ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA
Scuola di Ingegneria e Architettura

CORSO DI LAUREA MAGISTRALE IN
Ingegneria Elettronica e Telecomunicazioni per l'Energia

SIMULTANEOUS LOCALIZATION AND MAPPING TECHNOLOGIES

Elaborato in
Reti di Sensori Wireless per Monitoraggio Ambientale

Relatore
Chiar.mo Prof. Ing.
DAVIDE DARDARI

Correlatore
Dott. Ing.
NICOLÒ DECARLI
FRANCESCO GUIDI

Presentata da
NOUR NAGHI

Sessione III

Anno Accademico 2017-2018

Contents

Acknowledgements

Abstract

Il problema dello SLAM (Simultaneous Localization And Mapping) consiste nel mappare un ambiente sconosciuto per mezzo di un dispositivo che si muove al suo interno, mentre si effettua la localizzazione di quest'ultimo.

All'interno di questa tesi viene analizzato il problema dello SLAM e le differenze che lo contraddistinguono dai problemi di mapping e di localizzazione trattati separatamente.

In seguito, si effettua una analisi dei principali algoritmi impiegati al giorno d'oggi per la sua risoluzione, ovvero i filtri estesi di Kalman e i particle filter.

Si analizzano poi le diverse tecnologie implementative esistenti, tra le quali figurano sistemi SONAR, sistemi LASER, sistemi di visione e sistemi RADAR; questi ultimi, allo stato dell'arte, impiegano onde millimetriche (mmW) e a banda larga (UWB), ma anche tecnologie radio già affermate fra le quali il Wi-Fi.

Infine, vengono effettuate delle simulazioni di tecnologie basate su sistema di visione e su sistema LASER, con l'ausilio di due pacchetti open source di MATLAB. Successivamente, il pacchetto progettato per sistemi LASER è stato modificato al fine di simulare una tecnologia SLAM basata su segnali Wi-Fi.

L'utilizzo di tecnologie a basso costo e ampiamente diffuse come il Wi-Fi apre alla possibilità, in un prossimo futuro, di effettuare localizzazione indoor a basso costo, sfruttando l'infrastruttura esistente, mediante un semplice smartphone. Più in prospettiva, l'avvento della tecnologia ad onde millimetriche (5G) consentirà di raggiungere prestazioni maggiori.

Keywords

- SLAM
- Mapping
- Localization
- Indoor
- EKF
- RADAR
- LASER
- LIDAR
- UWB
- mmW
- Visual

List of Abbreviations

SLAM Simultaneous Localization And Mapping

PF Particle Filter

RBPF Rao-Blackwellized Particle Filter

KF Kalman Filter

EKF Extended Kalman Filter

PDF Probability Density Function

FB Feature Based

OGB Occupancy Grid Based

MD Mobile Device

RD Reference Device

RSS Reference Signal Strength

TOA Time Of Arrival

TDOA Time Difference Of Arrival

PDOA Phase Difference Of Arrival

AOA Angle Of Arrival

LOS Line Of Sight

IMU Inertial Measurement Unit

INS Inertial Navigation System

CPU Central Processing Unit

CPU Graphics Processing Unit

MSE Mean Squared Error

MMSE Minimum Mean Square Error

MAP Maximum A-Posteriori

ML Maximum Likelihood

BLUE Best Linear Unbiased Estimator

EKF Extended Kalman Filter

GPS Global Positioning System

LASER Light Amplification by Stimulated Emission of Radiation

RADAR Radio Detection And Ranging

LIDAR Laser Imaging Detection And Ranging

SONAR Sound Navigation And Ranging

UWB Ultra Wide Band

mmW millimetre Wave

2D 2 Dimensional

3D 3 Dimensional

RMSE Root Mean Square Error

Chapter 1

Introduction

The Simultaneous Localization And Mapping (SLAM) problem consists in the simultaneous reconstruction of an unknown environment and the localization of a device moving within it. This problem is more general and harder than the single mapping or localization problems, that can be seen as particular cases of SLAM.

In the mapping problem, the device is aware of his absolute location, so it can use this information along with the data coming from the sensors observing or interacting with the environment to infer its topological map.

In the localization problem, a map or some reference devices are made available in order to obtain the position of the device observing or interacting with them.

The scenarios where there is not any a-priori knowledge related to location and map, are those in which the SLAM becomes necessary.

Many applications require the reconstruction of a consistent map; in general, in many applications, a robot may have the goal of exploring an environment and report it to a human operator. For instance, if we consider outdoor autonomous navigation of robots, if the robot has access to GPS, the SLAM is not required; conversely, in indoor environments, the use of GPS for localization scope is ruled out.

The solutions to this problem are employed in several applications including:

- Self-driving cars and drive assistance systems
- Unmanned Aerial Vehicles (drones)
- Autonomous Underwater Vehicles
- Planetary rovers

- Domestic robots
- Security systems
- Autonomous navigation of robots
- Augmented reality

This thesis aims at understanding the general SLAM problem and his main techniques of resolution, investigating the actual state of the art and simulating through MATLAB different technologies.

It is suddivided in the following chapters:

- *Mathematical tools for the study of dynamic physical systems*: the mathematical tools useful for the comprehension and the study of the SLAM topics are here introduced and explained; specifically, the mathematical representation of a physical dynamic system and the methods for the implementation of Bayesian filters are put in evidence.
- *Localization and mapping in dynamic physical systems*: this chapter focuses on the general definition of localization, mapping and SLAM problems and the differences between them.
- *State of the art related to the SLAM problem*: here a general view on the state of the art is given; the anatomy of a SLAM implementation is viewed and the main technologies exploited are treated.
- *MATLAB implementation of different SLAM technologies*: The experimental simulations done on different technologies are here analyzed; Two different SLAM toolbox implemented in MATLAB are exploited; the first is based on vision and the second on the LASER.

Chapter 2

Mathematical tools for the study of dynamic physical systems

In this chapter, the mathematical tools useful for the comprehension and the study of the topics that are covered in this thesis are put on focus.

In particular, the mathematical tools that are used nowadays to solve the SLAM problem are going to be analyzed: in the next subchapters, the following topics are taken into consideration:

- The numerical representation of a physical dynamic system
- The Bayesian filter, used to estimate the state of a system
- A few existing techniques for the implementation of a Bayesian filter, including the Kalman filter, the extended Kalman filter and the particle filter

2.1 Representation of a physical dynamic system

In the study of a physical dynamic system, the estimation of the state is needed, starting from the noisy measurements performed on the system, in order to let the controller act consequently through a control signal on the actuator; this process is described in figure 2.1. The state of the system evolves in time and is conditioned by uncertainty, just like the measurements performed on it; generally, the estimation process of the state may generate optimal solutions with MMSE techniques (Minimum Mean Square Error),

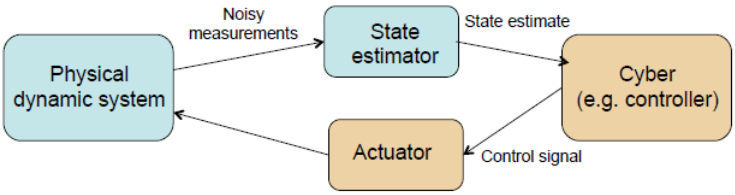

Figure 2.1: Representation of the control on a dynamic physical system[1]

but at the cost of large computational expenses, so that recursive approaches are preferred, since they are characterized by a lower complexity and the possibility to perform the estimation in real-time.

Physical systems are modelled by differential equations and a state space; since computers only process discrete-time data, this model is taken over by his numerical translation based on difference equations and discrete state space, where the involved variables are described as follows

$$\mathbf{x}_n = f(\mathbf{x}_{n-1}, \mathbf{u}_n) \tag{2.1}$$

$$\mathbf{y}_n = h(\mathbf{x}_n, \mathbf{u}_n) \tag{2.2}$$

where:

- $\mathbf{x}_n$ represents the state vector at the instant n
- $\mathbf{y}_n$ represents the measurements vector at the instant n
- $\mathbf{u}_n$ represents the action of the controller at the instant n

What has been described is valid only for deterministic systems, but if uncertainties are characterized by random processes, it becomes necessary to switch from the study of deterministic functions to the study of conditioned transition probability density functions (PDF) that are described as follows

$$p(\mathbf{x}_n|\mathbf{x}_{0:n-1}, \mathbf{u}_{1:n}) \tag{2.3}$$

$$p(\mathbf{y}_n|\mathbf{x}_{0:n-1}, \mathbf{y}_{1:n-1}, \mathbf{u}_{1:n-1}) \tag{2.4}$$

where:

- $\mathbf{x}_{0:n-1}$ represents the whole set of states until the instant $n-1$

- $\mathbf{u}_{1:n}$ represents the whole set of control actions until the instant n
- $\mathbf{y}_{1:n-1}$ represents the whole set of measurements until the instant $n-1$

An instance of a linear model with Additive Gaussian Noise is the following

$$\mathbf{x}_n = \mathbf{A}x_{n-1} + \mathbf{B}u_n + \mathbf{w}_n \tag{2.5}$$

The purpose is to obtain the best estimate (e.g., in the MMSE sense) of the succession of states of the system until an instant n (namely statistical inversion or optimal filtering):

$$\mathbf{x}_{0:n} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n\} \tag{2.6}$$

starting from a succession of n measurements until the instant n

$$\mathbf{y}_{0:n} = \{\mathbf{y}_0, \mathbf{y}_1, ..., \mathbf{y}_n\} \tag{2.7}$$

Solving this problem means calculating the a-posteriori joint probability distribution of the whole sequence of states n given all the noisy measurements and the control actions until the instant n:

$$p(\mathbf{x}_{0:n}|\mathbf{y}_{1:n}, \mathbf{u}_{1:n}) = \frac{p(\mathbf{y}_{1:n}, \mathbf{u}_{1:n}|\mathbf{x}_{0:n})p(\mathbf{x}_{0:n})}{p(\mathbf{y}_{1:n}, \mathbf{u}_{1:n})} \tag{2.8}$$

where:

- $p(\mathbf{x}_{0:n})$ is the a-priori PDF of the succession of states
- $p(\mathbf{y}_{1:n}|\mathbf{u}_{1:n}, \mathbf{x}_{0:n})$ is the likelihood function of the noisy measurements and the control actions
- $p(\mathbf{y}_{1:n}, \mathbf{u}_{1:n})$ is a normalization constant

Regarding the problem treated in this thesis, starting from the noisy observations $\mathbf{y}_{0:n}$ and the control actions $\mathbf{u}_{1:n}$, what is wanted is not the resolution of the whole succession of states $\mathbf{x}_{0:n}$, which can be done only after all measurements have been collected, then not in real time; it is wanted the resolution of the single current state $\mathbf{x}_n$ for each instant n, using the marginal a-posteriori PDF, called *belief*

$$Bel(\mathbf{x}_n) = p(\mathbf{x}_n|\mathbf{y}_{1:n}, \mathbf{u}_{1:n}) \tag{2.9}$$

This can be realized using Bayesian filters.

2.2 Bayesian filters

The main drawback in applying the 2.9 is that it has to be recomputed whenever a new measurement is taken, making the computational complexity intractable as n increases. Thanks to the Bayesian filters, the complexity can be drastically reduced exploiting the following Markov hypothesis [8]:

- The state at the step n depends only on the immediately preceding state $\mathbf{x}_{n-1}$ and the applied control $\mathbf{u}_n$, while it is independent of the observations

$$p(\mathbf{x}_n|\mathbf{x}_{0:n-1}, \mathbf{y}_{1:n-1}, \mathbf{u}_{1:n-1}) = p(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{u}_n) \tag{2.10}$$

  This PDF is called *motion or mobility model* [5]

- The current noisy observation $\mathbf{y}_n$ is independent of the previous states, input controls and observations, but depends only on the current state $\mathbf{x}_n$

$$p(\mathbf{y}_n|\mathbf{x}_{0:n-1}, \mathbf{y}_{1:n-1}, \mathbf{u}_{1:n-1}) = p(\mathbf{y}_n|\mathbf{x}_n) \tag{2.11}$$

  This PDF is called *perception, measurement or observation model* [5]

These hypothesis allow to get the Markovian model of the state space, which is characterized from the following properties:

- At first, the a-priori PDF $p(\mathbf{x}_0)$ is used to define the initial uncertainty of the system state
- The observation model of the system $p(\mathbf{y}_n|\mathbf{x}_n)$, shows the dependence of the noisy measurements at a certain instant from the state at the same instant
- The system state dynamics is described by the mobility model $p(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{u}_n)$

Figure 2.2: Bayesian filter

The Bayesian filter is a recursive estimator that at every step exploits the mobility model $p(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{u}_n)$, the control action $\mathbf{u}_n$ and the new measurement $\mathbf{y}_n$ to get the belief of the current state starting from the one of the previous instant $p(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1})$ (figure 2.2)

The estimation of the state occurs according to the following iterative procedure, also illustrated in figure 2.3:

- The a-priori PDF of the state $p(\mathbf{x}_0)$ poses as the initial belief of the state;

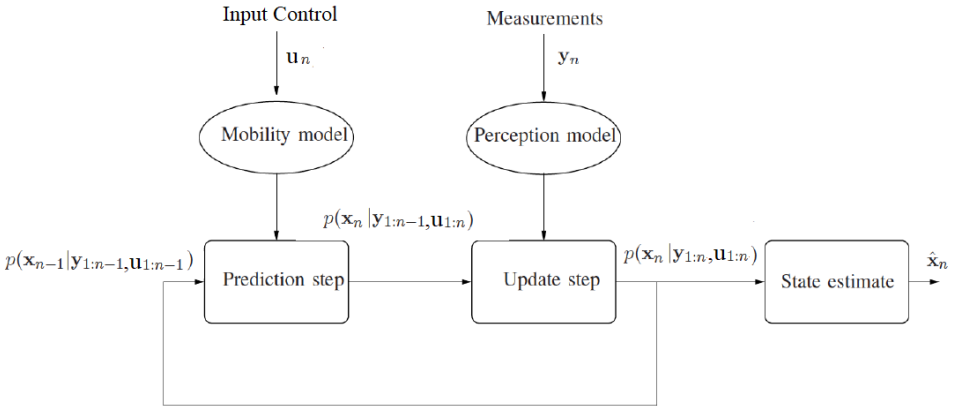

Figure 2.3: Flux diagram of a Bayesian filter [1]

- The prediction step (figure 2.4) is performed (obtained exploiting the 2.10) [8]

$$p(\mathbf{x}_n|\mathbf{y}_{1:n-1}, \mathbf{u}_{1:n}) = \int p(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{u}_n)p(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}, \mathbf{u}_{1:n-1})\, d\mathbf{x}_{n-1} \tag{2.12}$$

- The update step (Figure 2.5) is realized (obtained exploiting the 2.11) [8] [1]

$$p(\mathbf{x}_n|\mathbf{y}_{1:n}, \mathbf{u}_{1:n}) = \frac{p(\mathbf{y}_n|\mathbf{x}_n)p(\mathbf{x}_n|\mathbf{y}_{1:n-1}, \mathbf{u}_{1:n})}{\int p(\mathbf{y}_n|\mathbf{x}_n)p(\mathbf{x}_n|\mathbf{y}_{1:n-1}, \mathbf{u}_{1:n})\, d\mathbf{x}_n} \tag{2.13}$$

- Once obtained the belief of the state n, it is possible to use it to estimate the state using a chosen criterion:

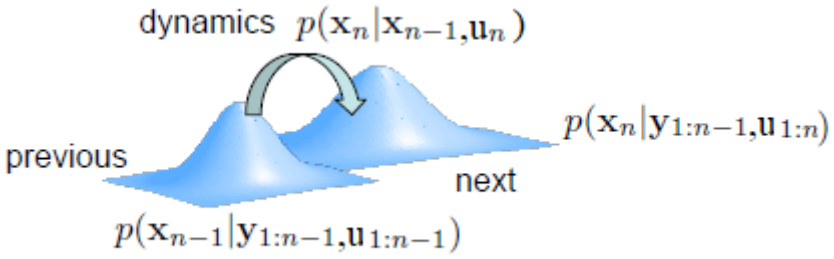

Figure 2.4: Prediction step [1]

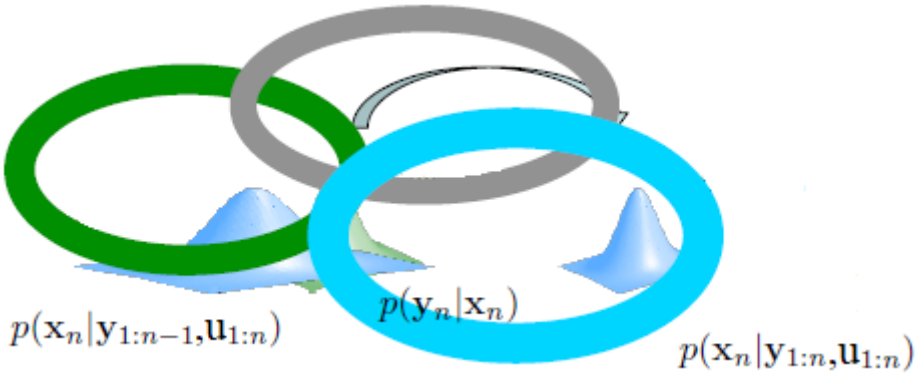

Figure 2.5: Update step [1]

– MMSE:

$$\hat{\mathbf{x}}_n^{MMSE} = \int \mathbf{x}_n p(\mathbf{x}_n|\mathbf{y}_{1:n}, \mathbf{u}_{1:n}) d\mathbf{x}_n \tag{2.14}$$

– MAP

$$\hat{\mathbf{x}}_n^{MAP} = argmax_{\mathbf{x}_n} p(\mathbf{x}_n|\mathbf{y}_{1:n}, \mathbf{u}_{1:n}) \tag{2.15}$$

–

It is well known that when the posterior distributions are Gaussian, the MAP and MMSE estimates coincide [8]. This method makes it unnecessary to recalculate equation 2.8 at each time instant; this procedure would involve the whole sequence of states and measurements in the computation. Instead, the Bayesian filters use at every step the current data to recursively update the belief.

2.3 Kalman filters

The Kalman filter (KF) is a closed form solution that is used to implement a Bayesian filter; it applies when the observation and mobility models are linear and the noise is Gaussian according to the following equations:

$$\mathbf{x}_n = \mathbf{A}_{n-1}\mathbf{x}_{n-1} + \mathbf{B}_n\mathbf{u}_n + \mathbf{w}_n \tag{2.16}$$

$$\mathbf{y}_n = \mathbf{H}_n\mathbf{x}_n + \boldsymbol{\nu}_n \tag{2.17}$$

with
$\mathbf{w}_n = \mathcal{N}(\mathbf{0}, \mathbf{Q}_n)$ process noise
$\boldsymbol{\nu}_n = \mathcal{N}(\mathbf{0}, \mathbf{R}_n)$ measurement noise

Hence the following mobility model and marginal a-posteriori PDF is obtained

$$p(\mathbf{x}_n|\mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{x}_n; \mathbf{A}_{n-1}\mathbf{x}_{n-1} + \mathbf{B}_n\mathbf{u}_n, \mathbf{Q}_{n-1}) \tag{2.18}$$

$$p(\mathbf{y}_n|\mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n; \mathbf{H}_n\mathbf{x}_n, \mathbf{R}_n) \tag{2.19}$$

It can be proven that the prediction step and the update step (2.12,2.13) can be solved in closed form and the resulting distributions are Gaussian:

$$p(\mathbf{x}_n|\mathbf{y}_{1:n-1}, \mathbf{u}_{1:n}) = \int p(\mathbf{x}_n|\mathbf{x}_{n-1}, \mathbf{u}_n) p(\mathbf{x}_{n-1}|\mathbf{y}_{1:n-1}, \mathbf{u}_{1:n-1})\, d\mathbf{x}_{n-1} = \mathcal{N}(\mathbf{x}_n; \mathbf{m}_n^-, \mathbf{P}_n^-) \tag{2.20}$$

$$p(\mathbf{x}_n|\mathbf{y}_{1:n}, \mathbf{u}_{1:n}) = \frac{p(\mathbf{y}_n|\mathbf{x}_n) p(\mathbf{x}_n|\mathbf{y}_{1:n-1}, \mathbf{u}_{1:n})}{\int p(\mathbf{y}_n|\mathbf{x}_n) p(\mathbf{x}_n|\mathbf{y}_{1:n-1}, \mathbf{u}_{1:n})\, d\mathbf{x}_n} = \mathcal{N}(\mathbf{x}_n; \mathbf{m}_n, \mathbf{P}_n) \tag{2.21}$$

Made these premises, the Kalman filter is an algorithm that calculates in a recursive way the parameters $\mathbf{m}_n^-, \mathbf{m}_n, \mathbf{P}_n^-, \mathbf{P}_n$. Practically, the calculation of the PDF is not made in every point, but only the mean values and the covariance matrices are calculated, from which it is possible to exhaustively describe a Gaussian PDF.

