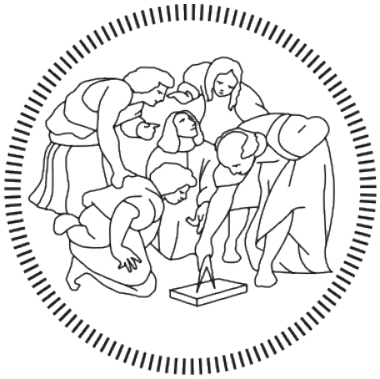

POLITECNICO DI MILANO

AIR Lab - Artificial Intelligence and Robotics Lab
Dipartimento di Elettronica, Informazione e Bioingegneria

MSc. in Computer Science and Engineering

Implementation of a 6 DoF EKF-SLAM for Leonardo Drone Contest

Author:
Diego Emanuel Avila
Matricola: 903988

Supervisor:
Prof. Matteo Matteucci

December 2020

Preface

Some preface

Diego Emanuel Avila
Milano
October 2020

Abstract

Abstract

Contents

List of Figures

List of Tables

List of Acronyms

Introduction 1

Theoretical Background 2

2.1 Robot Operating System

The *Robot Operating System* (ROS) is an open source middleware that provides inter-process communication through a message passing mechanism. It is, also, a collection of libraries, tools and conventions with the aim of simplify the development of software for robots. In this sense, one of the main components of the framework are nodes which encapsulate processes and/or algorithms.

2.1.1 Nodes

Nodes are processes that perform some computation and communicate between them using a publisher/subscriber infrastructure based on topics. A node subscribes to a particular topic, and it will receive messages of that type from a publisher node. This way, publisher is hid to the subscriber, reducing the coupling between them. The advantage of this mechanism is that nodes can be developed separately once the message structure is defined, forcing developers to implement clear interfaces for communication by using a message *Interface Description Language* (IDL). Moreover, this enables a modular and distributed development of the robotic system, while providing some fault tolerance and reduced code complexity.

2.1.2 Topics

As mentioned before, communication between nodes is done via a publisher/subscriber architecture based on topics, which are named buses over which messages are exchanged. A node that generates data, publishes it over a topic, and this information is consumed by those nodes subscribed to that topic. There can be several publishers (as well as subscribers) for a topic. The transport protocol used for exchanging messages is defined at runtime and can be of two types: TCPROS or UDPROS. The description of these protocols is beyond the scope of this document and the reader is invited to read the ROS documentation for detailed information.

2.1.3 Services

As well as topics, services are a way to communicate nodes. The difference is that topics are an asynchronous way of communication, while services are synchronous. This is a key difference, because nodes decide when to trigger a service and so, services are used to retrieve specific information or to ask another node to do something out of caller's node scope.

2.1.4 Tools

Regarding the tools provided by ROS, one that is be crucial for debugging and/or experimentation is the bag recording and playback. Since the message passing infrastructure is anonymous (meaning that nodes communicate between each others without knowing which node sent or received a message) it is possible to record the messages during a period of time,

without taking into consideration which node sent a message and which node received it. This recorder bag is useful for debugging because it can be played back, hence reproducing a previous experiment. Also, it can be used for development of new nodes that depend on the messages contained in the bag.

Another useful tool provided is *ROS Visualization* (RViz), a 3D visualization tool. With this tool it is possible to see the robot, orientation, reference frames, covariance matrices, etc. In addition, it is possible to draw lines, arrows, text and others, onto the environment in order to see useful information that cannot be extracted from messages.

2.2 Kalman Filter

Kalman Filter (KF), was introduced by Kalman (1960), and provides a recursive method for estimating the state of a dynamic system with presence of noise. It is a technique for filtering and prediction in *linear Gaussian systems* and applicable only to continuous states. One of its key features is that it simultaneously maintains estimates of the state vector and the estimate error covariance. It assures that the posteriors are Gaussian if the Markov assumption is hold, in addition to three conditions. The Markov assumption states that the past and future data are independent if one knows the current state; the other three conditions are the following:

1. The state transition probability $p(x_t|u_t, x_{t-1})$ must be a linear function in its arguments. This is guaranteed by $x_t = A_t x_{t-1} + B_t u_t + \epsilon_t$. Here, x_t and x_{t-1} are the state vectors, of size n, and u_t represents the control vector, of size m, at time t; A_t and B_t are matrices of size $n \times n$ and $n \times m$ respectively. In this way, the state transition function becomes linear in its arguments, hence KF assumes a linear system dynamics.

