_r)^\top$ is:

$$E = \begin{pmatrix} \sigma_l^2 & 0 \\ 0 & \sigma_r^2 \end{pmatrix}$$

Notice that here we are implicitly supposing the left and right error to be uncorrelated, which may be, in general, not true[§].

### 3.2.2 The observation noise

Whenever the control software receives a measurement vector $\mathbf{s}$ from its exteroceptive sensor, we assume such data to be inherently altered be the *observation* noise $\boldsymbol{\delta} \in \mathbb{R}^p$, i.e., the actual measurement vector produced by the sensor is:

$$\mathbf{s}' = \mathbf{s} + \boldsymbol{\delta}$$

For what concerns the EKF, it is important for $\boldsymbol{\delta}$ to be modeled as a multi-normal random vector with null mean and covariances matrix $\Delta \in \mathcal{M}_{p\times p}(\mathbb{R})$

We can therefore consider each measurement vector $\bar{\mathbf{s}}$ provided to the EKF to be the expected value of some multi-normally distributed random vector $\mathbf{s}'$, having covariances matrix $\Delta$.

**Example: Laser scanner on top of a robot.** In this case we assume the laser sensor to be characterized by a normally distributed error with null mean for both the distance ($\delta_\rho$) and the angle ($\delta_\alpha$). The standard deviations of the distance error and the angular error are $\sigma_\rho$ meters and $\sigma_\alpha$ radians, respectively.

Under such hypotheses, the covariances matrix of $\boldsymbol{\delta} = (\delta_\rho, \delta_\alpha)^\top$ is:

$$\Delta = \begin{pmatrix} \sigma_\rho^2 & 0 \\ 0 & \sigma_\alpha^2 \end{pmatrix}$$

Notice that here we are implicitly assuming that the distance and angle error are uncorrelated and that the distance error does not depend on the distance itself, which may be, in general, not true[§].

[§]not sure about this, TO-DO: check

## 3.3 The prediction phase

In this phase the motion model $g(\cdot)$ is exploited to produce an estimation of the current robot state $\bar{\mathbf{x}}_t$ taking into account the previous robot state $\bar{\mathbf{x}}_{t-1}$ and the last control vector $\bar{\mathbf{u}}_t$. The uncertainty about the system state $\Sigma_t$ is computed accordingly.

Conceptually, this step is simple: the robot just needs to compute[¶]

$$\mathbf{x} \leftarrow \hat{g}(\mathbf{x}, \mathbf{u})$$

where $\hat{g}(\cdot)$ is the function applying the motion model $g(\cdot)$ to the first $n$ components of $\mathbf{x}$ and leaving the others unchanged[¶]:

$$\mathbf{x} = \begin{pmatrix} \mathbf{r} \\ \mathbf{m} \end{pmatrix} \leftarrow \begin{pmatrix} g(\mathbf{r}, \mathbf{u}) \\ \mathbf{m} \end{pmatrix} = \hat{g}(\mathbf{x}, \mathbf{u})$$

Unfortunately, the robot never knows the *real* system state or control vector or landmark positions, but it only knows their means and covariances as multi-normal variables. Mathematically, this step is complicated by $g(\cdot)$ being, in general, non-linear. So, while applying a linear transformation to the moments of some multi-normal variables, as explained in Appendix A.2, surely produces the moments of a multi-normal variable; applying a non-linear transformation does not guarantee so. This limitation is overcome by linearizing the $\hat{g}(\cdot)$ function into the point $(\bar{\mathbf{x}}_{t-1}, \bar{\mathbf{u}}_t)^\top$ as explained in Appendix A.4.

Hence, the predicted robot state mean $\bar{\mathbf{x}}_t$ is computed as follows[¶]:

$$\bar{\mathbf{x}} = \begin{pmatrix} \bar{\mathbf{r}} \\ \bar{\mathbf{m}} \end{pmatrix} \leftarrow \begin{pmatrix} g(\bar{\mathbf{r}}, \bar{\mathbf{u}}) \\ \bar{\mathbf{m}} \end{pmatrix} = \hat{g}(\bar{\mathbf{x}}, \bar{\mathbf{u}}) \tag{3.1}$$

assuming that the robot motion do not affect the landmark positions. Linearizing $\hat{g}(\cdot)$ allows us to compute the predicted robot state covariances matrix $\Sigma_t$. Such step is usually presented as follows:

$$\Sigma_t = \hat{G}_{\mathbf{x}} \cdot \Sigma_{t-1} \cdot \hat{G}_{\mathbf{x}}^\top \; + \; \hat{G}_{\mathbf{u}} \cdot E \cdot \hat{G}_{\mathbf{u}}^\top \tag{3.2}$$