The Kalman Filter is realized by the following update equations related to the steps previously introduced (2.12,2.13):

- Prediction step:

$$\mathbf{m}_n^- = \mathbf{A}_{n-1}\mathbf{m}_{n-1} + \mathbf{B}_n\mathbf{u}_n \tag{2.22}$$

$$\mathbf{P}_n^- = \mathbf{A}_{n-1}\mathbf{P}_{n-1}\mathbf{A}_{n-1}^T + \mathbf{Q}_{n-1} \tag{2.23}$$

- Update step:

$$\mathbf{v}_n = \mathbf{y}_n - \mathbf{H}_n\mathbf{m}_n^- \tag{2.24}$$

$$\mathbf{S}_n = \mathbf{H}_n\mathbf{P}_n^-\mathbf{H}_n^T + \mathbf{R}_n \tag{2.25}$$

$$\mathbf{K}_n = \mathbf{P}_n^-\mathbf{H}_n^T\mathbf{S}_n^{-1} \tag{2.26}$$

$$\mathbf{m}_n = \mathbf{m}_n^- + \mathbf{K}_n\mathbf{v}_n \tag{2.27}$$

$$\mathbf{P}_n = \mathbf{P}_n^- - \mathbf{K}_n\mathbf{S}_n\mathbf{K}_n^T \tag{2.28}$$

where:

- $\mathbf{v}_n$ is called *innovation*, that is the difference between the expected measurement at the instant n, based on the belief $\mathcal{N}(\mathbf{m}_n^-, \mathbf{P}_n^-)$ obtained in the previous step, and the actual measurement y_n.

- $\mathbf{K}_n$ is the Kalman gain, that is a parameter that lets the algorithm update the state in function of the current measurement according to the reliability of the current measurement.

The Kalman filter is the optimum filter when the model is linear with additive Gaussian Noise; conversely, when the noise is not Gaussian, the Kalman filter is not in general optimum, but it represents the Best Linear Unbiased Estimator (BLUE). When the model is not linear or Gaussian it is in general no more optimum. $\mathbf{m}_n$ can be used as the point estimate at time step n, while $\mathbf{P}_n$ gives an idea of the accuracy of the estimate

2.4 Extended Kalman filters

It often happens in practical applications that the dynamic and measurement models are not linear and the Kalman filter is not appropriate. However, often the filtering distributions of this kind of model can be approximated by Gaussian distributions. The Extended Kalman Filter (EKF) is then applicable through the linearization of the non-linear models showed in 2.29 and 2.30 by using Taylor series [7]:

$$\mathbf{x}_n = \mathbf{f}(\mathbf{x}_{n-1}, \mathbf{u}_n) + \mathbf{q}_{n-1} \tag{2.29}$$

$$\mathbf{y}_n = \mathbf{h}(\mathbf{x}_n) + \mathbf{r}_n \tag{2.30}$$

where

f represents the dynamic model function

h represents the measurement model.

The stochastic translation of these models assumes Gaussian distribution for the mobility model and the perception model, in order for the EKF to be applicable as follows:

$$p(\mathbf{x}_n|\mathbf{x}_{n-1}) = \mathcal{N}(\mathbf{x}_n|\mathbf{f}(\mathbf{x}_{n-1}, \mathbf{u}_n), \mathbf{Q}_{n-1}) \tag{2.31}$$

$$p(\mathbf{y}_n|\mathbf{x}_n) = \mathcal{N}(\mathbf{y}_n|\mathbf{h}(\mathbf{x}_n), \mathbf{R}_n) \tag{2.32}$$

The linearization through Taylor series allows to approximate also the PDF of the belief into a Gaussian distribution as follows:

$$p(\mathbf{x}_n|\mathbf{y}_{1:n}) \simeq \mathcal{N}(\mathbf{x}_n|\mathbf{m}_n, \mathbf{Q}_n) \tag{2.33}$$

The algorithm that realize this approximation is described by the following prediction and update steps:

- Prediction step

$$\mathbf{m}_n^- = f(\mathbf{m}_{n-1}, \mathbf{u}_n) \tag{2.34}$$

$$\mathbf{P}_n^- = \mathbf{F}_x(\mathbf{m}_{n-1})\mathbf{P}_{n-1}\mathbf{F}_x^T(\mathbf{m}_{n-1}) + \mathbf{Q}_{n-1} \tag{2.35}$$

- Update step

$$\mathbf{v}_n = \mathbf{y}_n - \mathbf{h}(\mathbf{m}_n^-) \tag{2.36}$$

$$\mathbf{S}_n = \mathbf{H}_x(\mathbf{m}_n^-)\mathbf{P}_n^-\mathbf{H}_x^T(\mathbf{m}_n^-) + \mathbf{R}_n \tag{2.37}$$

$$\mathbf{K}_n = \mathbf{P}_n^-\mathbf{H}_x^T(\mathbf{m}_n^-)\mathbf{S}_n^{-1} \tag{2.38}$$

$$\mathbf{m}_n = \mathbf{m}_n^- + \mathbf{K}_n\mathbf{v}_n \tag{2.39}$$

$$\mathbf{P}_n = \mathbf{P}_n^- - \mathbf{K}_n\mathbf{S}_n\mathbf{K}_n^T \tag{2.40}$$

where
$\mathbf{F}_x$ is the Jacobian matrix of f
$\mathbf{H}_x$ is the Jacobian matrix of h

It has to be taken into consideration that the algorithm here discussed realizes the first order additive noise EKF and it will not work when considerable non-linearities are present. The filtering model is also restricted in the sense that only Gaussian noise processes are allowed.

The EKF also requires the measurements model and the mobility model functions to be differentiable. Moreover, the Jacobian matrices.

On the other hand, with respect to other non-linear filtering methods, the EKF is relatively simple compared to his performance and is able to represent many real cases [7].

2.5 Particle filters

When a linear model of the system and a Gaussian noise model are not suitable for the system, the utilization of an EKF becomes pointless. The same statement can be made when the PDF are multimodal or discrete. In this case the implementation of a Particle Filter (PF), also known as Sequential Montecarlo Method becomes useful.

Here the belief is represented as a set of Dirac deltas (particles):

$$Bel(\mathbf{x}_n) = \sum_i w_i \delta(x_n - x_n^i) \tag{2.41}$$

where
w_n^i are the importance factors: they are weigths that describe the probability to be in a certain state x_n at every step n.

The set of weights w_n^i is updated at every step according to the new control u_n and the new measurement y_n, starting from the old one w_{n-1}^i.

The particle filter may suffer from the weights degeneration problem as at every step all weights are multiplied by a factor less than 1, thus leading to a set of weights where all but a few particles have negligible weights. This problem is unavoidable and is shown in 2.6: in this instance, an initial a-priori PDF $p(\mathbf{x}_{n-1})$ sampled into weights, is conditioned by a new measurement y_n and leads to the resulting belief, which weights are now weaker then the former. A procedure of resampling must be actuated whenever the degradation is detected as shown in 2.7; the resampling must be performed with a major density where the weights are greater [4].

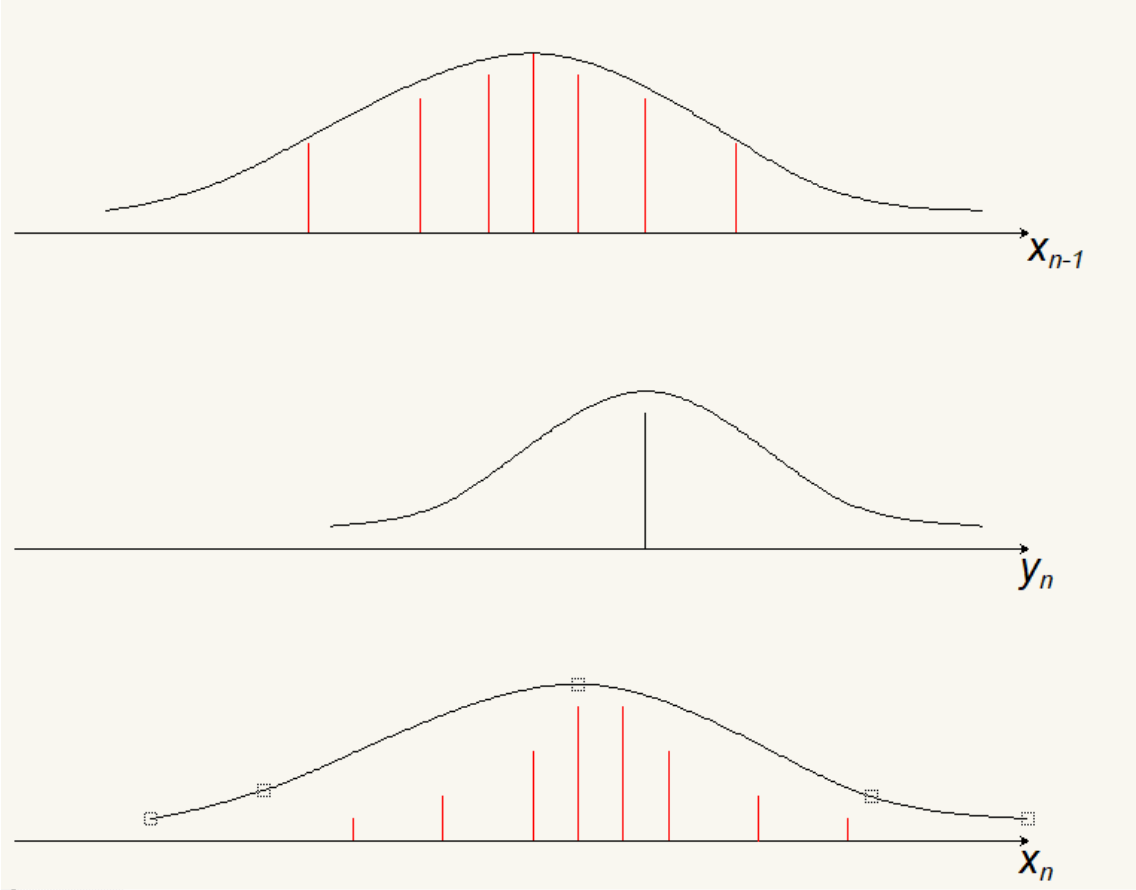

Figure 2.6: Degradation of weights in particle filters

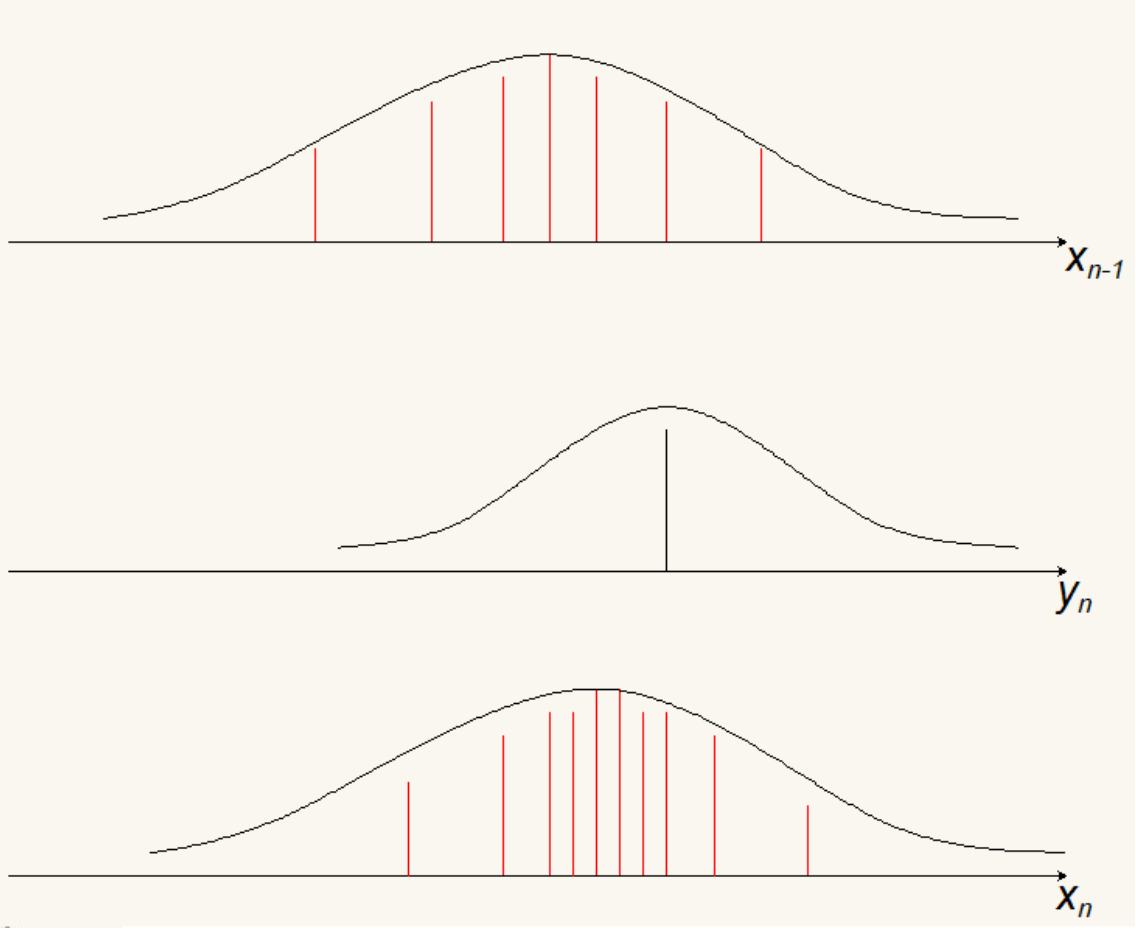

Figure 2.7: Importance resampling in particle filters

Chapter 3

Localization and mapping in dynamic physical systems

The SLAM problem (Simultaneous Localization And Mapping) is known as the issue of building, or updating, the map of an unknown environment, using the observations collected by a device that explores it, while tracking itself. Compared to the classic mapping problem in which the location of the object is known during the mapping, here the localization task has to be performed in the absence of reference nodes (often called "anchor nodes"), i.e., in the absence of a dedicated infrastructure. Generally for the resolution of this problem successive approximation methods like the Particle Filter(PF) and the Extended Kalman Filter(EKF) are adopted. The SLAM algorithms are tailored in function of the available informations and resources.

Before dealing with the SLAM problem, a general view on the separated problems regarding the mapping and the localization is here given.

3.1 The mapping problem

The mapping problem is considered as a simplified SLAM problem, in which the mobile location is assumed known (in figure 3.1, a vehicle maps the surrounding environment using the GPS to obtain his position)

The issue can be formalized as follows: the a-posteriori PDF $P(\mathbf{m}_n|\mathbf{y}_{1:n})$, conditioned on the observations $\mathbf{y}_{1:n}$, has to be estimated given the series of (known) positions of the device $\mathbf{x}_{1:n}$, in order to estimate the state vector of the system, using a chosen criterion. $\mathbf{m}_n$ is the state vector and it usually contains the positions of the landmarks representing the environment in an instant n

$$\mathbf{m}_n = [m_{xn}^{(1)}, m_{yn}^{(1)}, m_{xn}^{(2)}, m_{yn}^{(2)}, ..., m_{xn}^{(N)}, m_{yn}^{(N)}]$$

where N is the number of landmarks. The mapping problem can be treated with different approaches regarding the representation of the environment, that can be resumed in three categories:

Figure 3.1: A mobile vehicle that performs mapping using a LIDAR system (https://hu.wikipedia.org/wiki/Fájl:LIDAR_equipped_mobile_robot.jpg)

- Feature Based approach (FB): here the environment can be modeled as a collection of entities called *features*, that code all the relevant informations concerning walls, edges, corners, objects and other obstacles. The mapping problem is all about estimating the state of each feature, whose main element is the position, although it may include orientation, colour or other properties [2].

From the computational point of view, this is a fast approach that does not require a considerable occupation of memory, but since the extracted features are predefined, an a-priori knowledge of the environment structure becomes necessary. Generally, this approach is preferred when the mapping must be performed in outdoor contexts, where there are large spaces and few landmarks [3].

- Occupancy Grid Based approach (OGB): the space here is decomposed in a grid of cells, each one characterized by an occupation PDF; the latter must be used to estimate whether or not an obstacle is present. The computational burden and the necessary memory to save all the informations is greater than the FB. Moreover, to correct the mobile position, the detection of a certain number of obstacles becomes necessary, hence in presence of large spaces with very few landmarks and limited spatial scan available, this approach can enter into crisis. On the other hand, the capability to represent objects and obstacles of any shape is achieved. Generally, this approach is preferred in indoor environment [3].

- Mixed approach: the purpose of this approach is to improve the performances when the mapping of environments with a few features and indoor contexts are both needed; at every step, based on the current measures, the state is updated and one of the previous approaches can be implemented. In this way, the representation of a mixed environment is possible without suffering too much from the drawbacks of each technique. [3].

3.2 The localization problem

The localization problem consists in identifying the position of a device that is situated inside a map that is assumed known (figure 3.2). This issue can be formalized as follows: the a-posteriori PDF $P(\mathbf{x}_n|\mathbf{y}_{1:n})$, conditioned from the observations $\mathbf{y}_{1:n}$, has to be estimated given the series of landmarks of the environment $\mathbf{m}_{1:n}$ assumed known and typically named anchors or reference devices (RDs), in order to estimate the position of the device, using a chosen criterion.

$\mathbf{x}_n$ is the state vector, that mainly contains the positions of the device and, possibly, its speed and orientation at the instant n

$$\mathbf{x}_n = [x_{xn}, x_{yn}, v_{xn}, v_{yn}, o_{xn}, o_{yn}]$$

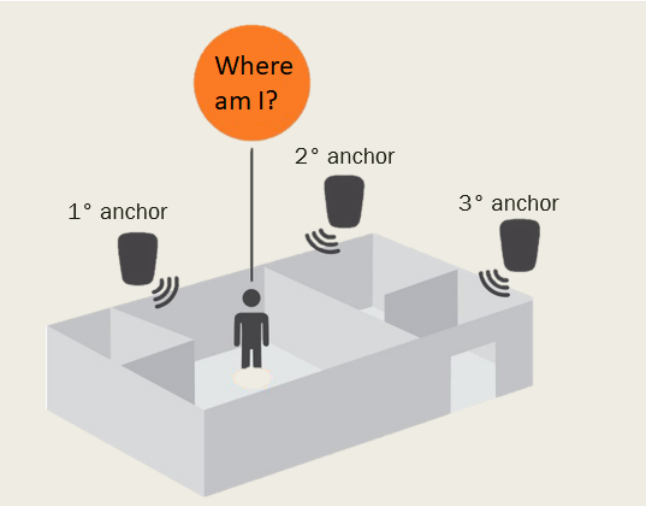

Figure 3.2: Representation of the localization problem

The localization problem can be treated with different approaches; the main methods are the following:

- Triangulation: this type of method exploits the geometric relationship between the mobile device (MD) and the reference devices (RDs). For this purpose, different properties of signals exchanged among them can be used [4]:
  - Received Signal Strength (RSS)
  - Time Of Arrival (TOA)
  - Time Difference Of Arrival (TDOA)
  - Phase Difference Of Arrival (PDOA)
  - Angle Of Arrival (AOA)
- Fingerprinting: this method involves measuring position-dependent fingerprints of the signal at known positions to construct a fingerprint database of an environment that can be used to localize and track the MD. Generally, it is composed of two steps: the first one is to collect the position-dependent fingerprints into a database, while the second deals with matching the measured fingerprints of the MD with those of the database through a pattern-recognition algorithm, in order to locate and track the MD. It can be adopted when the line of sight (LOS) path is blocked, which typically makes it difficult to apply geometric relationships between MD and RDs.