   ϵ_t represents the uncertainty introduced by the state transition, and it is a Gaussian random variable with zero mean and R_t covariance.

2. The measurement probability $p(z_t|x_t)$ must be linear in its arguments. This is guaranteed by $z_t = C_t x_t + \delta_t$, where z_t represents the measurement vector of size k, C_t is a matrix of size $k \times n$ and δ_t is a Gaussian noise with zero mean and Q_t covariance.

3. The initial belief should be normally distributed.

Given these three conditions, we are guaranteed that the posterior probability is Gaussian.

Algorithm 2.1: Kalman Filter algorithm

Input: μ_{t-1}, Σ_{t-1}, u_t, z_t

$\hat{\mu}_t = A_t \mu_{t-1} + B_t u_t$

$\hat{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

$K_t = \hat{\Sigma}_t C_t^T \left(C_t \hat{\Sigma}_t C_t^T + Q_t \right)^{-1}$

$\mu_t = \hat{\mu}_t + K_t (z_t - C_t \hat{\mu}_t)$

$\Sigma_t = (I - K_t C_t) \hat{\Sigma}_t$

return μ_t, Σ_t

The KF can be seen in Algorithm 2.1 and its input is the belief at time $t-1$, represented by its mean (μ_{t-1}) and its covariance (Σ_{t-1}), in addition to the control vector (u_t) and the observations (z_t). As a result, it returns the current belief characterized by its mean μ_t and

covariance Σ_t.

The first step in the algorithm (lines 1 and 2), represent the prediction step. It calculates the current belief before incorporating the observations, but after adding the control vector. The estimated belief is characterized by its mean, $\hat{\mu}_t$, and its covariance $\hat{\Sigma}_t$.

The second step (from line 3 to 5) starts by calculating K_t, the Kalman gain, which specifies the degree in which the observation is incorporated to the new state estimate. K_t can be seen as the weighting factor that weights the relationship between the accuracy of the predicted state estimate and the observation noise. When K_t is large, the observations have more importance in the final estimate; while if K_t is small, the observations do not have much importance in the correction step. Following, at line 4, the new mean is estimated by means of the Kalman gain and the *innovation*, which is the difference between the observation z_t and the expected measurement $C_t\hat{\mu}_t$. Finally, the new covariance is calculated.

2.2.1 Extended Kalman Filter

The linearity conditions that make the KF to work are, in some cases, far from reality: state transition functions and measurements are rarely linear in practice. The *Extended Kalman Filter* (EKF) works through a process of linearization, where nonlinear state transition and observation functions are approximated by a Taylor series expansion.

The Figure 2.1a shows the linear transformation of a random Gaussian variable, whose density function is $\mathcal{N}\left(x; \mu, \sigma^2\right)$. Assuming that the random variable is transformed using a linear function $y = ax + b$, the resulting random variable will be Gaussian with mean $a\mu + b$ and variance $a^2\sigma^2$.

However, as shown in Figure 2.1b, this does not happen if the transformation is not linear. In this case, assuming the original random variable is transformed using a nonlinear function g, the density of the resulting random variable is not Gaussian anymore.

The state transition probability and observation probabilities are ruled by nonlinear functions g and h respectively. Matrices A and B are replaced by function $g\left(u_t, x_{t-1}\right)$ and matrix H is replaced by function $h\left(x_t\right)$, making the belief not Gaussian. This is solved in EKF by approximating to the true belief, not the exact one as happens with linear KF. The approximation is done using a linearization method that approximates the nonlinear function by a linear function that is tangent to it, thereby maintaining the Gaussian properties of the posterior belief.

The used method is the first order Taylor expansion, which constructs a linear approximation of a function g from g's value and slope, which is given by

$$g'\left(u_t, x_{t-1}\right) = \frac{\partial g\left(u_t, x_{t-1}\right)}{\partial x_{t-1}} \tag{2.1}$$

Since g depends on the control variable u and the state x, we need to define a value for x, and the logical choice is the mean of the posterior in the previous time step: μ_{t-1}. This way

$$g\left(u_t, x_{t-1}\right) = g\left(u_t, \mu_{t-1}\right) + g'\left(u_t, \mu_{t-1}\right)\left(x_{t-1} - \mu_{t-1}\right) \tag{2.2}$$

where g' is the *Jacobian* of g, usually expressed as G_t, and it depends on u_t and μ_{t-1}, hence it changes through time.