where $\hat{G}_{\mathbf{x}} = \partial \hat{g}(\bar{\mathbf{x}}_{t-1}, \bar{\mathbf{u}}_t)/\partial \mathbf{x}$ and $\hat{G}_{\mathbf{u}} = \partial \hat{g}(\bar{\mathbf{x}}_{t-1}, \bar{\mathbf{u}}_t)/\partial \mathbf{u}$ are the Jacobians of the $\hat{g}(\cdot)$ function into the point $(\bar{\mathbf{x}}_{t-1}, \bar{\mathbf{u}}_t)^\top$ w.r.t. the system state and the control, respectively, and $E$ is the covariances matrix of the system noise.

[¶]Here we write $\alpha \leftarrow f(\alpha)$ as a simplified notation for $\alpha_t = f(\alpha_{t-1})$, where $\alpha$ is any variable to be updated according to the value it had during the previous computational step and $t$ ranges over computational steps.

Since we assumed the $\hat{g}(\cdot)$ function not affecting the landmarks positions, Equation 3.2 reduces to the following update rules[¶]:

$$\sigma_{\mathbf{r},\mathbf{r}} \leftarrow G_{\mathbf{r}} \cdot \sigma_{\mathbf{r},\mathbf{r}} \cdot G_{\mathbf{r}}^{\top} \; + \; G_{\mathbf{u}} \cdot E \cdot G_{\mathbf{u}}^{\top} \tag{3.3}$$

$$\sigma_{\mathbf{r},\mathbf{m}} \leftarrow G_{\mathbf{r}} \cdot \sigma_{\mathbf{r},\mathbf{m}} \tag{3.4}$$

$$\sigma_{\mathbf{m},\mathbf{r}} \leftarrow \sigma_{\mathbf{r},\mathbf{m}}^{\top} \tag{3.5}$$

$$\sigma_{\mathbf{m},\mathbf{m}} \leftarrow \sigma_{\mathbf{m},\mathbf{m}} \tag{3.6}$$

where $G_{\mathbf{x}} = \partial g(\bar{\mathbf{x}}_{t-1},\, \bar{\mathbf{u}}_t)/\partial \mathbf{x}$ and $G_{\mathbf{u}} = \partial g(\bar{\mathbf{x}}_{t-1},\, \bar{\mathbf{u}}_t)/\partial \mathbf{u}$ are the Jacobians of the motion model $g(\cdot)$ into the point $(\bar{\mathbf{x}}_{t-1},\, \bar{\mathbf{u}}_t)^{\top}$ w.r.t. the system state and the control, respectively.

Notice that, under such assumptions, the uncertainty related to the landmarks positions is not affected by the robot motion.

*Recap.* The prediction phase consists of the update rules expressed by equations 3.1, 3.3, 3.4, 3.5 and 3.6.

## 3.4 The observation phase

Suppose that $M > 0$, i.e., the robot already knows some landmarks positions, and suppose its exteroceptive sensor produced a measurement vector $\bar{\mathbf{s}}_i$ which allowed to *somehow* recognize the $i$-th landmark, whose currently estimated position is $\bar{\mathbf{m}}_{i,t}$ with uncertainty $\sigma_{\mathbf{m}_i,\mathbf{m}_i,t}$.

This phase exploits the direct observation model $h(\cdot)$ to estimate the moments of the multi-normally distributed random vector $\mathbf{z} \in \mathbb{R}^p$ (a.k.a. the *correction*) representing the difference between the actual measurement $\bar{\mathbf{s}}_i$ and the expected one $h(\bar{\mathbf{r}}_t,\, \bar{\mathbf{m}}_{i,t})$ considering the current robot pose $\bar{\mathbf{r}}_t$.

Conceptually this step is simple: the robot just needs to compute:

$$\mathbf{z} = \mathbf{s} - \hat{h}_i(\mathbf{x}_t) = \mathbf{s} - h(\mathbf{r}_t,\, \mathbf{m}_i)$$

where $\hat{h}_i(\cdot)$ is the function applying the direct observation model to the current robot pose $\mathbf{r}_t$ and the $i$-th landmark position $\mathbf{m}_i$ and ignoring the other landmarks stored into the state vector $\mathbf{x}_t$.

Sadly, as for the prediction phase, the robot never knows its *real* pose or any landmark position, but it only knows their first and second order moments as multi-normal variables. Mathematically, this step is complicated by $h(\cdot)$ being, in general, non-linear. As for the motion model, this limitation

is overcome by linearizing the $\hat{h}_i(\cdot)$ function into the point $\bar{\mathbf{x}}_t$, as explained in Appendix A.4.