- Proximity: proximity-based techniques rely on a dense grid of RDs previously deployed at known positions in an environment. Each RD has got its own detection radius. Once the MD enters an RD's detection region, the MD and this RD are connected; according to this connection, the MD is localized. This kind of technique requires a simple receiver architecture, but also a pre-deployment of dense grids of RDs. In advance, when multiple RDs detect the MD, more sophisticated smoothing method are needed, even though the localization accuracy could be poor compared to other methods.

- Self-measurements: Unlike the types of measurements mentioned above that rely on an infrastructure, self-measurements can be collected by a stand-alone MD with an Inertial Measurement Unit (IMU). Typically, an IMU is composed of a 3D gyroscope and a 3D accelerometer, which provide angular velocity and linear acceleration, respectively. Given the initial position and the MD's velocity and orientation information, the standalone MD can be localized and tracked by integrating the measured angular velocity and linear acceleration with the Bayesian filters. However, the measurement errors of the IMU will be integrated and propagated unbounded over time, which limits stand-alone inertial localization and tracking with a low-cost IMU [4].

3.3 The SLAM problem

The SLAM problem merges the previous presented issues of building the map of an unknown environment and localizing the mobile device within it. Both the trajectory of the MD and the location of all landmarks are estimated online without the need for any a-priori knowledge of location (figure 3.3).

The formalization of this problem follows: considering a mobile robot moving through an environment and taking relative observations of a number of unknown landmarks, at a time instant k, the following quantities are defined:

- $\mathbf{x}_n$: the vector that describes the location and orientation of the vehicle at time n

$$\mathbf{x}_n = [x_{xn}, x_{yn}, v_{xn}, v_{yn}, o_{xn}, o_{yn}]$$

- $\mathbf{u}_n$: the control vector, applied at time n-1 to drive the vehicle to a state $\mathbf{x}_n$ at time n

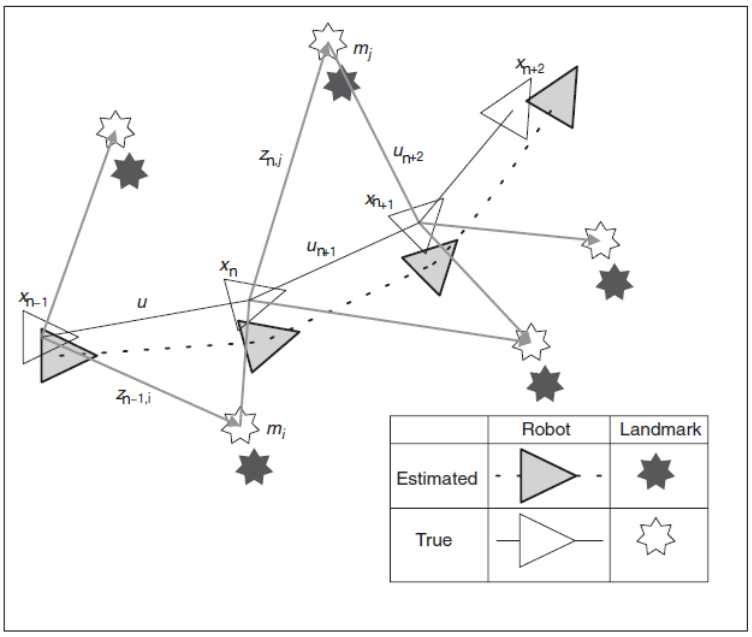

Figure 3.3: Representation of a SLAM problem: both location of the MD and landmarks are estimated[5]

- $\mathbf{m}_n$: a vector describing the locations of the landmarks whose true location is assumed time invariant at a time n

$$\mathbf{m}_n = [m_{xn}^{(1)}, m_{yn}^{(1)}, m_{xn}^{(2)}, m_{yn}^{(2)}, ..., m_{xn}^{(N)}, m_{yn}^{(N)}]$$

- $\mathbf{y}_n$ the vector of the observations taken from the vehicle related to the locations of the landmarks at time n

$$\mathbf{y}_n = [y_{xn}^{(1)}, y_{yn}^{(1)}, y_{xn}^{(2)}, y_{yn}^{(2)}, ..., y_{xn}^{(N)}, y_{yn}^{(N)}]$$

- $\mathbf{x}_{1:n}$: the history of vehicle locations

$$\mathbf{x}_{0:n} = \mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_n$$

- $\mathbf{u}_{1:n}$: the history of control inputs

$$\mathbf{u}_{1:n} = \mathbf{u}_1, \mathbf{u}_2..., \mathbf{u}_n$$

- $\mathbf{y}_{1:n}$: the history of all the observations performed on the landmarks

$$\mathbf{y}_{1:n} = \mathbf{y}_1, \mathbf{y}_2, ..., \mathbf{y}_n$$

The SLAM problem requires the computation of the a-posteriori PDF

$$P(\mathbf{x}_n, \mathbf{m}_n | \mathbf{y}_{1:n}, \mathbf{u}_{1:n}, \mathbf{x}_0) \tag{3.1}$$

that is conditioned on the observations $\mathbf{y}_{1:n}$, the control inputs $\mathbf{u}_{1:n}$ and the initial state $\mathbf{x}_0$ without any information regarding his location or the landmarks of the environment. The aim is to estimate, using a chosen criterion, the state vector of the system, which in this case is the union of the vectors $\mathbf{x}_n$ and $\mathbf{m}_n$ and contains both the estimated device location and map information [5]. To this purpose, many statistical algorithms can be employed based on Bayesian Filters, in particular Extended Kalman Filters (EKF) and Particle Filters (PF).

Chapter 4

State of the art related to the SLAM problem

The autonomous navigation in unknown spaces requires the capability of a mobile robot to be aware of his location and the locations of other places of interest nearby. This is one of the main skills that a mobile autonomous robot should achieve. Many devices base this capability in prior reliable mapping or positioning information such as the Global Positioning System (GPS).

In first place, without a notion of location, a robot is limited to reactive behaviour based solely on local stimuli and is incapable of planning actions beyond its immediate sensing range [9]. The quality of localization is classically driven by the reliability of the provided map: in the absence of the latter, dead-reckoning methods would quickly drift over time, while the use of a map of the environment allows the reset of the localization error by revisiting the known areas: this concept is known as *loop closure*.

The resolution of the SLAM problem enables a robot to navigate any unknown environment without the need for prior map or location information; this allows the approaching to a more independent behaviour of the robot, where all high level navigation operations such as goal reaching, region coverage, exploration and obstacle avoidance are carried out without external support.

This brings to the main reason for which the SLAM is needed: the scenarios in which a prior map is not available need the simultaneous reconstruction of both environment and location[10].

4.1 Anatomy of a SLAM system

The architecture of a SLAM system can be divided in a front-end component and a back-end component. The front-end role is to abstract sensor data into estimation models like those described in Chapter 1, while the back-end performs inference on the abstracted data produced by the front-end [10]. This architecture is shown in figure 4.1.

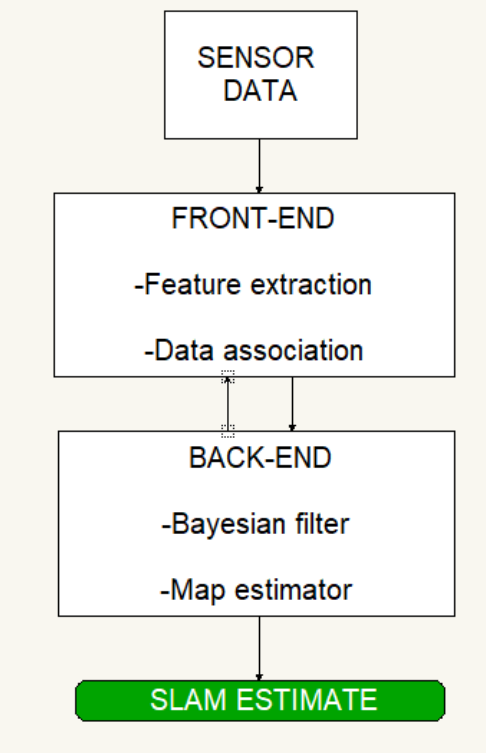

Figure 4.1: Front-end and back-end in a typical SLAM system. The back-end can provide feedback to the front-end for loop closure detection and verification [10]

4.1.1 Back-end

The back-end component of the system must provide an estimation of the state based on the belief defined in Chapter 1 (2.9): as mentioned before, the state typically includes the estimated pose of the robot (location and orientation) and the position of landmarks in the environment; the belief is generally obtained using an improved version of the EKF and PF presented in Chapter 1.

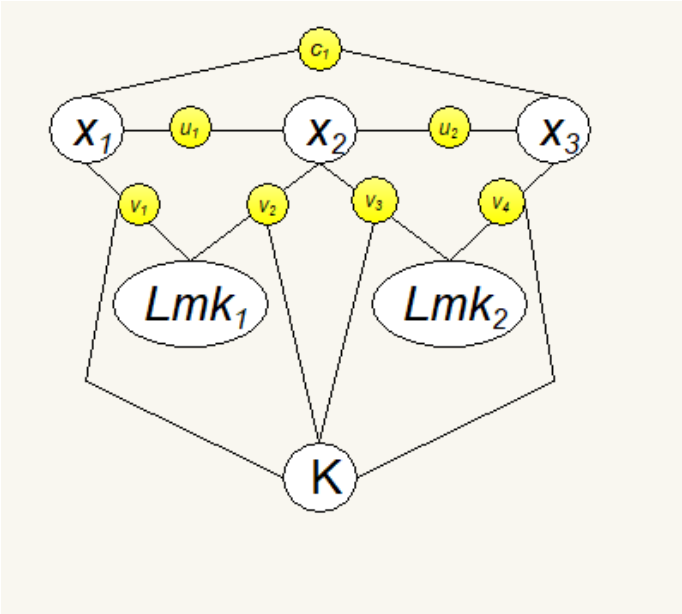

Figure 4.2: Representation of the SLAM with factor graphs [10]

The EKF is still largely diffused in SLAM applications because of his linear Gaussian assumptions in the representation of both the measurement and motion model.

When linear measurement and motion model are not available and the noise of the system can not be assumed Gaussian, the PF, based on Monte Carlo sampling, must be used. The main drawback is that the application of particle filters on the high dimensional growing state-space of the SLAM problem is unfeasible; however, through Rao-Blackwellization, a joint state can be partitioned according to the product rule, obtaining the so called Rao-Blackwellized Particle Filter (RBPF); as an instance, here is a joint PDF of 2 states:

$$p(\mathbf{x}_1, \mathbf{x}_2) = p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_1) \tag{4.1}$$

In this way, if $p(\mathbf{x}_2|\mathbf{x}_1)$ is available,only $p(\mathbf{x}_1)$ needs to be sampled, simplifying the estimation of the joint state; this technique allows the factorization into a vehicle component and a conditional map component of the joint SLAM state, that improves the speed of the algorithm as the map is represented as a set of independent Gaussian processes [5]. The RBPF

Currently, many approaches formulate SLAM as a MAP estimation problem, often using the formalism of *factor graphs* [10]: the dependence of poses and landmarks, both represented as circles, from each other are put in evidence with the aid of factors related to:

- Odometry constraints: if they link two poses
- Sensor observations: if they link a pose and a landmark
- Loop closure: if an event of loop closure is detected

This can be considered a more general approach with respect to Bayesian filtering based on the message passing of beliefs.

In figure 4.2, is shown a simple factor graph representation that highlights the main dependence of different variables at consecutive time steps, where:

- $(x_1; x_2; ...)$ is the sequence of robot poses
- $(Lmk_1; Lmk_2; ...)$ is the sequence of landmarks positions
- $(u_1; u_2; ...)$ is the sequence of motion controls
- $(v_1; v_2; ...)$ is the sequence of measurements
- K is the set of intrinsic calibration parameters
- $(c_1, c_2, ...)$ is the set of closure loops

The main advantages brought by the factor graph representation of the SLAM problem can be resumed in:

- Simplification of the view, that can be appreciated in figure 4.2
- Generality of the problem, enabling the modeling of complex problems through interconnections of heterogeneous variables and factors; this approach

4.1.2 Front-end

Generally, is hard to use all the sensor measurements for the MAP estimation because they should all be expressed as analytic function of the state. This justifies the presence of a front-end module that extracts only the relevant features from the sensor data [10].

The front-end also deals with the problem of associating each measurement to the correct landmark; this problem is commonly addressed as *data association*.

4.1.3 Odometry

According to the *odometry* principle, if a robot is aware of his position and his future movements than he is aware of his pose. Knowing the motion model of the robot, his initial pose and a global reference coordinate system, localization is doable.

As an instance, wheel odometry uses sensors mounted on wheels to measure the travelled road and estimate the trajectory. The main limit of this method is the accumulated measurement error that becomes dominant over time causing estimated robot pose to drift from its actual location. This effect is showed in figure 4.3. Another source of error that should not be underestimated is wheel slippage in uneven terrain or slippery floors: if the world is modelled as a plane, a non-flat ground surface may cause estimated and real position to take different directions, while slippage can be responsible of mismatches of actual and estimated travelled road.

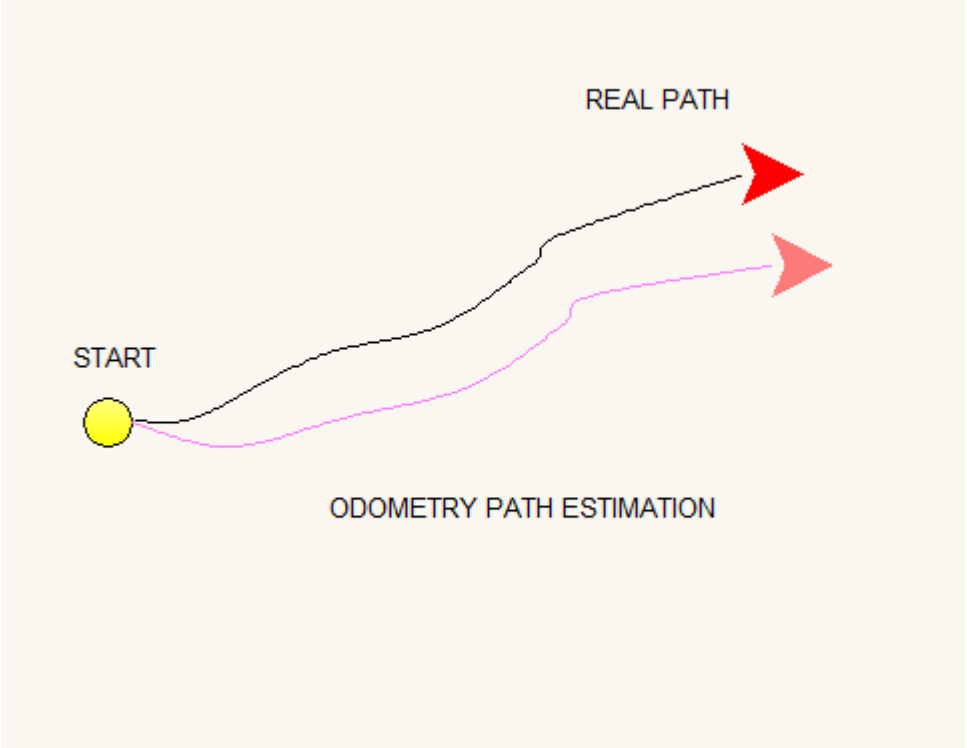

Figure 4.3: Representation of the accumulated odometry measurement error

This problems are significantly reduced through the use of an Inertial Measurement Unit (IMU): it is a device that exploits a triaxial gyroscope and a triaxial accelerometer to track acceleration, deceleration and position

(through a double integration of the acceleration), of the vehicle in which is mounted. This makes possible the detecting of changes in direction and velocity, allowing an augmented odometry. Conversely, IMU fails in observation of low-frequency faults such as model biases, which might lead to error accumulation. . For this reason, IMU is suited for the detection of low term motion, between the map observations, but is not sufficient for longer term[9]. IMUs are widely used in Inertial Navigation Systems (INS), that are aid systems for aero-mobile navigation mainly made of an IMU, sensors and a CPU, and in advanced automotive and motorcycle industry and in robotics.

Visual Odometry is the current state of the art and uses the images seen from a single or multiple camera to estimate the ego motion of the vehicle in which is mounted. It is demonstrated that this system performs better than wheel odometry or LASER scanners [12].

4.1.4 Sensors in SLAM

In addition to the already mentioned sensors that implement odometry, there are different kind of sensors that are used to perform SLAM. They exploit interactions with landmarks to estimate the state of the system, including pose of the robot and pose of landmarks.

The type of sensor used is what mainly differentiate a SLAM system and his applications from another one. Another way to see the matter is the following: both SLAM sensors and odometry perform sensing activity, each one on a different subject, then they cooperate fusing the obtained data for the achievement of the same goal.

As previously mentioned, a SLAM technology depends on the type of sensor implemented; the main branches of the current state of art are the following and are deepened in the next section:

- SONAR SLAM
- RADAR SLAM
- LASER SLAM
- Visual SLAM

4.2 The main SLAM technologies

SLAM requires the simultaneous reconstruction of the environment and the pose of a robot inside it. The context in which SLAM is needed can influence

the choice of the technology. As an instance, in undersea environment the main reliable technology is SONAR due to the distortion that water imprints to light. A glance of the main technologies developed until now follows.

4.2.1 SONAR SLAM

SONAR means Sound Navigation And Ranging and is a system which exploits an acoustic sensor to perform the propagation of ultrasounds and the following reception of their reflected version whether an obstacle is present.

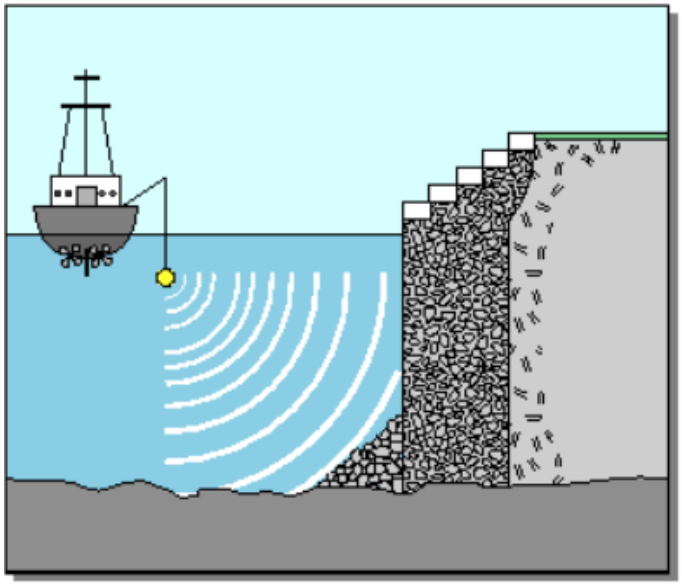

Figure 4.4: SONAR systems are commonly used by ships to probe the seabed

Mobile robot navigation and mapping is far more difficult using SONAR sensor, instead of a LASER one as an instance. A reason is the modest spatial resolution obtained using the former in comparison to the latter; another one is that SONAR is characterized by an inferior degree of accuracy with respect to LASER; last but not least, is the lower speed of response with respect to the other sensors here presented.

Despite these difficulties, SONAR SLAM is still interesting to research for some reasons. SONAR sensors are less expensive than LASER scanners, as an instance, actually ultrasonic sensors are the cheapest available source of spatial sensing for mobile robots. Furthermore, they suffer of lower signal attenuation in underwater environment, where visual and LASER sensors are ruled out from this point of view due to their high frequency signals

[13]. Thus, SONAR sensors become the ideal sensor with whom an AUV (Autonomous Underwater Vehicle) can perform SLAM.

SONAR SLAM systems exploit transceivers characterized by low frequency sound waves which allow deep penetration of water (10-150 m), while high frequency signals like those of LASER are highly attenuated. Time of flight methods are used to localize the landmarks [14].

4.2.2 RADAR SLAM

The meaning of RADAR is Radio Detection And Ranging and it represents a system that employs radiating electromagnetic waves (EM waves) to let a transceiver detect whether they are reflected from an obstacle. In figure 4.5 is shown the exchange of two RADAR signals for the identification of an airplane. The working principle is the same as SONAR, except for using EM waves instead of acoustic waves.