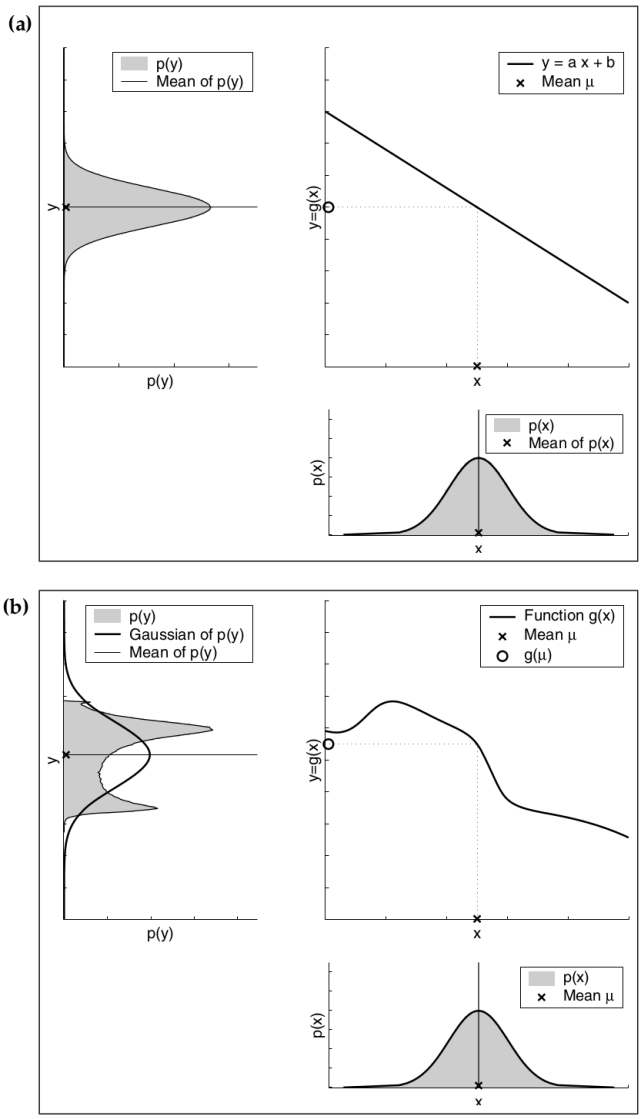

Figure 2.1: Linear **(a)** and nonlinear **(b)** transformation of a Gaussian random variable. The lower right plot shows the density function of the random variable. The upper right plot shows the transformation of the random variable. The upper left plot shows the resulting density function. Thrun et al. (2006)

The same linearization is applied to the observation function h:

$$h(x_t) = h(\hat{\mu}_t) + h'(\hat{\mu}_t)(x_t - \hat{\mu}_t) \tag{2.3}$$

$$h'(\hat{\mu}_t) = \frac{\partial h(x_t)}{\partial x_t} \tag{2.4}$$

where h' is the Jacobian of h, usually expressed as H_t. In this case, the linearization is done around $\hat{\mu}_t$, which is the state estimate just before computing h.

The Figure 2.2 depicts the approximation of g by a linear function that is tangent around its mean. The resulting density function is shown in the upper left plot with a dashed line,