So, the correction mean $\bar{\mathbf{z}}$ is computed as follows:

$$\bar{\mathbf{z}} = \bar{\mathbf{s}} - \hat{h}_i(\bar{\mathbf{x}}_t) \tag{3.7}$$

Linearizing $\hat{h}_i(\cdot)$ allows us to compute the correction covariances matrix $Z \in \mathcal{M}_{p\times p}(\mathbb{R})$. Such step is usually presented as follows:

$$Z = \hat{H}_{i,\mathbf{x}} \cdot \Sigma_t \cdot \hat{H}_{i,\mathbf{x}}^{\top} + \Delta \tag{3.8}$$

where $\hat{H}_{i,\mathbf{x}} = \partial \hat{h}_i(\bar{\mathbf{x}}_t)/\partial \mathbf{x}$ is the Jacobian of the $\hat{h}_i(\cdot)$ function into the point $\bar{\mathbf{x}}_t$ w.r.t. the system state, and $\Delta$ is the covariances matrix of the observation noise.

The $\hat{h}_i(\cdot)$ function, by definition, only takes into account $\bar{\mathbf{r}}_t$ and $\bar{\mathbf{m}}_{i,t}$, and consequently it is easy to demonstrate that $\hat{H}_{i,\mathbf{x}}$ is in the form:

$$\hat{H}_{i,\mathbf{x}} = \begin{pmatrix} H_{\mathbf{r}} & \mathbf{0} & \cdots & \mathbf{0} & H_{\mathbf{m}_i} & \mathbf{0} & \cdots & \mathbf{0} \end{pmatrix} \tag{3.9}$$

where $H_{\mathbf{r}} = \partial h(\bar{\mathbf{r}}_t,\, \bar{\mathbf{m}}_{i,t})/\partial \mathbf{r}$ and $H_{\mathbf{m}_i} = \partial h(\bar{\mathbf{r}}_t,\, \bar{\mathbf{m}}_{i,t})/\partial \mathbf{m}_i$ are the Jacobians of the observation model $h(\cdot)$ into the point $(\bar{\mathbf{r}}_t,\, \bar{\mathbf{m}}_{i,t})^{\top}$ w.r.t. the robot pose and $i$-th landmark position, respectively. Thus, Equations 3.7 and 3.8 reduce to:

$$\bar{\mathbf{z}} = \bar{\mathbf{s}} - h(\bar{\mathbf{r}}_t,\, \bar{\mathbf{m}}_i) \tag{3.10}$$

$$Z = \begin{pmatrix} H_{\mathbf{r}} & H_{\mathbf{m}_i} \end{pmatrix} \cdot \begin{pmatrix} \sigma_{\mathbf{r},\mathbf{r},t} & \sigma_{\mathbf{r},\mathbf{m}_i,t} \\ \sigma_{\mathbf{m_i},\mathbf{r},t} & \sigma_{\mathbf{m}_i,\mathbf{m}_i,t} \end{pmatrix} \cdot \begin{pmatrix} H_{\mathbf{r}}^{\top} \\ H_{\mathbf{m}_i}^{\top} \end{pmatrix} + \Delta \tag{3.11}$$

where $\Delta$ is the observation noise.

*Recap.* The observation phase consists of Equations 3.10 and 3.11.

## 3.5 The correction phase

This phase exploits the correction moments $\bar{\mathbf{z}}$ and $Z$, produced by the observation phase for the $i$-th landmark, to correct the robot state moments $\bar{\mathbf{x}}_t$ and $\Sigma_t$ produced by the prediction phase. In other words, the correction phase improves the estimation about the robot pose and the landmarks positions taking into account the difference between the expected and actual measurements of the $i$-th landmark.

Such a step only consists of algebraic operations. An clear intuition of what follows can be found in [Bre14, Unit C]. For a more exhaustive explanation, refer to [TBF05, Ch. 3].

The correction phase is usually presented as composed by the following equations:

$$K = \Sigma_t \cdot \hat{H}_{i,\mathbf{x}}^{\top} \cdot Z^{-1} \tag{3.12}$$

$$\bar{\mathbf{x}}_t \leftarrow \bar{\mathbf{x}}_t + K \cdot \bar{\mathbf{z}} \tag{3.13}$$

$$\Sigma_t \leftarrow \Sigma_t \ - \ K \cdot Z \cdot K^{\top} \tag{3.14}$$

where $K \in \mathcal{M}_{(n+M\cdot q)\times p}(\mathbb{R})$ is the so-called *Kalman gain* and $\hat{H}_{i,\mathbf{x}}$ is the Jacobian of the function $\hat{h}_i$ into the point $\bar{\mathbf{x}}_t$ w.r.t. the state vector, as defined in Subsection 3.4.