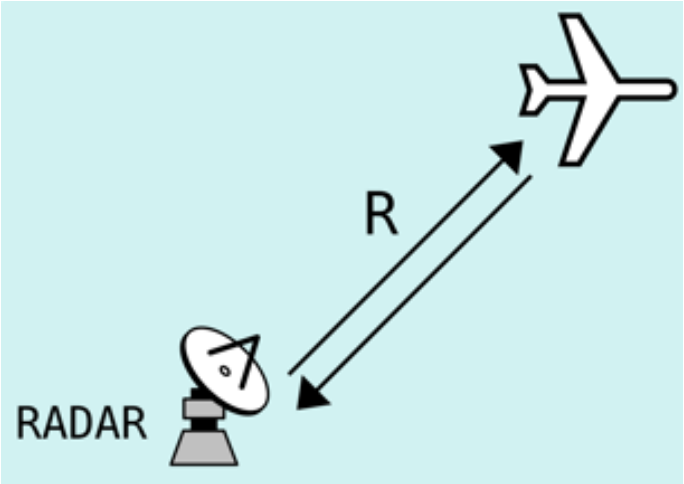

Figure 4.5: RADAR systems are widely used for aerospace purposes

This technology enables SLAM in situations where visibility is compromised such that Vision and LASER systems are not suitable; an example is an emergency scenario like a fire, where dust and smoke are present and can impede to individuate not only walls and objects but also unconscious people. Millimetre Wave RADAR SLAM (mmW RADAR SLAM) technology and UWB RADAR SLAM technology are now the main branch in development in RADAR area [18].

Many works certify the robustness of this technology; to mention one, in [19] is showed an approach that exploits the specular reflections of EM waves on flat surfaces in order to localize a mobile user using the reflections coming

from just one reference node. The concept of virtual anchor, which is the mirror image of a physical anchor, is then used in order to use the multipath components to do the SLAM estimate.

Under study is also the possibility to perform SLAM using Wi-Fi signals, considering access point as landmarks, with the purpose of localization inside a building. The main obstacle is that Wi-Fi standard, as well as other possible suited technologies like RFID, NFC and ZigBee to mention some, were not created with localization or mapping purpose [15] [16] and hence they typically provide measurements scarcely related to distance or position. The *crowd-sensing* concept can however come to help: this principle is due to the consideration that nowadays most people are provided with a smartphone equipped with a large bunch of sensors (light, temperature, pressure, sound, acceleration and magnetism can be sensed) [8].

The idea behind crowd sensing is to mitigate the scarce accuracy of measurements through the exploitation of a huge amount of measurements shared among users.

Mentioned this, let us have a look at main RADAR technologies.

Ultra Wide Band SLAM

Ultra Wide Band (UWB) RADAR SLAM Technology has grown since 2002, when Federal Communications Commission opened up 7,5 GHz of spectrum (from 3,1 to 10,6 GHz) to be used by UWB devices.

This system exploits narrow time-domain pulses, of a duration in the order of a nanosecond, to modulate a signal spreading his spectrum over a wide frequency band larger than 500 MHz. The main advantage of this method is the higher resolution in estimation of the time of flight, which translates in ranging estimation with few centimetre accuracy [15].

An interesting example of UWB SLAM system is presented in [17] where the implementation of a small antenna array of bat-type, which is made of two receiving antennas and a transmitting one allows more advantages, like the individuation of different landmark types.

Millimetre Waves SLAM

In the last decade, mmW RADAR SLAM has gained popularity in research because where low frequency radio signals have a wide beam width, which reduces the resolution in detecting the obstacles, mmWs allow narrow beam shaping, thus augmenting the resolution in detecting obstacles with respect to lower frequency radio signals.

mmWs belong to the band of radio waves with frequency going from 30 to 300 GHZ. This band is commonly known as Extremely High Frequency range and their wavelength ranges from 10 to 1 mm.

In a SLAM application, an array of antennas can be used to achieve a determinate angular resolution through use of mmW; the same can be done with UWB signals, but since the wavelength is lower, the compactness for the achievement of the same result is greater.

mmWs are subject to a high atmospheric attenuation which reduces the range and strength of the waves. Thus, they are limited in a communication range of about 1 kilometre. mmW RADAR SLAM signals are able to penetrate many objects and can provide information regarding distributed targets appearing in a single observation[16].

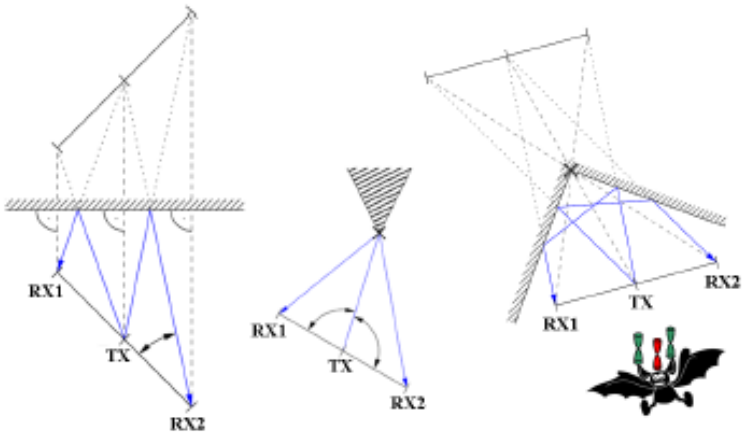

Figure 4.6: The detection of different types of landmarks is possible using a bat-type array of antennas [17]

4.2.3 LASER SLAM

The LASER working principle is the same of SONAR and RADAR and differently from the latter, here EM waves are in the range of UV, visible or infrared; this allows the reflection of waves from very small objects in the order of the wavelength used.

LASER Rangefinders also use time of flight and phase-shift techniques to measure distances. The high speed and accuracy of LASER rangefinders enable robots to generate precise distance measurements. This contributes to the significant popularity of LASER rangefinders in solving SLAM problems

since they are capable of obtaining robust results in both indoor and outdoor environments. A LASER scanner is the best sensor for extracting planar features (like walls) due to the dense range data provided. The LASER sensor is referred to as *LIDAR*, which means LASER Imaging Detection And Ranging. However, the cons of this technology are the previously mentioned difficulty in using it with non-visibility conditions, the weakness to other sources of lights, its required power consumption and its cost [11].

4.2.4 Visual SLAM

The last years have seen increasing interest in visual SLAM technologies. This is due not only to the great amount of available visual information related to the environment, but also to the low-cost of video sensors with respect to LIDARs. Visual uses pixel information to detect the angles formed by the vision direction of the camera and the line joining landmark and sensor positions.

Visual SLAM can intrinsically provide a wide range of landmarks types, but the data coming from images must be processed and selected by the front-end part of the SLAM system. This translates into a higher computational cost and to an increased complexity of algorithms that extract the main features. Thanks to the advances in CPU and GPU technologies, the real time implementation of the required complex algorithms are no longer quite a problem [12].

The figures 4.7 and 4.8 show the results of the implementation of a visual SLAM system.

4.3 Main challenges

Despite all progresses made by SLAM community research in the last decades, some problems still need solutions providing better performances. An overview of the main difficulties follows.

4.3.1 Computational Complexity

The SLAM problem is intrinsically a complex problem, due to the required estimation of a joint state composed of the robot pose and landmarks locations. Computational efficiency becomes important for the scalability of the problem: the robot performing SLAM must be able to operate over an extended period of time and to explore new spaces without letting the required memory grow unbounded [10].

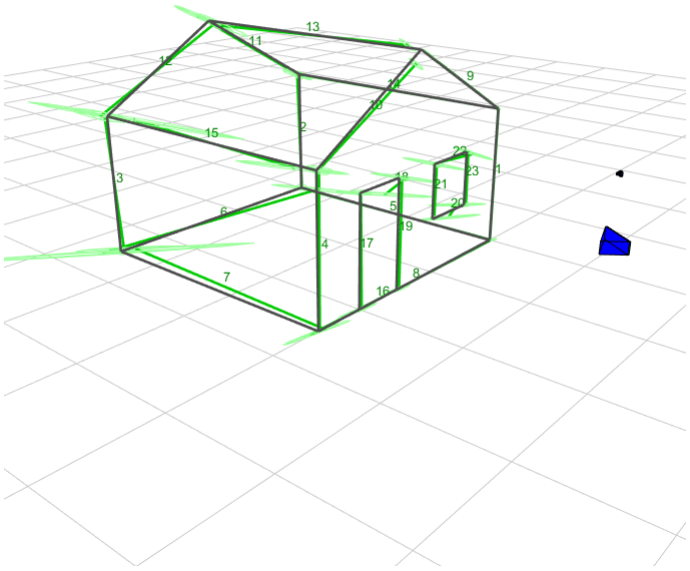

Figure 4.7: Here is showed the simulated reconstruction of the landmarks of a house through a vision SLAM system

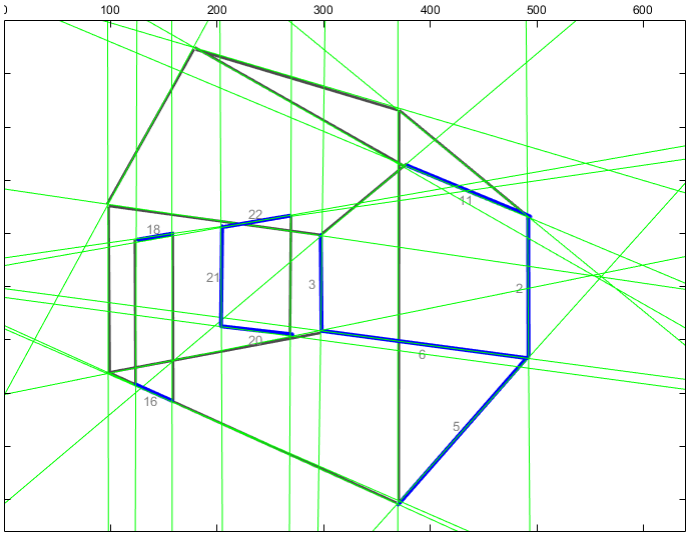

Figure 4.8: The figure shows the view of the camera performing vision SLAM

4.3.2 Data Association

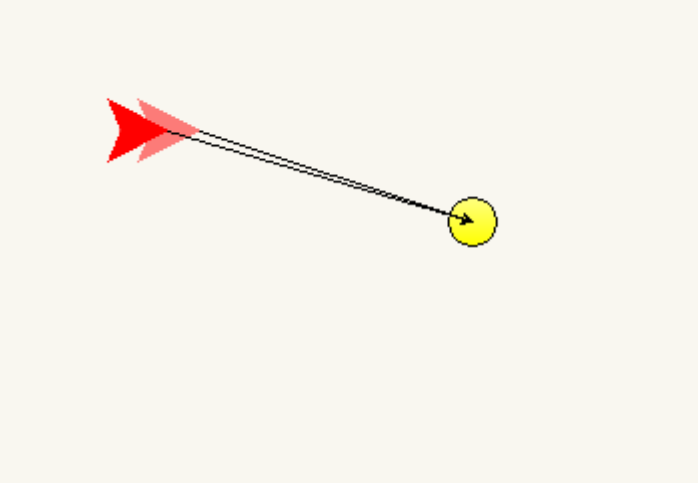

Figure 4.9: Short term data association must recognise the measurements related to the same landmark in successive instants

As previously introduced, data association is one of the critical issues in SLAM implementations. It consists in associating the measurements to the correct landmarks. Outliers, which are false positive landmarks, can degrade the estimate, which in turn degrades the ability to individuate them [10].

The problem can be subdivided in two submodules:

- Short term data association (figure 4.9): it deals with the fact that the sampling rate of the sensor is faster than the dynamics of the robot, thus making the last features observed very close to the current ones.
- Long term data association (figure 4.10): it is more complicate because it aims at detecting and validating the loop closure events. A brute-force approach that tries to compare all new features with the older becomes numerically impracticable as the number of landmarks grows up. At the same time, a single wrong data association can induce divergence into the map estimate [6].

The data association problem becomes harder when the landmarks look different from different viewpoints [6].

4.3.3 Environment representation

Formerly, SLAM environment representation was supposed to hold as simple geometric primitives such as points, lines or circles. This hypothesis does

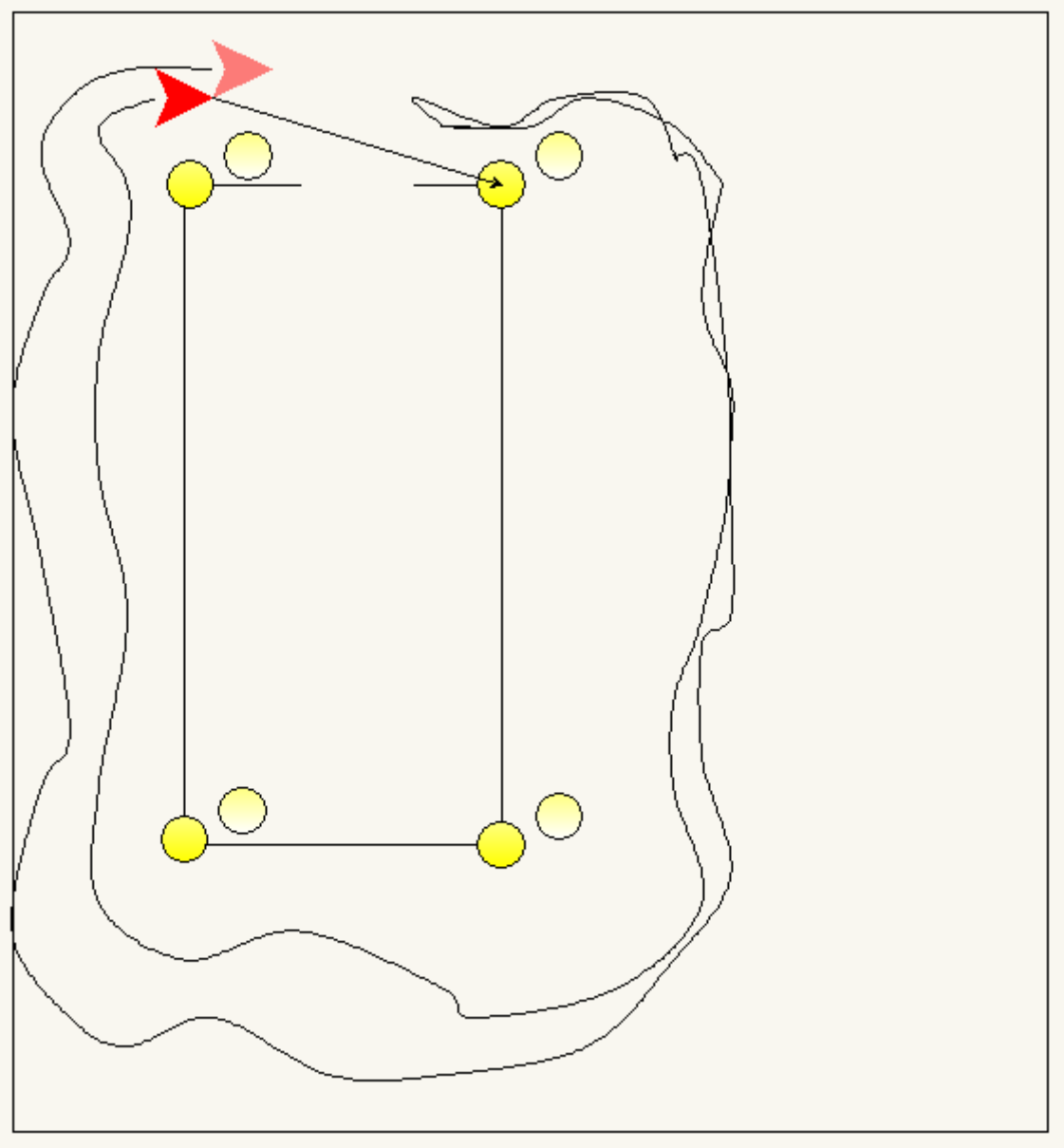

Figure 4.10: Long term data association must provide loop closure for landmarks that are being revisited

not hold in complex and unstructured environments like subsea, outdoor and underground.

The previously introduced (Chapter 2) characterization as feature-based maps and occupancy grid maps appears to be the simplest geometric environment modelling in the 2D case. In the 3D case, the complexity is increased as vehicle motion and observation model must be generalized adding a dimension. The situation is complicated by the fact that the assumption of a static world, that is often taken, does not always hold as sometimes landmarks are not fixed. The mapping in the long term or large spaces underlines this problem, since a change in the environments becomes more probable [10] [6].

Furthermore, when vision SLAM is employed, the question whether using semantic or topological mapping opens up; while topological mapping is about recognising a previously seen place, the semantic mapping deals with classifying the place according to a pre-recorded semantic database [10]

4.3.4 New frontiers for SLAM

The development of new algorithms and the availability of more advanced sensors have always been the main trigger for the progress in SLAM. Applications such as autonomous cars or augmented reality has taken off respectively due to progresses in LASER and vision systems [10].

New sensors in the vision field are now under study by research community:

- Range cameras: they are light-emitting depth cameras that works according to different principles, such as structured light, time of flight, interferometry, or coded aperture. While structured light cameras work by triangulation and use different perspective illumination of an object to deduce his shape. This kind of camera is provided with his own light source, thus making them work in dark and untextured scenes.

- Light-field cameras: they record not only the intensity of the light impacting the single pixel, but also the direction of the ray. Light-field cameras have different advantages with respect to standard cameras, such as depth estimation, noise reduction, video stabilization, isolation of distractors, and specularity removal. However, the manufacturing cost is responsible of their low resolution, still less than a megapixel in the commercially available ones.

- Event-based cameras: they are basically different from standard cameras, because the images are not sent entirely in fixed frame rates, but

only the local pixel-level change is sent when a movement in the environment occurs. This makes possible the increase of dynamic range, update rate and the decrease of power consumption, latency. The requirements of bandwidth and memory are also reduced [10].

Chapter 5

MATLAB implementation of different SLAM technologies

In this chapter, the results of simulated implementation of SLAM through different technologies are showed. For this purpose, I exploited two different MATLAB toolbox:

- Visual EKF-SLAM Toolbox: developed by Joan Solà in 2013, it implements an EKF algorithm simulating a 3D environment
- LIDAR EKF-SLAM Toolbox: developed by Tim Bailey and Juan Nieto in 2004, it can be used to simulate the implementation of an EKF algorithm in a 2D environment

The performance of SLAM is evaluated when different sensing technologies are exploited, though the same 2D context has been considered through executing the visual SLAM with landmarks and robot sensor placed in the same plane; in this way the problem can be approximated as a 2D one, such as LIDAR toolbox.

The chosen context is part of the corridor on the 3rd floor in Cesena Campus of the University of Bologna, shown in figure 5.1.

During the simulation, the robot follows a closed path (round) along the corridors. The rounds performed by the robot are 15 in each technology: the choice of this high number is made to simulate the crowd-sensing concept. If we imagine that every round corresponds to a different robot, then we can think of 15 different robots performing SLAM in successive times.

The results are finally reported in order to have a comparison between different technologies at different conditions.