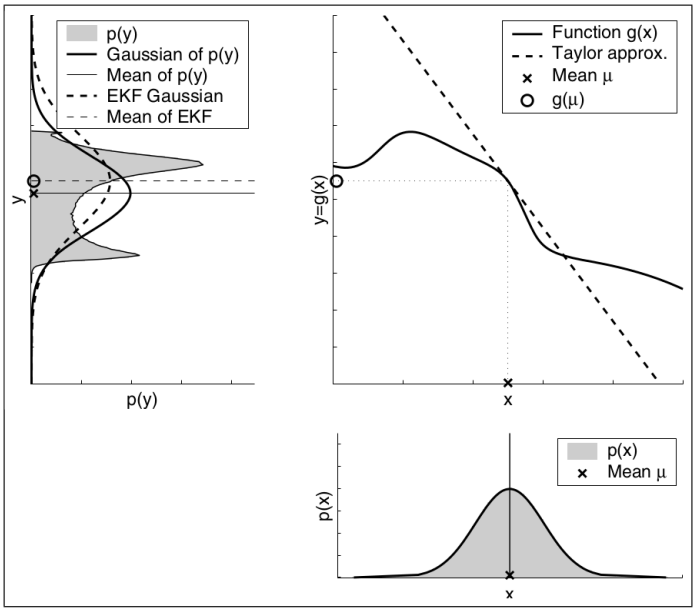

FIGURE 2.2: Linearization applied in EKF. In this case, the nonlinear function g is approximated using first order Taylor expansion, that is a linear function tangent to g at the mean of the original density function. The linearization is not perfect, so it adds an error, depicted in the upper left plot. This error is the difference between the dashed line and the solid line. Thrun et al. (2006)

that is similar to the original density function.

Algorithm 2.2: Extended Kalman Filter algorithm

Input: μ_{t-1}, Σ_{t-1}, u_t, z_t

1 $\hat{\mu}_t = g\left(u_t, \mu_{t-1}\right)$
2 $\hat{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
3 $K_t = \hat{\Sigma}_t H_t^T \left(H_t \hat{\Sigma}_t H_t^T + Q_t\right)^{-1}$
4 $\mu_t = \hat{\mu}_t + K_t \left(z_t - h\left(\hat{\mu}_t\right)\right)$
5 $\Sigma_t = (I - K_t H_t)\hat{\Sigma}_t$
6 **return** μ_t, Σ_t

The EKF algorithm can be seen in Algorithm 2.2, and it is similar to Algorithm 2.1. The difference lies in the usage of the nonlinear functions g and h and their Jacobians, G_t and H_t respectively.

2.3 Simultaneous Localization and Mapping

Among all the problems faced by autonomous mobile robots, two of them are relevant for this work: localization and mapping. The former one, is related to the problem of where the robot is, while the later is related to building a map of the environment. However, to accurately localize itself the robot needs a map of the environment in which it is immersed in,

and, in order to build a map, it needs to know where it currently is, giving us a chicken-egg situation. Hence, the *Simultaneous Localization And Mapping* (SLAM) problem appears when the robot does not know the map nor its localization, while measurements and controls are given.

The problem of building a map can be summarized in the following steps:

1. The robot sense the environment using its sensors
2. It creates a representation of the acquired data
3. It integrates the processed sensor data with the previously learned map structure

While this process can be done by manually moving the robot around the environment, it is more challenging to build the map while the robot is moving autonomously.

On the other hand, assuming that the robot already knows the map, the localization problem could be trivial if no noise is present at all. The sensors, wheel encoders, different kinds of terrain, battery life, etc, all of these can make the robot to increase its uncertainty related to where it is. As robot moves around the environment it uses its sensors to estimate its position, increasing its uncertainty regarding its position relative to the map. At some point, the robot will "see" a known landmark or feature in the environment, correcting its position while reducing the uncertainty.

The localization and mapping problems can be solved together by using a SLAM technique, with which the robot will build the map while localizing itself in it. An example of this problem can be seen in Figure 2.3, where a robot moves around the environment and sees some features or landmarks. The uncertainty regarding its position is low when it starts, and keeps growing while it moves around. At the end, it sees a known landmark (m_0) making the uncertainty to shrink. As can be seen in the figure, the robot adds new landmarks to the map (m_1 and m_2) with their corresponding uncertainty, and when the robot sees the first landmark, not only its own uncertainty decreases, but also the two new landmarks' uncertainty. In this way, the robot's position is correlated with the observations' position estimates.

The idea of the SLAM problem is to estimate a posterior belief that involves not only the robot pose, but also the map: $p\left(x_t, m | z_{1:t}, u_{1:t}\right)$, where x_t is the robot's pose at time t, m is the map, $z_{1:t}$ are the measurements, and $u_{1:t}$ are the controls given to the robot.