The Kalman gain definition can be simplified, since the Jacobian $\hat{H}_{i,\mathbf{x}}$ is sparse, as shown in Equation 3.9. Therefore, Equation 3.12 reduces to:

$$K = \begin{pmatrix} \sigma_{\mathbf{r},\mathbf{r},t} & \sigma_{\mathbf{r},\mathbf{m}_i,t} \\ \sigma_{\mathbf{m},\mathbf{r},t} & \sigma_{\mathbf{m},\mathbf{m}_i,t} \end{pmatrix} \cdot \begin{pmatrix} H_{\mathbf{r}}^{\top} \\ H_{\mathbf{m}_i}^{\top} \end{pmatrix} \cdot Z^{-1} \tag{3.15}$$

The new moments of the robot state, $\bar{\mathbf{x}}_t$ and $\Sigma_t$, produced by the correction phase, represent the outcome of the $t$-th step of the EKF algorithm and, consequently, the input of the $(t+1)$-th prediction phase.

*Recap.* The correction step consist of Equations 3.15, 3.13 and 3.14.

## 3.6 The recognition phase

TO-DO: write this

# A Mathematical background

## A.1 Affine transformations on the plane

Here we recall the affine transformation equation, mapping each point from a source 2D plane to another translated, scaled and rotated 2D plane:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x \cdot \cos x & -\sin x \\ \sin x & s_y \cdot \cos x \end{pmatrix} \cdot \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{A.1}$$

and its inverse:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{1}{s_x} \cdot \cos x & \sin x \\ -\sin x & \frac{1}{s_y} \cdot \cos x \end{pmatrix} \cdot \begin{pmatrix} x' \\ y' \end{pmatrix} - \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{A.2}$$

where $(x,\, y)^\top$ is a point in the source reference frame, $(x',\, y')^\top$ is the transformed reference frame, $(t_x,\, t_y)^\top$ is a translation vector, $s_x$ and $s_y$ are scale factors for the horizontal and vertical coordinates, respectively, and $\theta$ is the angle between the source reference frame abscissas axis and the transformed frame one.

## A.2 Multivariate normal distribution

Here we recall the notion of multivariate normal distribution and its linearity and geometric properties. Let $\mathbf{x} \in \mathbb{R}^n$ be a normally distributed vector having mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and covariances matrix $\Sigma \in \mathcal{M}_{n\times n}(\mathbb{R})$, then we write:

$$\mathbf{x} \sim \mathcal{N}(\boldsymbol{\mu},\, \Sigma)$$

meaning that the probability density function of $\mathbf{x}$ is the multidimensional Gaussian function:

$$pdf(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^n \cdot |\Sigma|}} \cdot \mathrm{e}^{-\frac{1}{2}\cdot(\mathbf{x}-\boldsymbol{\mu})^\top \cdot \Sigma^{-1} \cdot (\mathbf{x}-\boldsymbol{\mu})}$$

**Linearity.** Let $\mathbf{y} \in \mathbb{R}^m$ be a random vector, obtained by linearly combining a number of normally distributed independent random vectors $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}_i,\, \Sigma_i)$:

$$\mathbf{y} = A_1 \cdot \mathbf{x}_1 + A_2 \cdot \mathbf{x}_2 + \ldots + \mathbf{b}$$

where $A_i \in \mathcal{M}_{m\times n}(\mathbb{R})$ are transformation matrices and $\mathbf{b} \in \mathbb{R}^m$ is a constant vector, then $\mathbf{y}$ is normally distributed too, having mean $\boldsymbol{\mu}_y$ and covariances matrix $\Sigma_y$, expressed as follows:

$$\boldsymbol{\mu}_y = \mathbf{b} + \sum_i A_i \cdot \boldsymbol{\mu}_i \tag{A.3}$$

$$\Sigma_y = \sum_i A_i \cdot \Sigma_i \cdot A_i^\top \tag{A.4}$$

**Representation.** Usually, a Multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is imagined and represented as an hyper-ellipsoid centered in $\boldsymbol{\mu}$, whose axes are the (left) singular vectors of $\Sigma$. Such an ellipsoid is scaled (w.r.t. each axis) according to the singular values of $\Sigma$. In order to produce a rendering of such an hyper-ellipsoid (which is an ordinary ellipse in the 2D case), it is sufficient to produce a singular-values-decomposition of the covariances matrix:

$$\Sigma \overset{svd}{=} V \cdot D \cdot V^\top = \begin{pmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{pmatrix} \cdot \begin{pmatrix} d_1^2 & 0 & \cdots & 0 \\ 0 & d_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n^2 \end{pmatrix} \cdot \begin{pmatrix} \mathbf{v}_1^\top \\ \vdots \\ \mathbf{v}_n^\top \end{pmatrix}$$

where the $i$-th column of $V$, namely $\mathbf{v}_i$, is the versor identifying the direction of the $i$-th axis of the ellipsoid, and the $i$-th diagonal element of $D$, namely $d_i^2$ can be thought to be the variance of the distribution according to the $i$-th axis, i.e., $d_i$ can be thought to be its standard deviation.