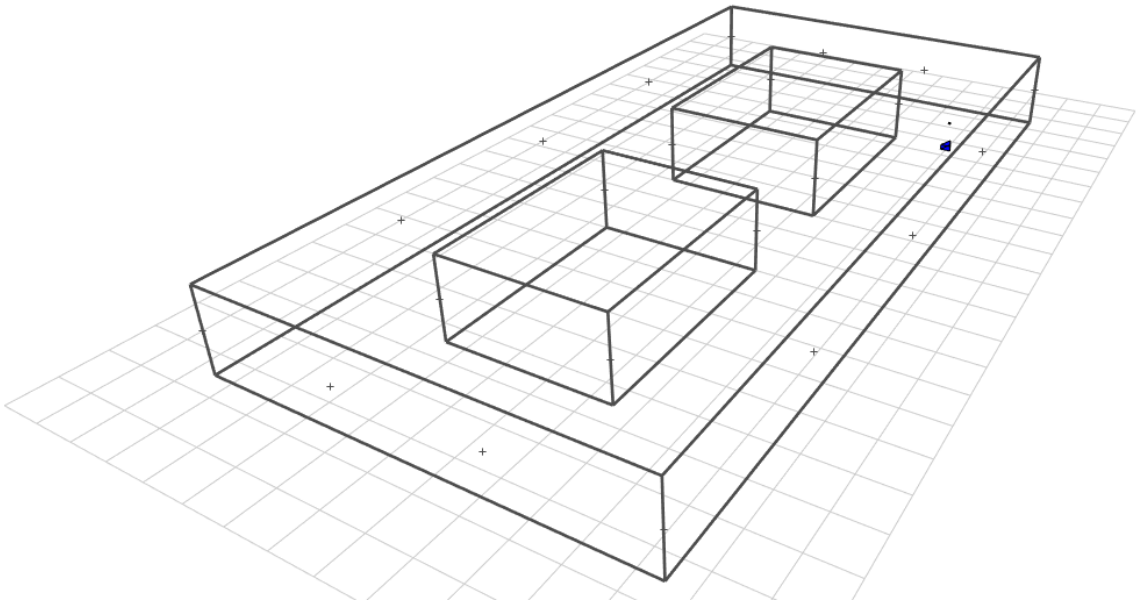

Figure 5.1: The corridor scenario simulated in Visual SLAM

5.1 Visual SLAM toolbox

The visual SLAM toolbox is an open source package that attempts to simulate an EKF-SLAM algorithm in a given environment using a visual SLAM technology implementation.

Since it simulates a visual-SLAM technology, the measurement model of the system is characterized by the choice of the sensor, which is between the followings:

- Omnidirectional camera *omniCam* with resolution 1280x800
- Pinhole camera *pinHole* with resolution 640x480

The motion models available in the package are the following:

- A constant speed motion model *constVel*
- A circular uniform motion model *odometry*

Regarding the landmarks that here can be implemented, the choice is between:

- Points: through the call to the function, called *thickCloister.m*, which generates a cluster of landmarks geometrically distributed through a 3D square ring volume, as shown in figure 5.2.
- Lines: through the call to the function, called *house.m*, which reproduces the shape of a house, as shown in figure 5.3.

5.1.1 Default main script *slamtb*

The package main file is called *slamtb.m* and it can be viewed in Appendix A. It initializes the environment through the call to the function *userDataPnt.m*, in which the choice of the sensor and the landmarks distribution can be made by editing the code following the guidelines. The choice of line landmarks, instead of point ones, can be made calling the function *userDataLin.m*.

In *userDataPnt.m*, not only the settings of the robot and sensor installed on it can be edited, but also the simulation and estimation options, ranging from standard error of sensor and number of landmarks initialized by the algorithm, to the choice of the type of landmarks that the algorithm must recognize.

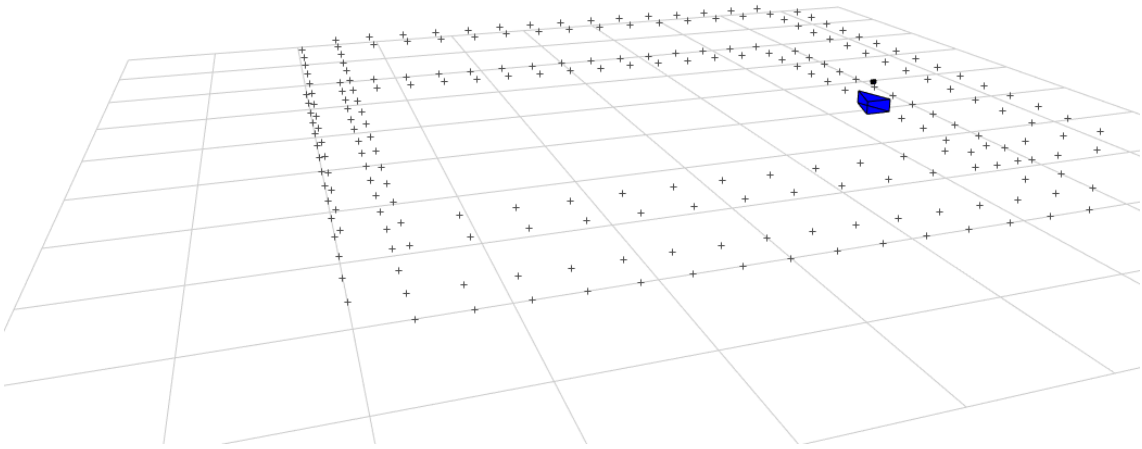

Figure 5.2: Simulation of landmarks as points in Visual SLAM toolbox

An important detail that will become important later to recreate the environment, is that even though only one type of landmarks can be initialized for recognition by the algorithm, lines and points can both be implemented. This allows the realization of buildings which contains the landmarks that needs to be detected.

Once initialized the environment, the main structure of the algorithm is initialized, including robot, sensor, landmarks and raw data structures.

To get a graphic support, also figures structures for the real-time plotting of the SLAM estimation progress are initialized.

Once finished the initialization phase, a code is repeated periodically through the use of a *for* structure. It implements:

- The motion of the robot
- The estimation of landmarks and robot pose through an EKF algorithm
- The update of the visual information on SLAM estimate

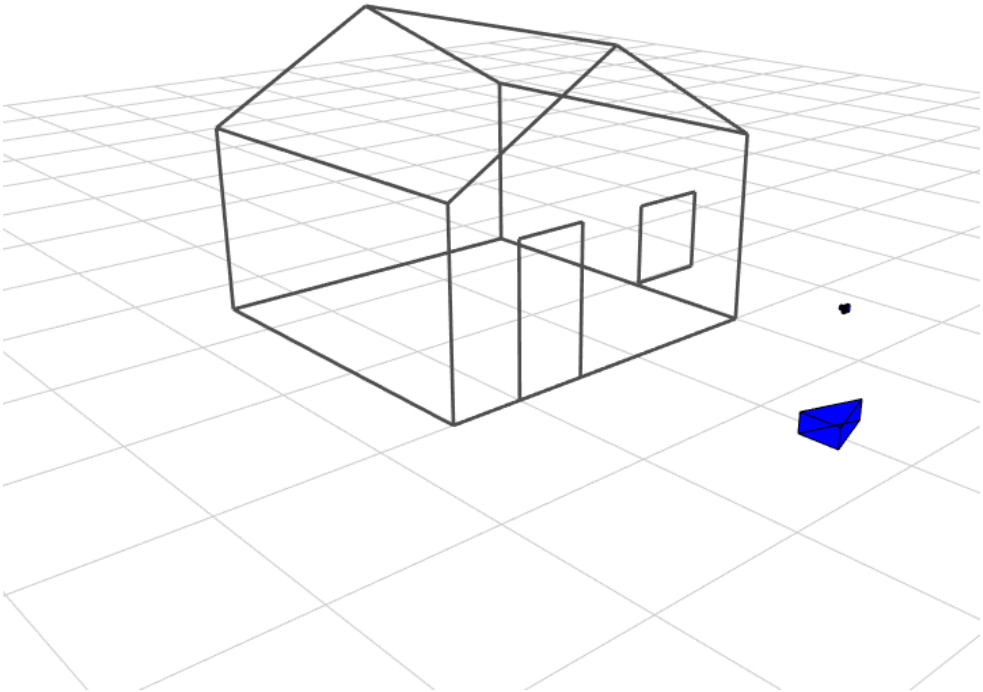

Figure 5.3: Simulation of lines constituting a house in Visual SLAM toolbox

5.1.2 Realization of the SLAM estimation in the considered scenario

Environment

As previously stated, the context that I want to simulate is different from the available ones; for this reason I have made some changes in the main functions and scripts.

The landmark type chosen is the point, so the initialization of environment is done editing the script *userDataPnt.m* into the script *userDataCorridor.m*, which recreates the shape of the corridor and collocates the point landmarks where desired; this code can be seen in Appendix A. The point landmarks that I have chosen represent the edges, the corners and the doors.

To realize the shape of the corridor, I have edited the function *house.m* to build a new function called *Corridor_Ing_Ces.m* (shown in Appendix A). According to some constants related to the shape of the corridor like height, length and width, this function builds the corridor as shown in 5.1.

The script also initializes the initial position and orientation of the robot and the sensor, together with several other parameters that are not here specified for coinciseness.

Motion

Assumed that the robot must realistically move inside the corridor, I have exploited the motion model *odo3.m* to this purpose: by default it changes continuously the direction of the robot, when recalled by the main loop, in order to realize a circle trajectory.

What I have done was editing the main loop in order to let it call the function *odo3.m* only when the robot is situated near the corners; only in that case his direction of motion should change

For this purpose, I realized a function called *locate_robot.m* to locate the robot position and orientation inside the corridor; this information is returned and exploited by a script I realized called *DetDir.m*. Given the detected pose of the robot, it determines whether the direction of the robot must change or not. Both this codes are shown in Appendix A.

Visibility of landmarks

Since the SLAM technology in exam is a visual SLAM, the algorithm simulated should not realistically be able to see the landmarks situated behind the walls. The observations are taken by the algorithm through a struct

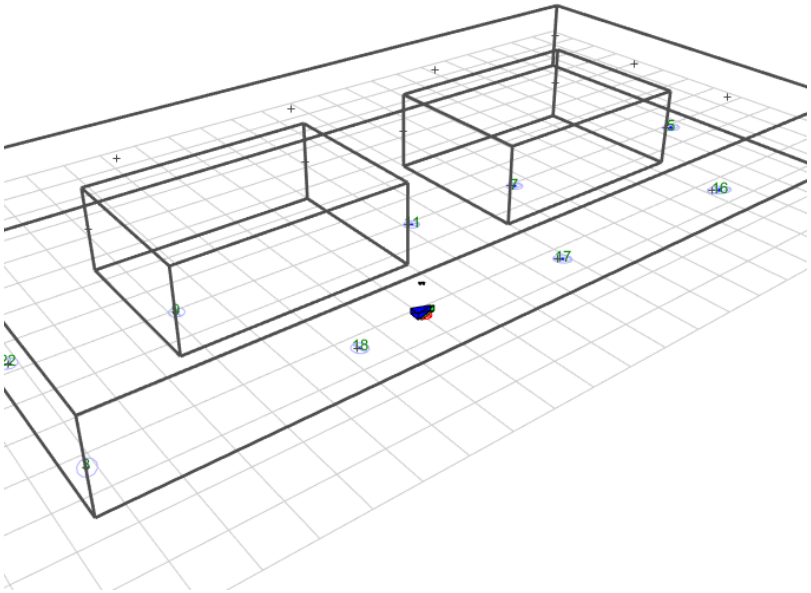

Figure 5.4: The landmarks that the robot observe are the numbered ones, the other are not in LOS, so they are not visible

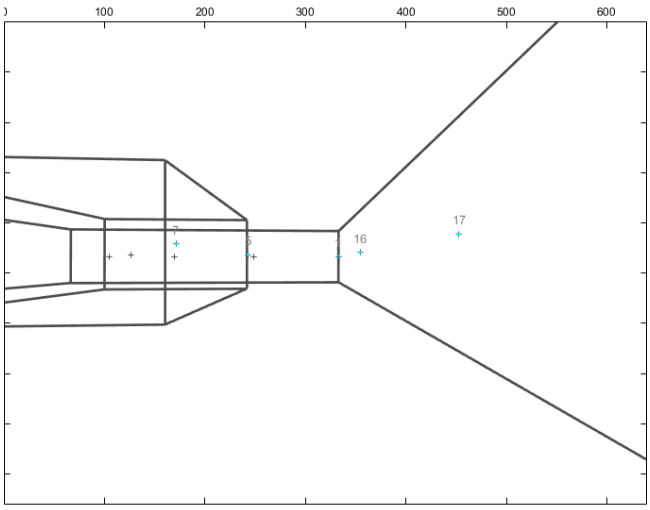

Figure 5.5: The view of the camera: it can be noticed that the not in LOS landmarks are not visible by the camera

called *Obs* which has a field called *vis* that flags if a landmark observation is possible or not.

To exploit this field, I have implemented a script, called *DetVisLmk.m* (it can be viewed in Appendix A), inside the function *CorrectKnownLmk.m*, which is called inside the main loop to correct the landmarks position according to the new observation; the script uses the geometrical information to set as not visible the landmarks not in LOS. In figures 5.4 and 5.5

Position estimation RMSE calculation

To have a precise idea of the behaviour of the algorithm, I have written a script called *calc_RMSE*: it performs the calculation of the RMSE of both the robot pose (5.1) and landmarks (5.2) position once a lap. This code is showed in Appendix A.

$$RMSE(\mathbf{x}_n) = \frac{\sum_{n=1}^{n+N}(\mathbf{x}_n - \hat{\mathbf{x}}_n)^2}{N} \tag{5.1}$$

$$RMSE(\mathbf{m}_n) = \frac{\sum_{i=1}^{L}(\mathbf{m}_n^i - \hat{\mathbf{m}}_n^i)^2}{L} \tag{5.2}$$

where

- N is the number of time steps for each lap
- L is the number of landmarks discovered until the instant n

Configuration file

The main parameters, which I have modified to perform different simulations, have been initialized in a configuration script called *configSLAM.m*. It can be viewed in Appendix A.

Inside this script, I grouped the main parameters, such as:

- *CHOICE_2D_3D*: decides whether the sensor must be placed in the same plane of the landmarks
- *N_LOOPS*: decides the number of loops that the robot must do around the corridor
- *NLOOPFRAME*: number of time steps (also called frames) necessary to perform a single loop
- *LAST_FRAME*: total number of frames

- *NLMKS_ROBOT*: number of landmarks that must be initialized by the SLAM algorithm
- *PIX_ERROR_STD*: standard deviation of pixel recognition of the camera sensor (it sets the accuracy with whom an object is localized in the right pixel by the camera)
- *POS_STD*: standard deviation of the robot motion error, it is set equal to 10 cm in x and y axes and equal to 0 in z axis
- *Sim_Corr*: decides if the environment to simulate must be a corridor

In the initial phase, I have also simulated other environments using the default landmarks generating functions of the toolbox, to realize scripts that reproduce an environment according to the following parameters:

- *KIND_LMK*: it sets if the landmarks have to be represented as points or lines
- *KIND_SENSOR*: it sets the kind of sensor (omnidirectional camera and pinhole camera are available)
- *N_ROBOTS* : it sets the number of the robots

5.1.3 The result of simulations

Given the introduced and edited codes, I have proceeded to simulate several times the implemented scenario; each time I changed the pixel standard error parameter of the camera to observe the different behaviour of the estimate as a function of this parameter.

The selected values of the error standard deviation varies between the following values, grouped in a vector:

$pix\_err\_std = (0.1, 1, 5, 10, 20, 30, 50)px$

I have also made the choice of keeping low the value of the standard deviation of the robot position error (10 cm in both x and y axis, 0 cm in z axis) and to not consider the z axis error in RMSE calculation; the reason is that I wanted to simulate a 2D scenario, similar to the one of LIDAR SLAM implementation, which will be explained in the next section.

The number of landmarks initialized by the algorithm has been kept as low as possible in order to make the algorithm faster.

The results of the simulations are summarized in the following tables (5.25.1) and figures (5.6,5.7,5.8,5.9,5.10,5.11,5.12)

Table 5.1: Position robot RMSE in metres referred to different number of loops and pixel standard deviation

N LOOPS\PIX STD ERR	0.1	1	5	10	20	30	50
1	0.102	0.065	0.098	0.154	0.460	0.533	0.341
2	0.118	0.079	0.090	0.136	0.395	0.778	0.598
3	0.113	0.087	0.088	0.124	0.434	0.837	0.671
4	0.109	0.089	0.088	0.121	0.467	0.843	0.709
5	0.108	0.092	0.095	0.129	0.485	0.835	0.733
6	0.106	0.094	0.103	0.137	0.491	0.801	0.725
7	0.106	0.097	0.110	0.146	0.493	0.778	0.717
8	0.104	0.097	0.113	0.147	0.502	0.763	0.725
9	0.103	0.097	0.116	0.148	0.510	0.749	0.731
10	0.104	0.097	0.119	0.151	0.515	0.737	0.738
11	0.104	0.097	0.120	0.152	0.517	0.719	0.738
12	0.104	0.097	0.122	0.155	0.515	0.702	0.735
13	0.104	0.097	0.124	0.155	0.517	0.687	0.735
14	0.104	0.097	0.125	0.154	0.518	0.673	0.737
15	0.105	0.097	0.126	0.155	0.518	0.662	0.739

Table 5.2: Position RMSE in metres referred to different number of loops and pixel standard deviation

N LOOPS\PIX STD ERR	0.1	1	5	10.000	20.000	30.000
1	0.127	36.457	90.959	0.220	1.145	35.097
2	0.125	0.093	85.537	0.162	0.455	72.699
3	0.103	0.082	0.076	0.110	0.581	18.031
4	0.109	0.102	0.109	0.168	0.598	1.412
5	0.101	0.099	0.116	0.188	0.574	1.240
6	0.104	0.099	0.128	0.189	0.587	1.121
7	0.105	0.105	0.145	0.194	0.553	0.997
8	0.100	0.089	0.117	0.154	0.604	0.884
9	0.107	0.090	0.135	0.171	0.595	0.862
10	0.108	0.085	0.125	0.168	0.575	0.752
11	0.109	0.089	0.120	0.157	0.564	0.716
12	0.112	0.090	0.128	0.163	0.542	0.672
13	0.115	0.096	0.153	0.190	0.521	0.612
14	0.114	0.089	0.122	0.142	0.592	0.650
15	0.117	0.087	0.122	0.151	0.573	0.626

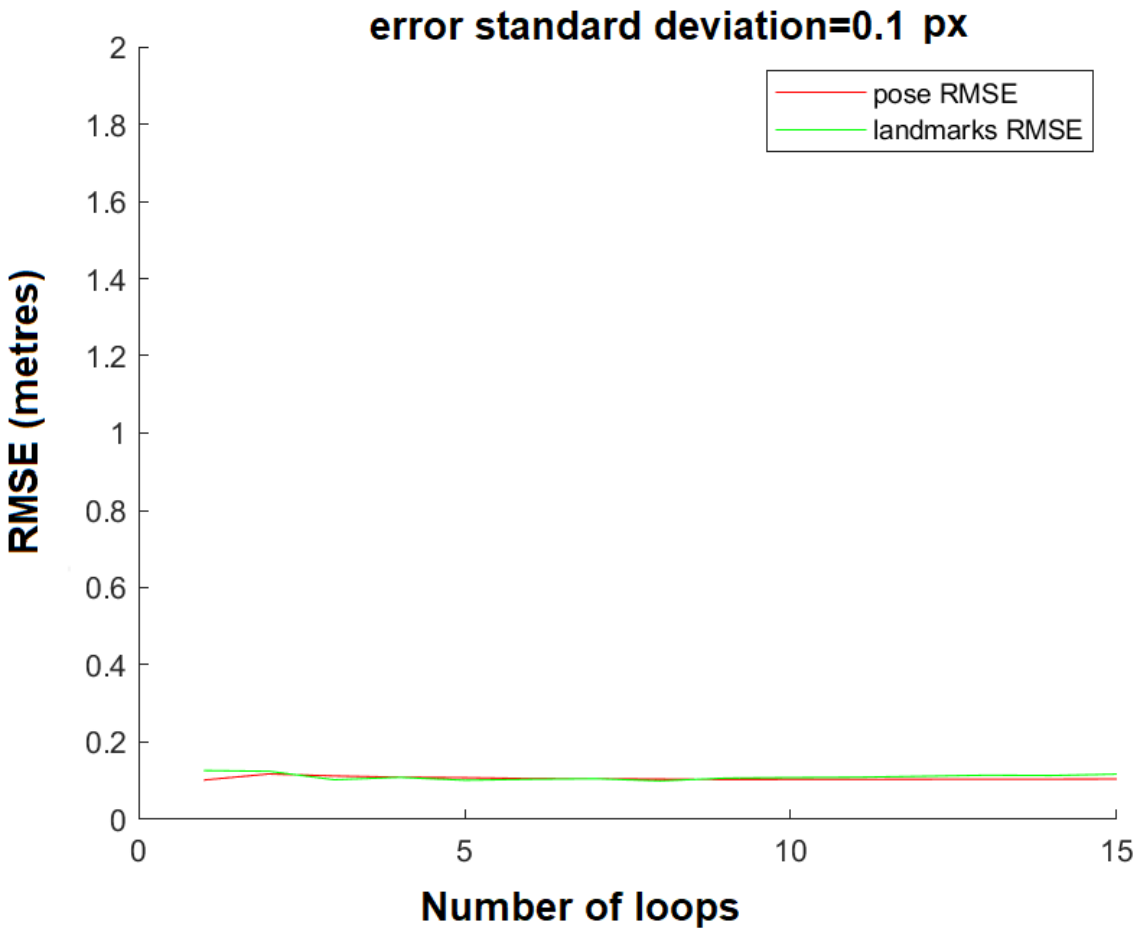

Figure 5.6: Result of simulation considering a error standard deviation value equal to 0.1