2.3.1 EKF-SLAM

The SLAM problem can be addressed, between others, using an EKF approach. The algorithm proceeds in the same way as shown in Section 2.2.1, being μ a state vector containing the information about the robot pose (q_r) and the landmarks' pose (m_i):

$$\mu = \begin{bmatrix} q_r & m_0 & \dots & m_{n-1} \end{bmatrix}^T \tag{2.5}$$

One thing that worth mentioning is that in EKF-SLAM maps are *feature based*, meaning that the features or landmarks are assumed to be points in the space: if the robot sees, for example, a chair, it will store a point that represents the chair in space. Also, as explained before, it assumes Gaussian noise for the robot motion and observations.

The EKF-SLAM algorithm estimates the robot's pose in addition to all encountered landmarks' poses along its way. Thus, there is a correspondence between robot's pose and landmarks, and that is why it is necessary to include the landmarks information into the

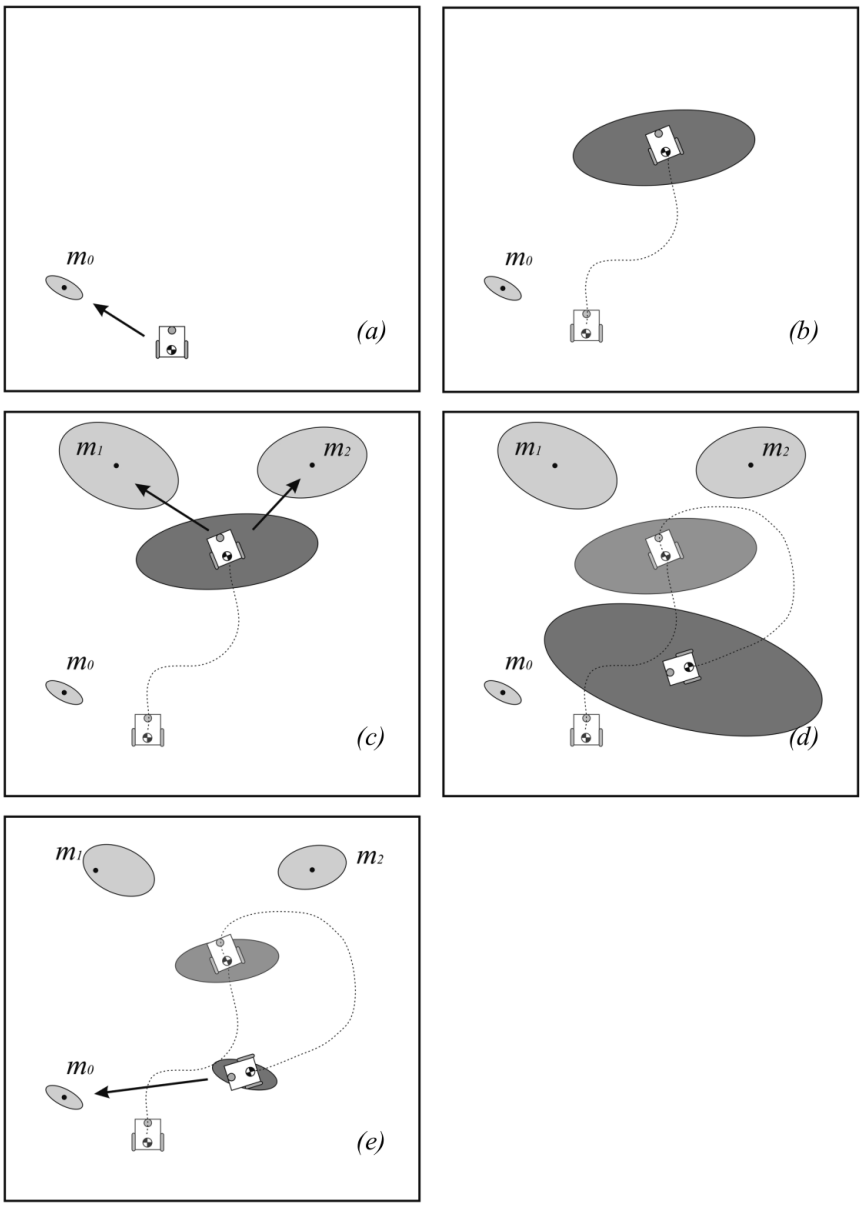

FIGURE 2.3: At the beginning **(a)** the robot has low uncertainty regarding its position. As it moves around the environment its uncertainty grows **(b), (c), (d)**, until it sees a known marker **(e)**, making the position uncertainty to shrink. Siegwart et al. (2004)

state vector. Hence, the algorithm estimates the posterior $p(\mu_t|z_t, u_t)$.