In the 1-dimensional case it is common to represent the $k$-standard deviation interval, i.e., the circular interval centered on the mean and including each point whose distance from the mean is lower than $k$ times the standard deviation. Analogously, the $k$-th ellipsoid centered in $\boldsymbol{\mu}$ can be represented for the $n$-dimensional case by applying the following affine transformation $T : \mathbb{R}^n \to \mathbb{R}^n$ to each point on the unitary hyper-sphere:

$$T(\mathbf{c}) = k \cdot \begin{pmatrix} \mathbf{v}_1 & \cdots & \mathbf{v}_n \end{pmatrix} \cdot \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_n \end{pmatrix} \cdot \mathbf{c} + \boldsymbol{\mu}$$

where $\mathbf{c} \in \{ (x_1, \ldots, x_n)^\top \in \mathbb{R}^n \mid x_1^2 + \ldots + x_n^2 = 1 \}$.

## A.3 Differential drive

A differential robot can only exploit two non-rotating wheels. Its actuators can simply impose a velocity value to each wheel. We will call $v_l$ and $v_r$ the left and right speed value, while the distance between the wheels is $2L$. Here we recall how the linear velocity value $v$ in the heading direction and

the angular velocity $\omega$, w.r.t. the *instantaneous center of curvature*, can be computed as a function of $v_l$ and $v_r$:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} v_{l,r} \\ 0 \end{pmatrix} \text{ if } v_l = v_r = v_{l,r} \tag{A.5}$$

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \frac{v_l + v_r}{2} \\ \frac{v_r - v_l}{2L} \end{pmatrix} \text{ if } v_l \neq v_r \tag{A.6}$$

## A.4 Linearization of vectorial function

Let $\mathrm{f} : \mathbb{R}^n \to \mathbb{R}^m$ a non-linear function. If $f(\mathbf{x})$ is *continuous* and *differentiable* in some point $\mathbf{p} \in \mathbb{R}^n$ then it can be locally approximated as follows:

$$\mathrm{f}(\mathbf{x}) \overset{\mathbf{x}\to\mathbf{p}}{=} \mathrm{f}(\mathbf{p}) + \frac{\partial \mathrm{f}(\mathbf{p})}{\partial \mathbf{x}} \cdot (\mathbf{x} - \mathbf{p}) + o(\|\mathbf{x} - \mathbf{p}\|) \tag{A.7}$$

where $\partial \mathrm{f}(\mathbf{p})/\partial \mathbf{x}$ is the *Jacobian* of function f in $\mathbf{p}$. This is a generalization of the Taylor series first order approximation for the multidimensional case.

The Jacobian of some function $\mathrm{f}(\mathbf{x}) = (\, f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}) \,)^\top$ w.r.t. some variable vector $\mathbf{x} = (x_1, \ldots, x_n)^\top$ is the matrix of partial derivatives of the components of f respect to the components components of $\mathbf{x}$:

$$\frac{\partial \mathrm{f}(\mathbf{x})}{\partial \mathbf{x}} \overset{def}{=} \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(\mathbf{x})}{\partial x_1} & \frac{\partial f_m(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_m(\mathbf{x})}{\partial x_n} \end{pmatrix} \tag{A.8}$$

# References


[Bre14] Claus Brenner. Slam lectures. https://www.youtube.com/playlist?list=PLpUPoM7Rgzi_7YWn14Va2FODh7LzADBSm, 2014.

[Fri13] Albin Frischenschlager. Slam algorithm. http://ti.tuwien.ac.at/cps/teaching/courses/networked-embedded-systems/seminar.pdf, 2013.

[RB] Søren Riisgaard and Morten Rufus Blas. Slam for dummies. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-412j-cognitive-robotics-spring-2005/projects/1aslam_blas_repo.pdf.

[Sol14] Joan Solà. Simultaneous localization and mapping with the extended kalman filter. http://www.iri.upc.edu/people/jsola/JoanSola/objectes/curs_SLAM/SLAM2D/SLAM%20course.pdf, 2014.

[TBF05] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.