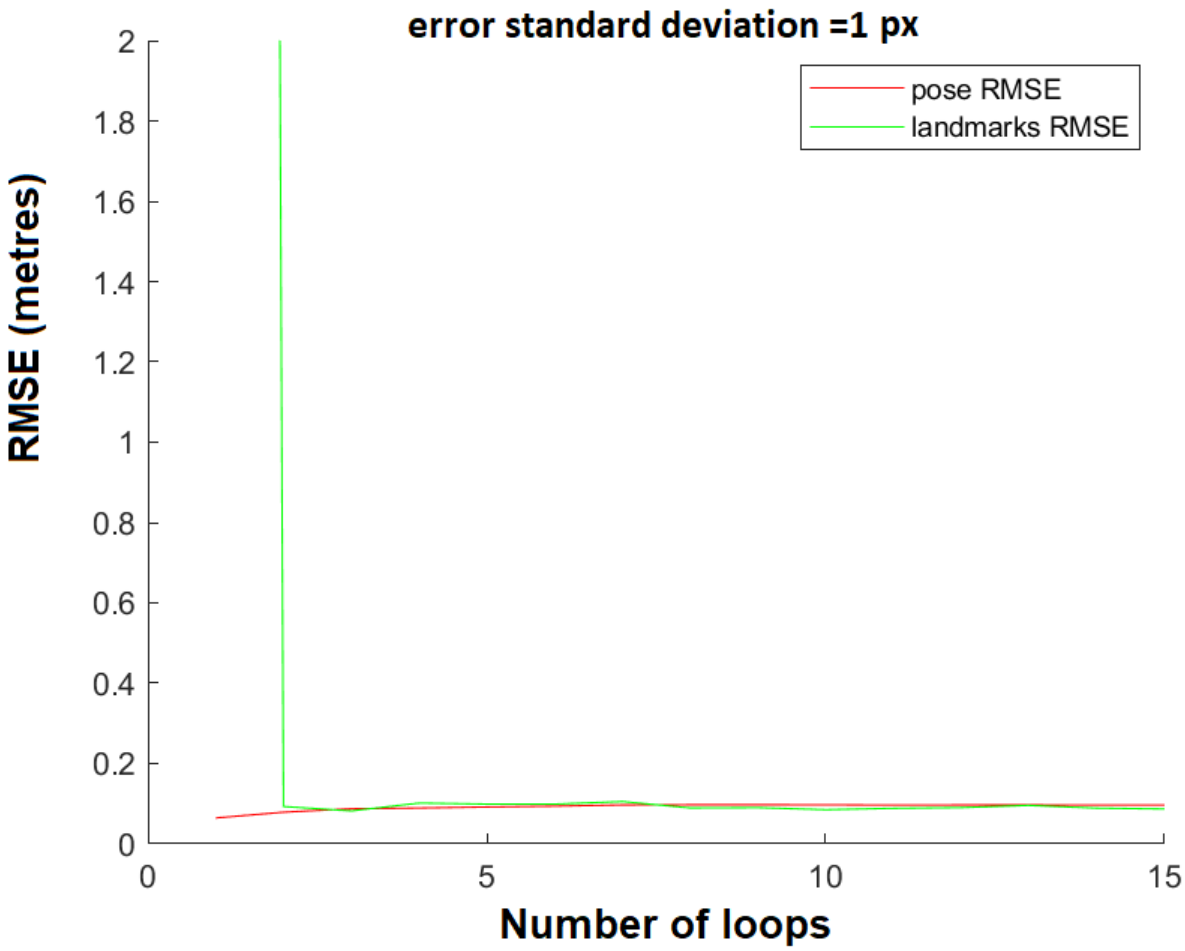

Figure 5.7: Result of simulation considering a error standard deviation value equal to 1

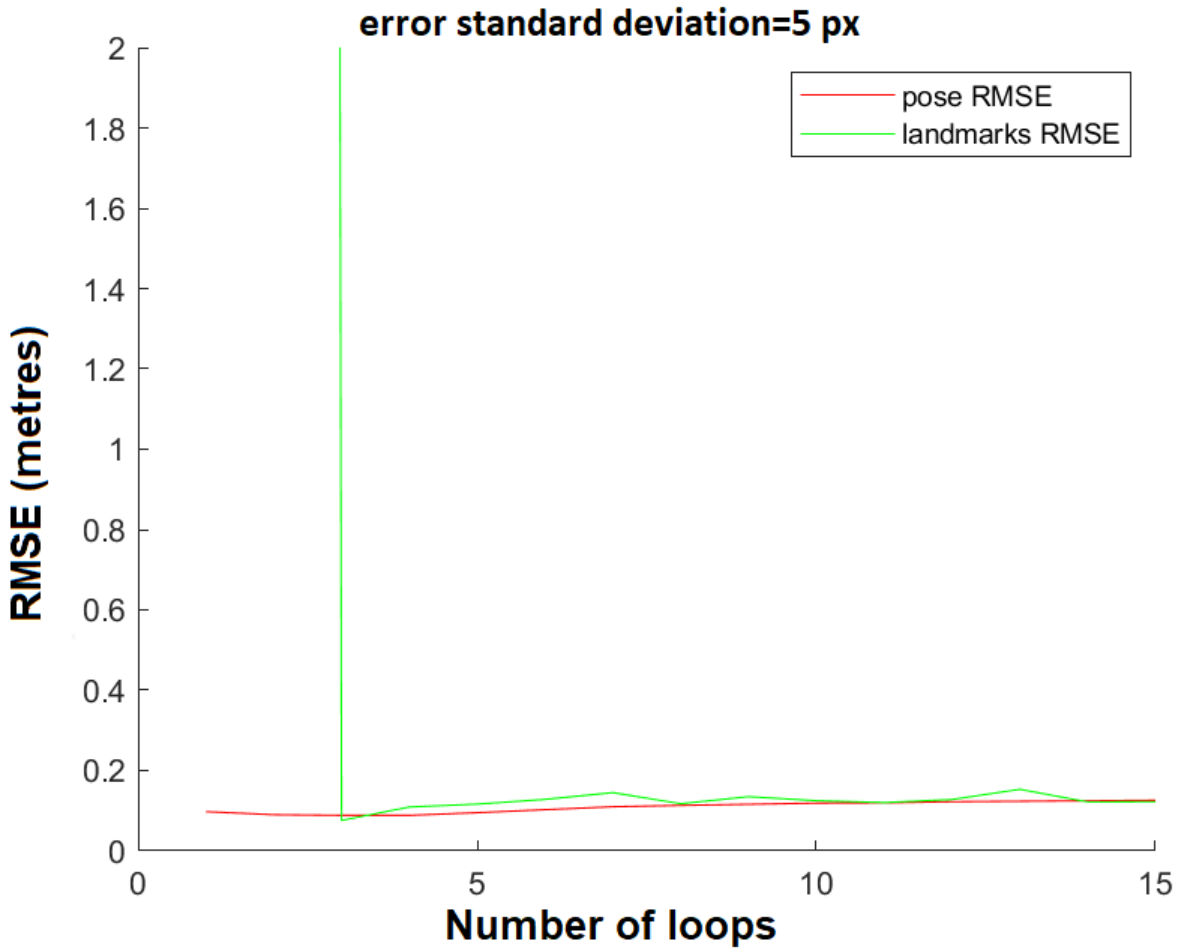

Figure 5.8: Result of simulation considering a error standard deviation value equal to 5

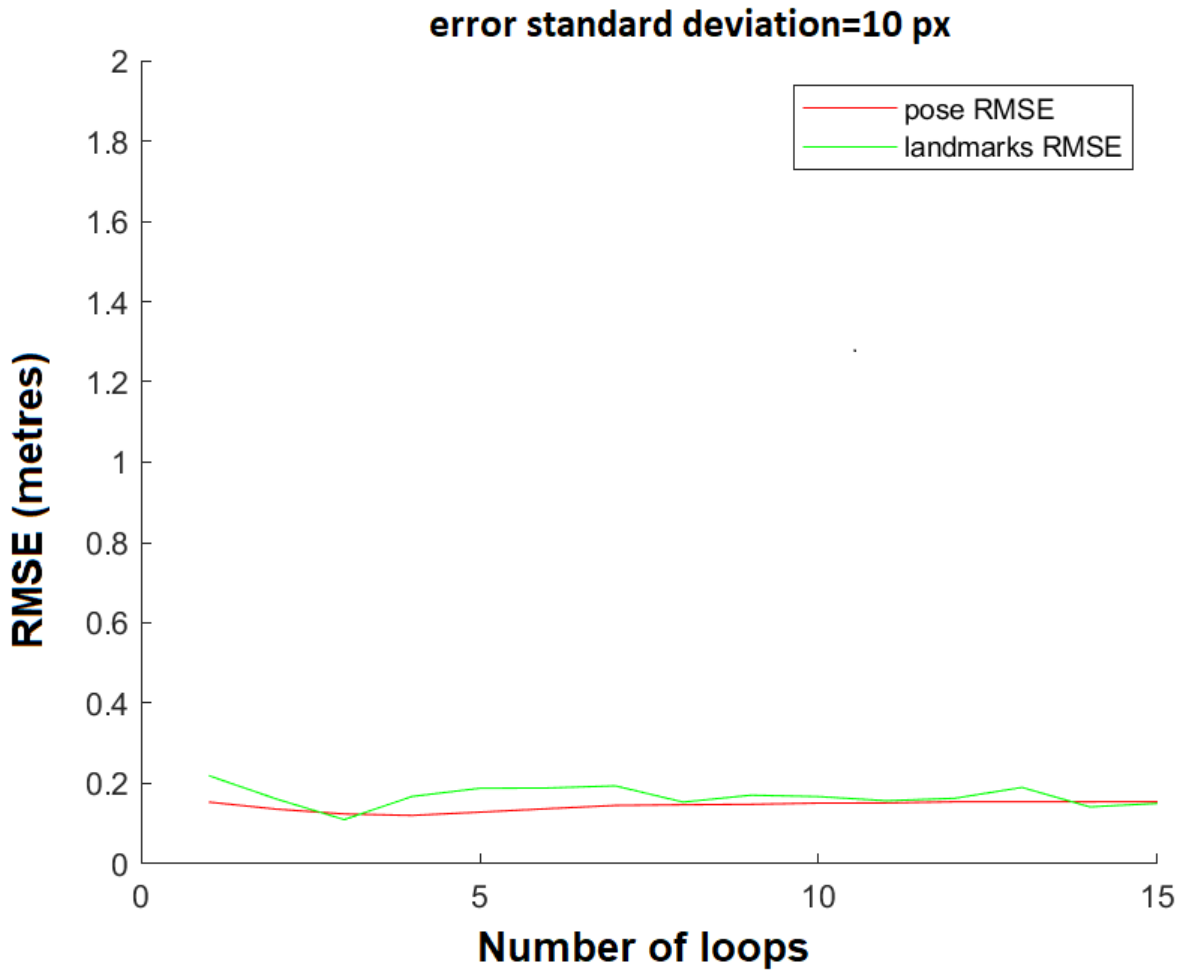

Figure 5.9: Result of simulation considering a error standard deviation value equal to 10

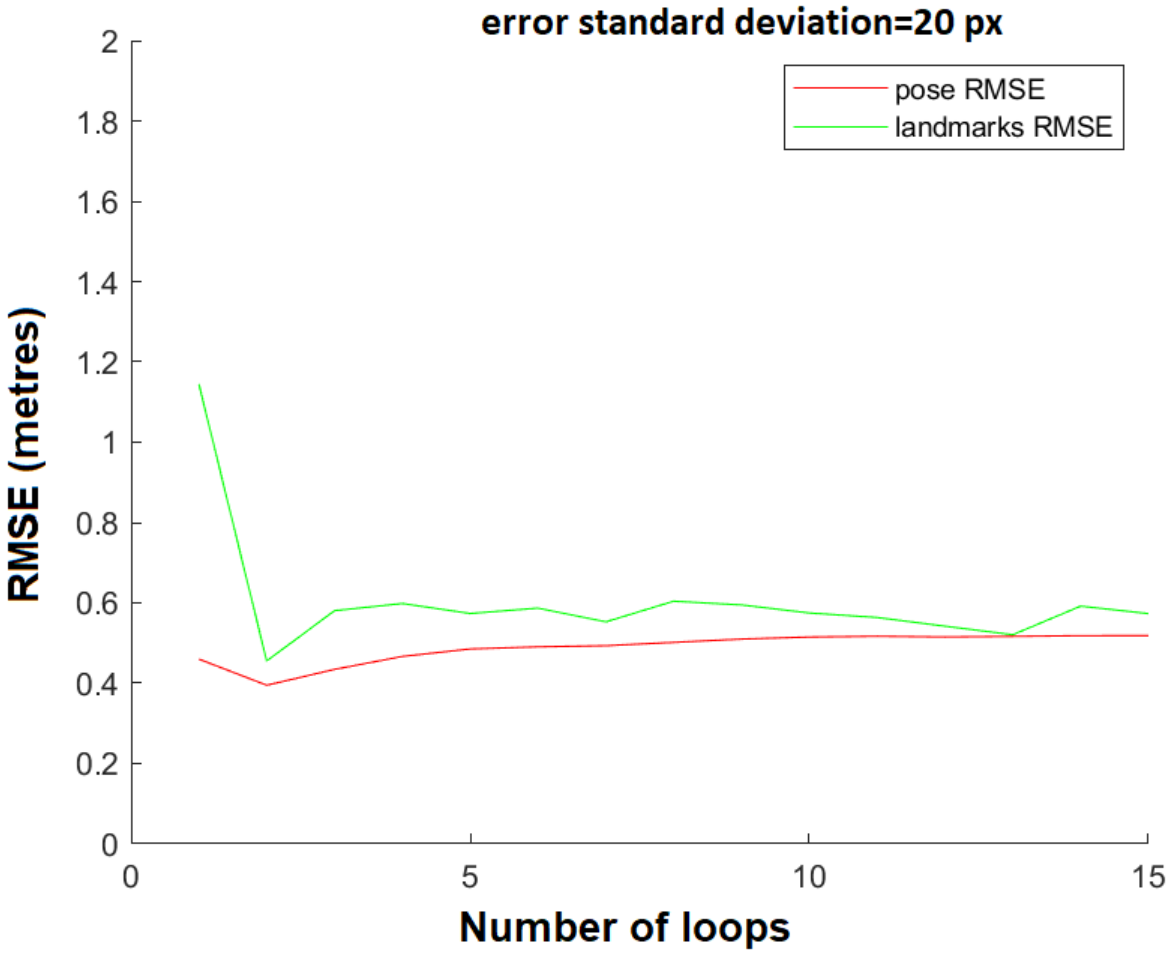

Figure 5.10: Result of simulation considering a error standard deviation value equal to 20

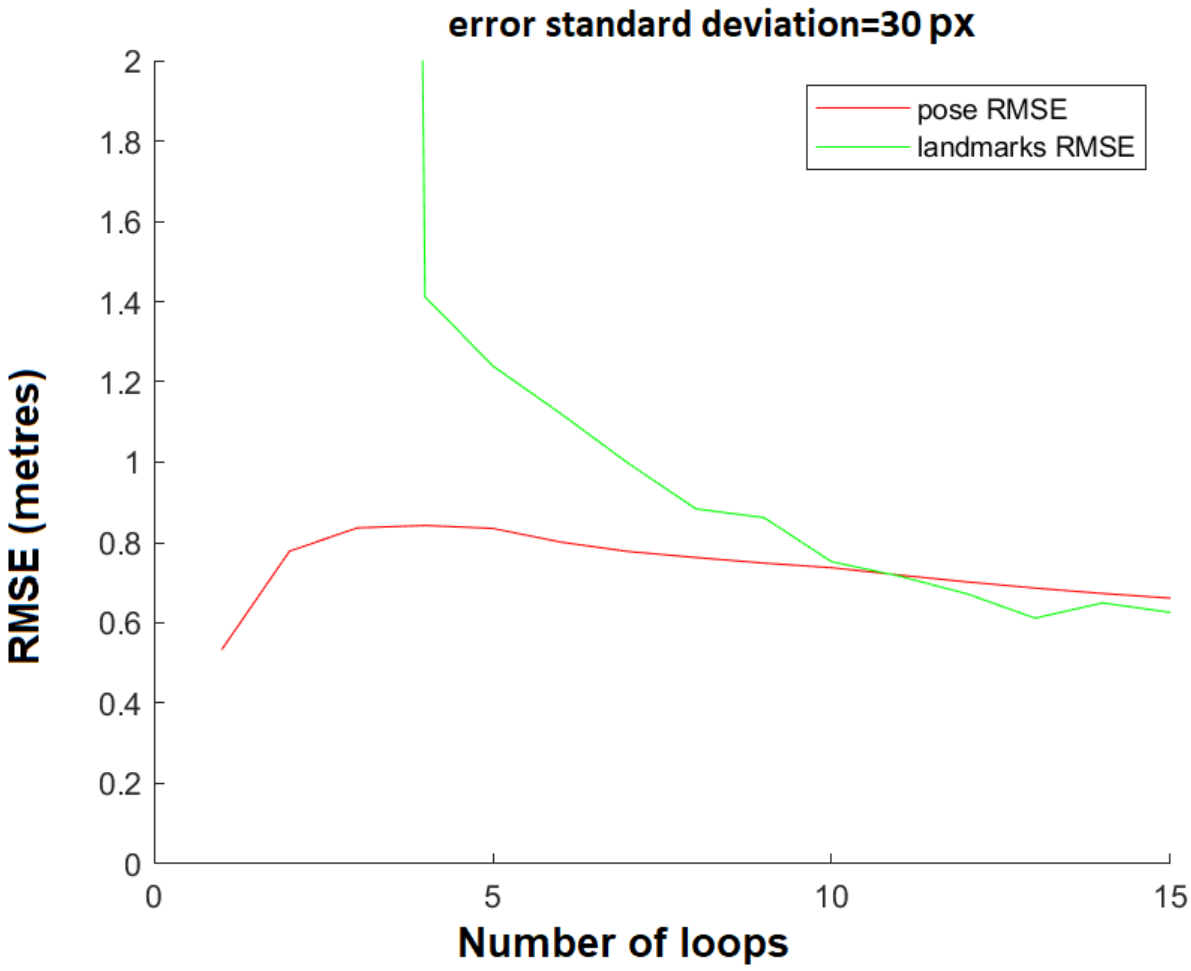

Figure 5.11: Result of simulation considering a error standard deviation value equal to 30