Assuming the robot's pose is composed by x, y, θ , the markers' pose is composed by x, y, and there are N markers, the state vector μ will have a length of $3 \times 2N$, and the covariance matrix Σ will have a size of $(3 \times 2N) \times (3 \times 2N)$.

The algorithm can be seen in Algorithm 2.3. From lines 1 to 3, the algorithm computes the state vector and covariance matrix updates; the function *computeJacobian* computes,

Algorithm 2.3: EKF-SLAM algorithm

Input: μ_{t-1}, Σ_{t-1}, u_t, z_t

1 $\hat{\mu}_t = g\left(\mu_{t-1}, u_t\right)$;
2 $G_t = computeJacobian\left(g\right)$;
3 $\hat{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$;
4 **foreach** *landmark observation* z_t^i **do**
5 **if** *landmark i has not being seen before* **then**
6 $addLandmarkToStateVector\left(z_t^i\right)$
7 $H_t^i = computeJacobian\left(h^i\right)$;
8 $S = H_t^i \hat{\Sigma}_t H_t^{iT} + Q_t$;
9 $K_t^i = \hat{\Sigma}_t H_t^{iT} S^{-1}$;
10 $\mu_t = \hat{\mu}_t + K_t^i \left(z_t^i - h^i\left(\hat{\mu}_t\right)\right)$;
11 $\Sigma_t = (I - K_t^i H_t^i)\hat{\Sigma}_t$;
12 **return** μ_t, Σ_t

indeed, the Jacobian matrix for the motion model g, the resulting matrix has the same size as the covariance matrix and has the following characteristic:

$$G_t = \begin{bmatrix} G_r & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$$

$$G_r = \begin{bmatrix} \frac{\partial x'}{\partial \mu_{t-1,x}} & \frac{\partial x'}{\partial \mu_{t-1,y}} & \frac{\partial x'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial y'}{\partial \mu_{t-1,x}} & \frac{\partial y'}{\partial \mu_{t-1,y}} & \frac{\partial y'}{\partial \mu_{t-1,\theta}} \\ \frac{\partial \theta'}{\partial \mu_{t-1,x}} & \frac{\partial \theta'}{\partial \mu_{t-1,y}} & \frac{\partial \theta'}{\partial \mu_{t-1,\theta}} \end{bmatrix}$$

where $\frac{\partial x'}{\partial \mu_{t-1,x}}$ is the derivative of g along x' dimension, taken with respect to x at μ_{t-1}. At line 4, it iterates through every observation z_t. If the landmark is not already in the state vector, at line 6 it is added by projecting the observation and calculating the landmark's pose, adding two new elements to the state vector and two new more columns and rows to the covariance matrix. At line 7 the Jacobian of the observation model is computed, while at line 9 the Kalman gain is computed. At line 10, the new state vector is estimated, and the gain propagates the information through all the state vector, updating not only the robot's pose, but also the landmarks' poses.

The fact that the Kalman gain is not sparse is important, because observing a landmark does not just improves the estimate of that landmark's pose, but also all the others, along with the robot's pose. This effect can be seen in Figure 2.3, with an additional explanation: most of the uncertainty of the landmarks' poses is caused by the robot's own uncertainty, so the location of those previously seen landmarks are correlated. When the robot gains information about its own pose, this information is propagated to the landmarks, and as result it improves the localization of other landmarks in the map.

Bibliography

Robot operating system. URL `https://www.ros.org/`.

Howie Choset et al. *Principles of robot motion: theory, algorithms, and implementation*. Intelligent Robotics and Autonomous Agents. The MIT Press, 2005. ISBN 9780262033275.

R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960. ISSN 0021-9223. doi: 10.1115/1.3662552. URL `https://doi.org/10.1115/1.3662552`.

Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. Intelligent robotics and autonomous agents. The MIT Press, 2nd edition, 2004. ISBN 978-0-262-01535-6.

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Intelligent robotics and autonomous agents. The MIT Press, 2006. ISBN 978-0-262-20162-9.