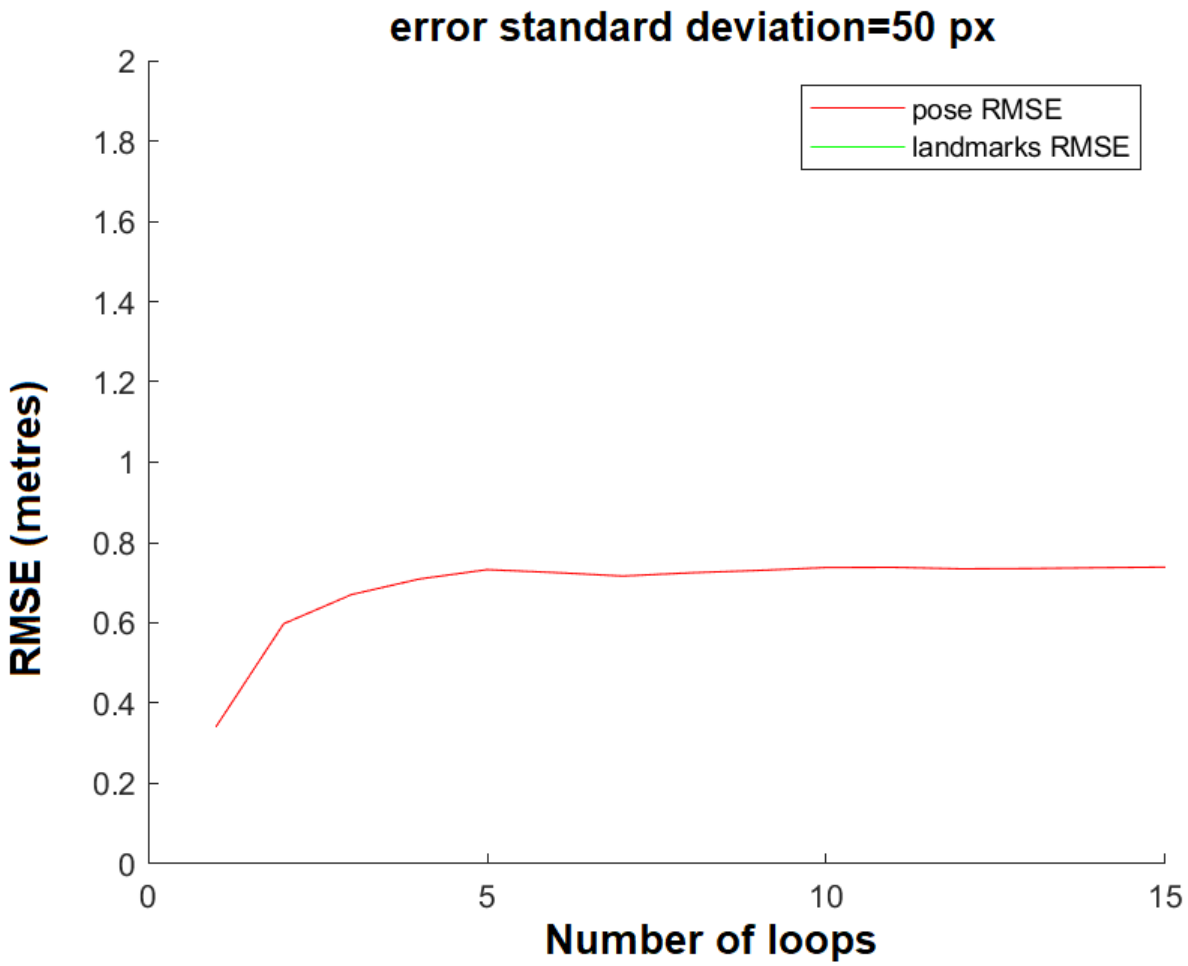

Figure 5.12: Result of simulation considering a error standard deviation value equal to 50

Given these results, some main considerations can be done:

- The position estimation error of both landmarks and robot grows up with the pixel standard error
- If we consider a camera sensor conditioned by a large pixel standard deviation error (up to 50), if the number of laps is sufficiently large, the robot localization performed by the algorithm still gives good results.
- With the increase of the number of laps travelled by the robot, the estimation error of the landmarks tends to diminish; this means that

the concept of crowd-sensing is useful in the localization of landmarks, whose positions are initially completely unknown. A remarkable example is shown in 5.2 in the column corresponding to a pixel standard deviation equal to 30: despite an initially bad landmarks position estimation, after few laps the system recovers.

- Since the pose of the robot is initially well known, crowd-sensing does not make a significant difference in his localization performance. This could be ascribed to the "loop closure" phenomenon of SLAM.

5.2 LIDAR SLAM toolbox

The LIDAR SLAM toolbox is an open source package that attempts to simulate the EKF-SLAM algorithm in a given environment using a LASER SLAM technology implementation.

This package is implemented in a quite different way than the previous one. Here the main script, called *execute_ekf*, loads a set of predefined landmarks and waypoints. The latter are the points through which the robot is supposed to move.

After the data load, the core function of the package is called; the function in exam is named *ekfslam_sim* and it performs the entire simulation of the algorithm using the input data that not only include landmarks and waypoints, but also parameters of the corridor that has to be simulated.

5.2.1 The main function *ekfslam_sim*

The first thing that the function does is setting the configuration parameters of the problem through the script *configfile*; this script can be viewed in Appendix B. Then the environment is plotted with a 2D representation.

After this, the main loop starts running and executing the following operations at every cycle:

- the motion control is computed, together with the associated noise
- the prediction and update steps of the EKF are computed
- the plots are updated.

5.2.2 Realization of the SLAM estimation in the considered scenario

Since this code package is less complicate and advanced than the previous one, here I limited my contribution in adapting the data saved after the execution of the vision SLAM algorithm, in order to let the robot follow the same path; specifically, during the execution of visual SLAM code, I recorded with a fixed time step the positions of the robot during the path and used them to define the waypoints given as input to the LIDAR SLAM code.

The meaning given to landmarks is different from that given in Visual SLAM: here they are used to emulate wireless access points (AP) supposed deployed in the corridor. In this way, by properly setting the parameters observation model (i.e., noise level), it is possible to simulate a Wi-Fi SLAM technology using a software initially designed to simulate a LIDAR-based SLAM system.

A similar reasoning can be made by setting very low error standard deviation parameters for angle of arrival and distance estimations, in order to simulate a mmW/UWB-based SLAM.

The results of the next subsection are obtained by making two hypotheses:

- The interaction between APs and robot brings a distance measurement through the analysis of the received signal strength (RSS)

- Through the installation of an array of antennas is possible to deduce a signal angle of arrival measurement.

The scenario is the following (figure 5.13):

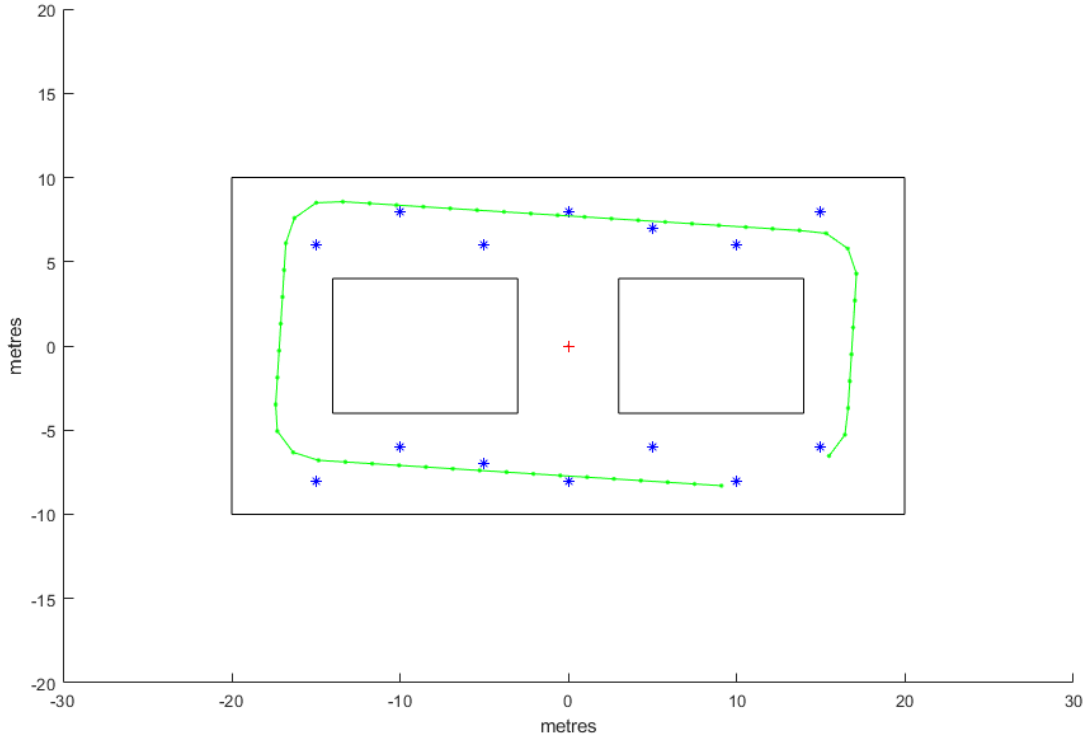

Figure 5.13: Simulated scenario: the landmarks are indicated in blue, while the waypoints are the green dots

The RMSE related to robot pose and landmarks position estimate is calculated through a script called *calc_RMSE_lidar*.

The parameters are set in the script called *configfile*; it is used to set the following parameters among the others:

- *Position*: it is a two-dimensional vector who defines the initial position of the robot through x and y coordinates expressed in m
- *Orientation*: it defines the initial orientation of the robot in *rad*
- *sigmaV*: it defines the standard deviation of the robot speed referred in m/s
- *sigmaG*: it defines the standard deviation in angle of motion referred in *rad*
- *MAX_RANGE*: it defines the maximum detection range of landmarks expressed in metres

- *sigmaR*: it defines the standard deviation of the distances observed from robot to landmarks referred in metres

- *sigmaB*: it defines the standard deviation of the angle observed between robot direction and landmark referred in radiants

5.2.3 Result of simulations

The first group of simulations has been done setting a fixed parameter for the angle error standard deviation (equal to 5°) and varying the distance error standard deviation through the following values referred in metres: $(0.01, 0.1, 0.5, 1, 1.5, 2)$

The results are showed in the following tables (5.3,5.4) and figures (5.14, 5.15, 5.16, 5.17, 5.18, 5.19)

Table 5.3: Robot position RMSE in metres referred to different number of loops and distance standard error

N_LOOPS\Std_err(m)	0.01	0.1	0.5	1	1.5	2
1	0.181	0.111	0.570	0.234	1.141	1.121
2	0.131	0.149	0.416	0.179	0.879	0.898
3	0.097	0.129	0.296	0.147	0.645	0.723
4	0.099	0.143	0.299	0.154	0.663	0.784
5	0.108	0.148	0.304	0.160	0.683	0.841
6	0.079	0.109	0.220	0.126	0.498	0.630
7	0.082	0.112	0.223	0.131	0.511	0.666
8	0.084	0.115	0.226	0.137	0.526	0.697
9	0.085	0.119	0.230	0.143	0.536	0.727
10	0.086	0.121	0.234	0.151	0.546	0.757
11	0.062	0.089	0.169	0.111	0.394	0.555
12	0.064	0.090	0.172	0.116	0.401	0.575
13	0.066	0.092	0.175	0.121	0.407	0.591
14	0.068	0.093	0.177	0.125	0.412	0.609
15	0.069	0.094	0.180	0.127	0.419	0.626

Table 5.4: Landmarks position RMSE in metres referred to different number of loops and distance standard error

N_LOOPS\Std_err(m)	0.01	0.1	0.5	1	1.5	2
1	0.036	0.253	0.233	0.214	0.834	0.653
2	0.024	0.172	0.096	0.137	0.486	0.624
3	0.032	0.134	0.088	0.124	0.389	0.721
4	0.033	0.093	0.126	0.098	0.319	0.641
5	0.049	0.081	0.127	0.119	0.344	0.651
6	0.043	0.079	0.139	0.133	0.364	0.649
7	0.035	0.065	0.111	0.135	0.335	0.633
8	0.037	0.066	0.106	0.124	0.343	0.613
9	0.036	0.061	0.128	0.126	0.326	0.608
10	0.036	0.062	0.121	0.137	0.312	0.634
11	0.038	0.069	0.128	0.135	0.319	0.633
12	0.042	0.067	0.137	0.138	0.301	0.625
13	0.039	0.061	0.128	0.115	0.286	0.623
14	0.039	0.063	0.128	0.111	0.297	0.612
15	0.035	0.060	0.131	0.105	0.305	0.597

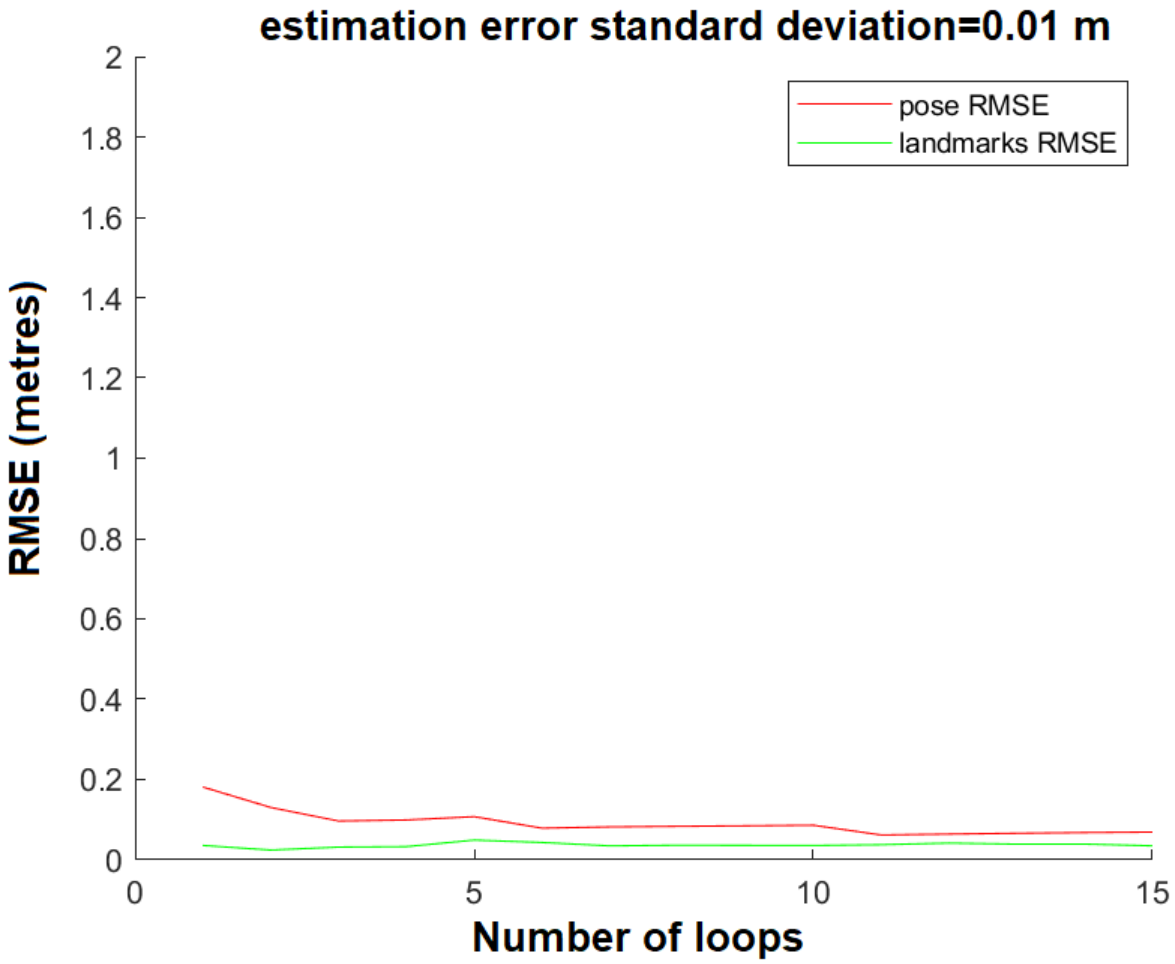

Figure 5.14: Result of simulation considering an estimation error standard deviation value equal to 0.01 m

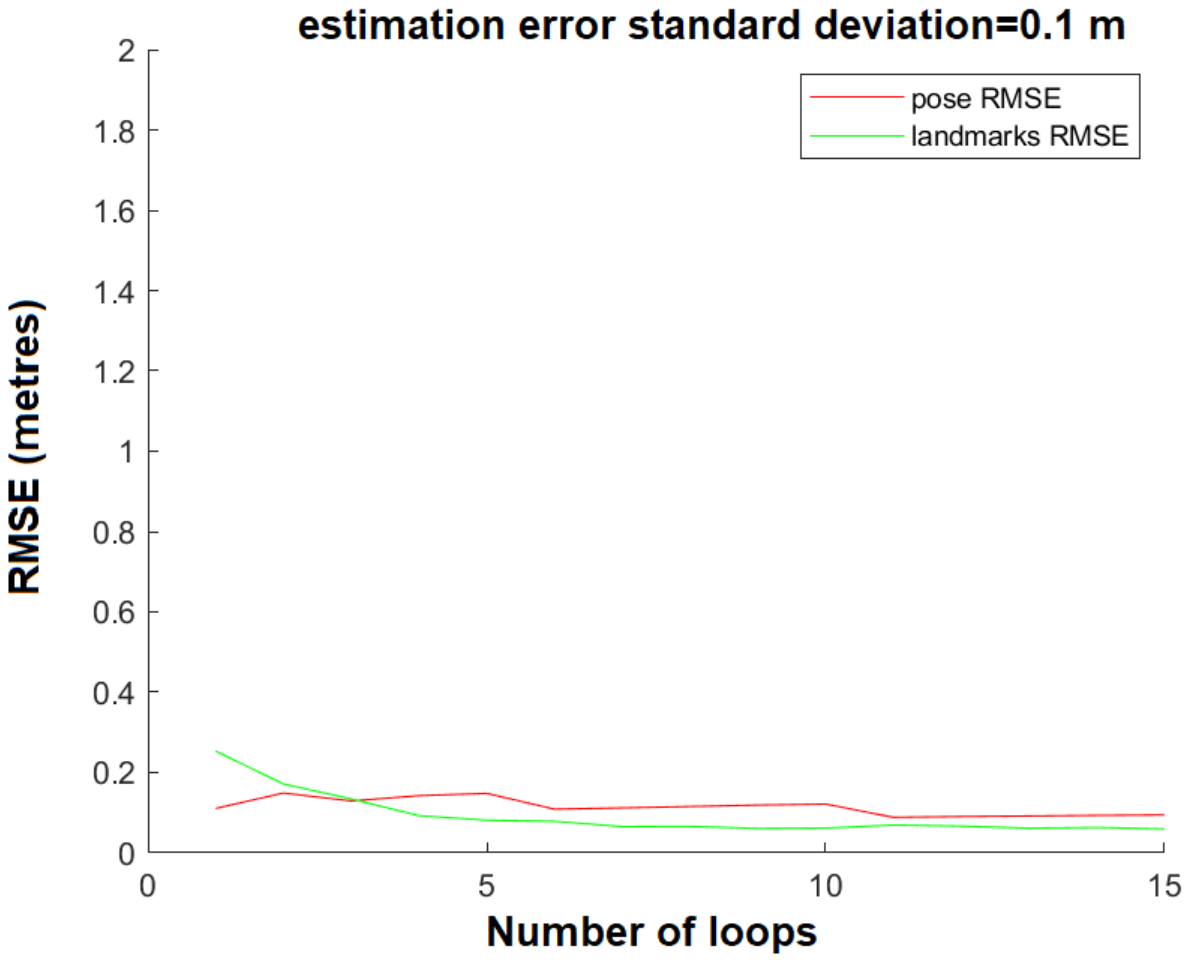

Figure 5.15: Result of simulation considering an estimation error standard deviation value equal to 0.1 m

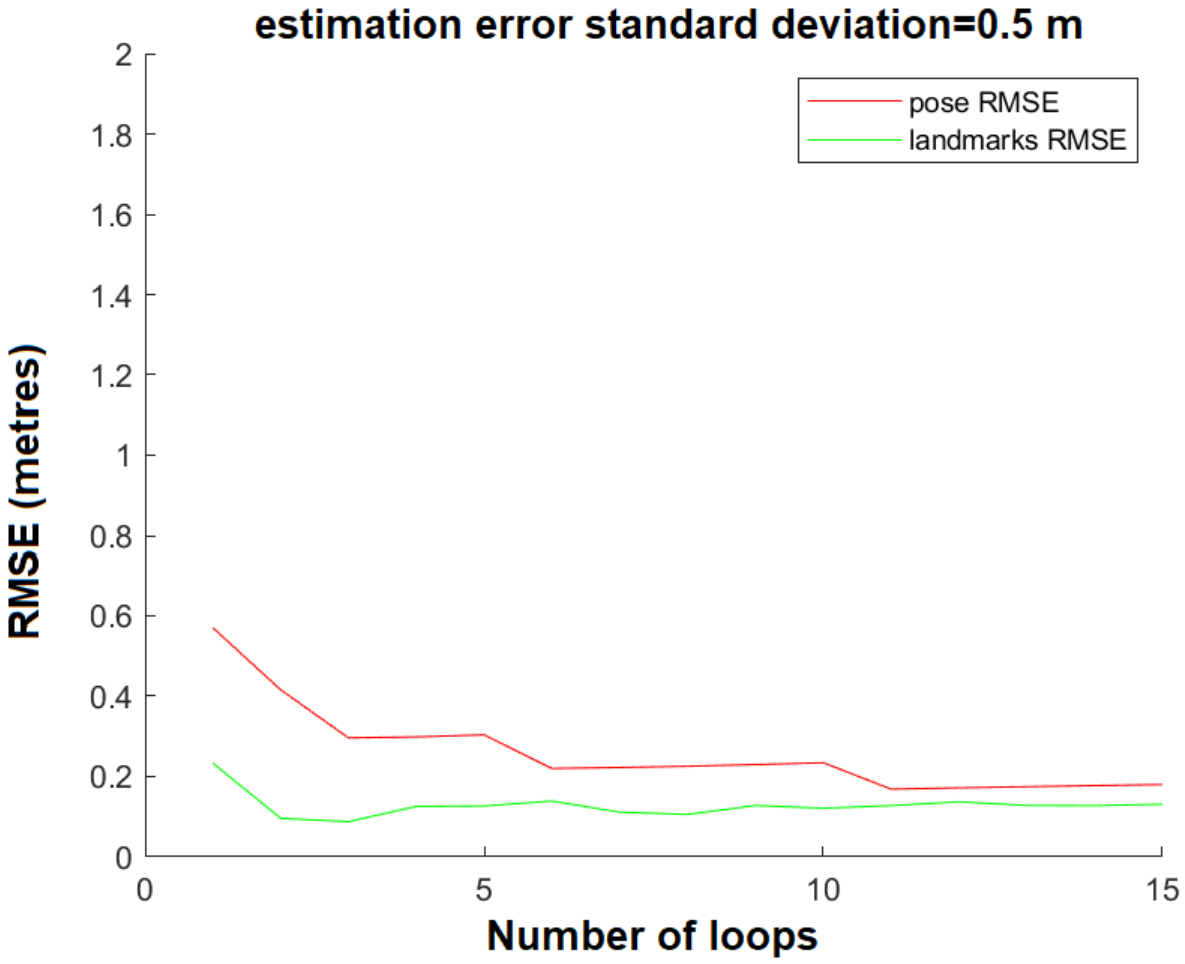

Figure 5.16: Result of simulation considering an estimation error standard deviation value equal to 0.5 m

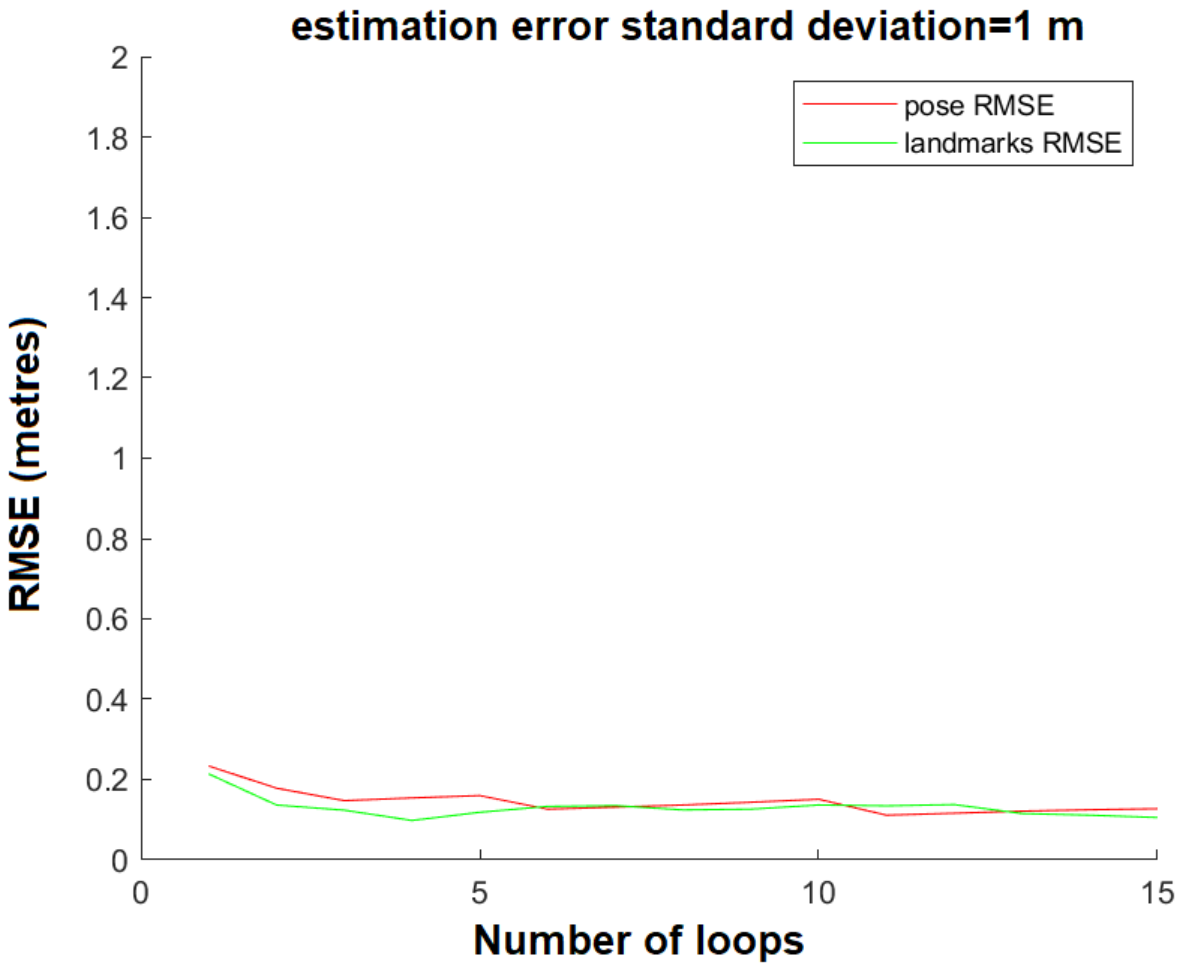

Figure 5.17: Result of simulation considering an estimation error standard deviation value equal to 1 m

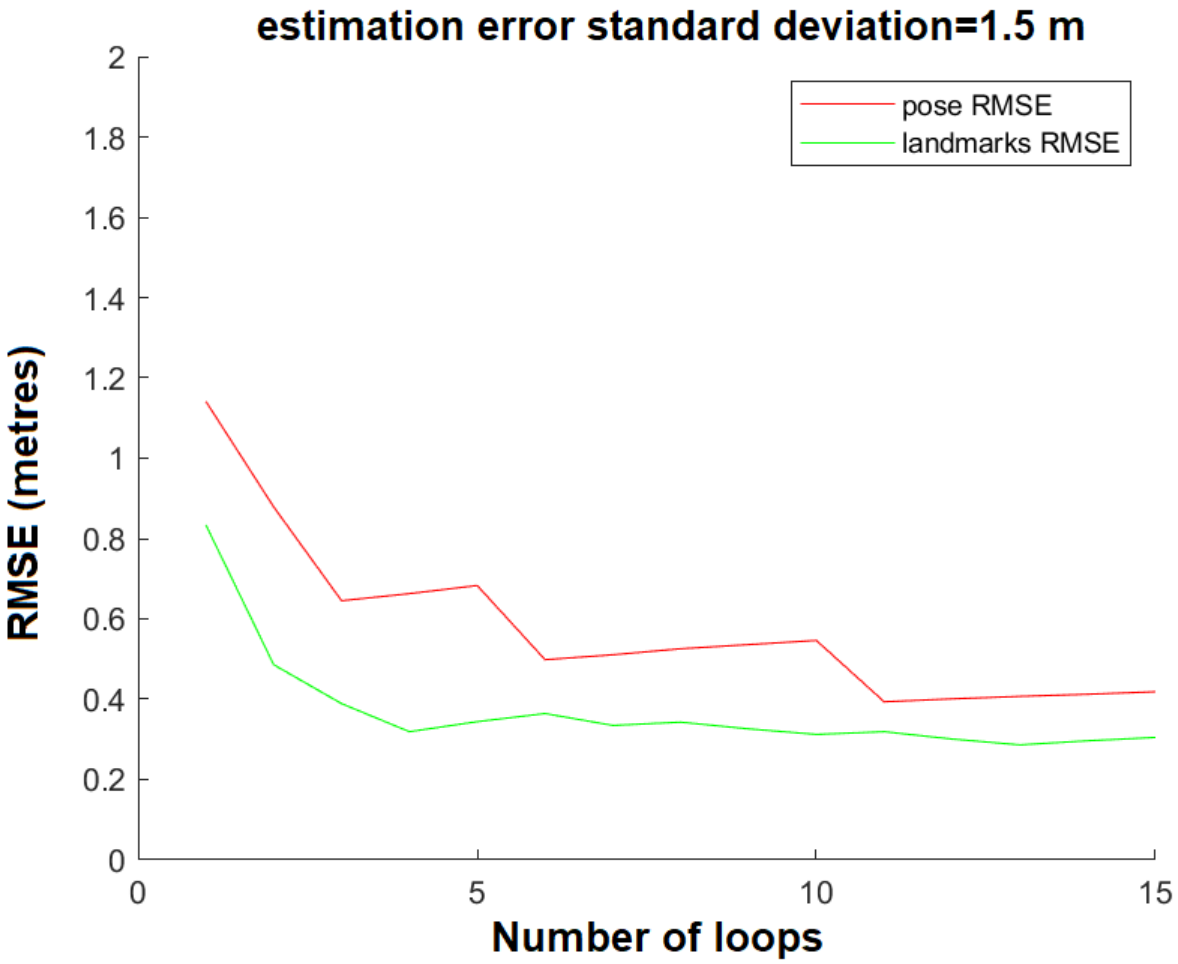

Figure 5.18: Result of simulation considering an estimation error standard deviation value equal to 1.5 m

Figure 5.19: Result of simulation considering an estimation error standard deviation value equal to 2 m

The second group of simulations has been done in a dual way, but this time the distance observation standard error has been kept constant (equal to $1m$) and the angle observation standard error has been varied through the following values: (5°,10°,20°,30°,45°,60°) .

The results are showed in the following tables (5.5,5.6) and figures (5.20,5.21,5.22,

5.23,5.24,5.25)

Table 5.5: Robot position RMSE in metres referred to different number of loops and angle standard deviation in degrees

N_LOOP\ANGLE_STD_ERR	5	10	20	30	45	60
1	0.453	0.146	0.451	0.346	0.224	0.358
2	0.432	0.122	0.398	0.277	0.224	1.031
3	0.340	0.100	0.355	0.213	0.202	1.119
4	0.370	0.112	0.418	0.219	0.224	1.443
5	0.403	0.124	0.482	0.238	0.271	1.672
6	0.298	0.091	0.366	0.182	0.226	1.343
7	0.312	0.094	0.398	0.190	0.258	1.474
8	0.325	0.100	0.432	0.197	0.289	1.609
9	0.337	0.105	0.471	0.203	0.314	1.742
10	0.349	0.110	0.495	0.219	0.342	1.850
11	0.254	0.081	0.367	0.158	0.263	1.383
12	0.262	0.084	0.384	0.164	0.280	1.452
13	0.268	0.089	0.403	0.172	0.291	1.517
14	0.273	0.094	0.416	0.174	0.309	1.574
15	0.280	0.098	0.431	0.176	0.326	1.632

Table 5.6: Landmarks position RMSE in metres referred to different number of loops and angle standard deviation in degrees

N_LOOP\ANGLE_STD_ERR	5	10	20	30	45	60
1	0.362	0.170	0.441	0.338	0.663	2.148
2	0.377	0.102	0.426	0.215	0.569	1.950
3	0.301	0.105	0.516	0.166	0.465	1.937
4	0.327	0.080	0.479	0.176	0.450	1.973
5	0.299	0.070	0.461	0.178	0.433	1.922
6	0.278	0.066	0.504	0.169	0.449	1.930
7	0.279	0.073	0.499	0.163	0.457	1.944
8	0.284	0.074	0.471	0.162	0.444	1.916
9	0.284	0.078	0.466	0.162	0.438	1.910
10	0.271	0.077	0.471	0.155	0.436	1.910
11	0.267	0.080	0.482	0.148	0.441	1.912
12	0.266	0.075	0.478	0.146	0.445	1.886
13	0.245	0.091	0.485	0.142	0.455	1.881
14	0.259	0.085	0.471	0.131	0.462	1.885
15	0.251	0.074	0.465	0.135	0.465	1.892

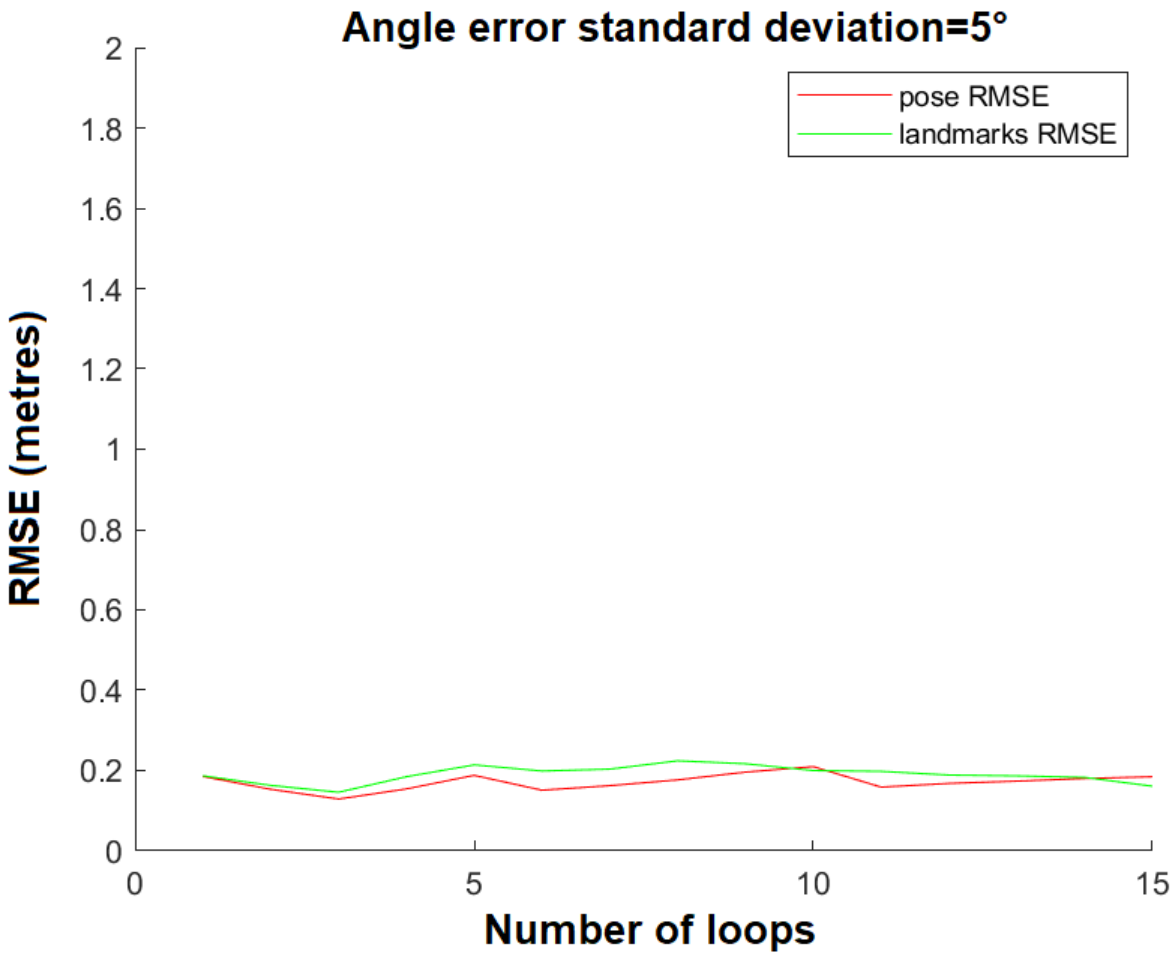

Figure 5.20: Result of simulation considering an angle error standard deviation value equal to 5°

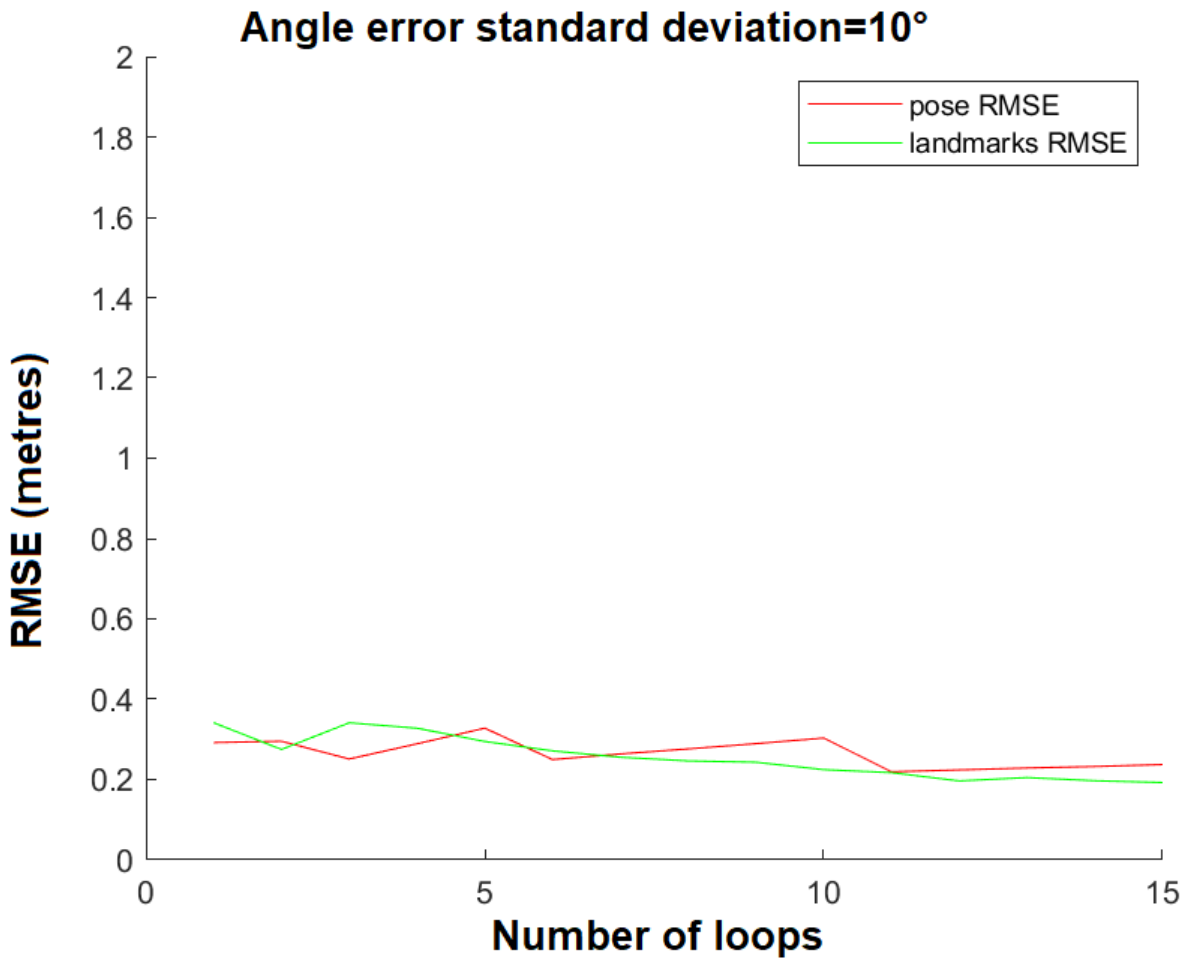

Figure 5.21: Result of simulation considering an angle error standard deviation value equal to 10°

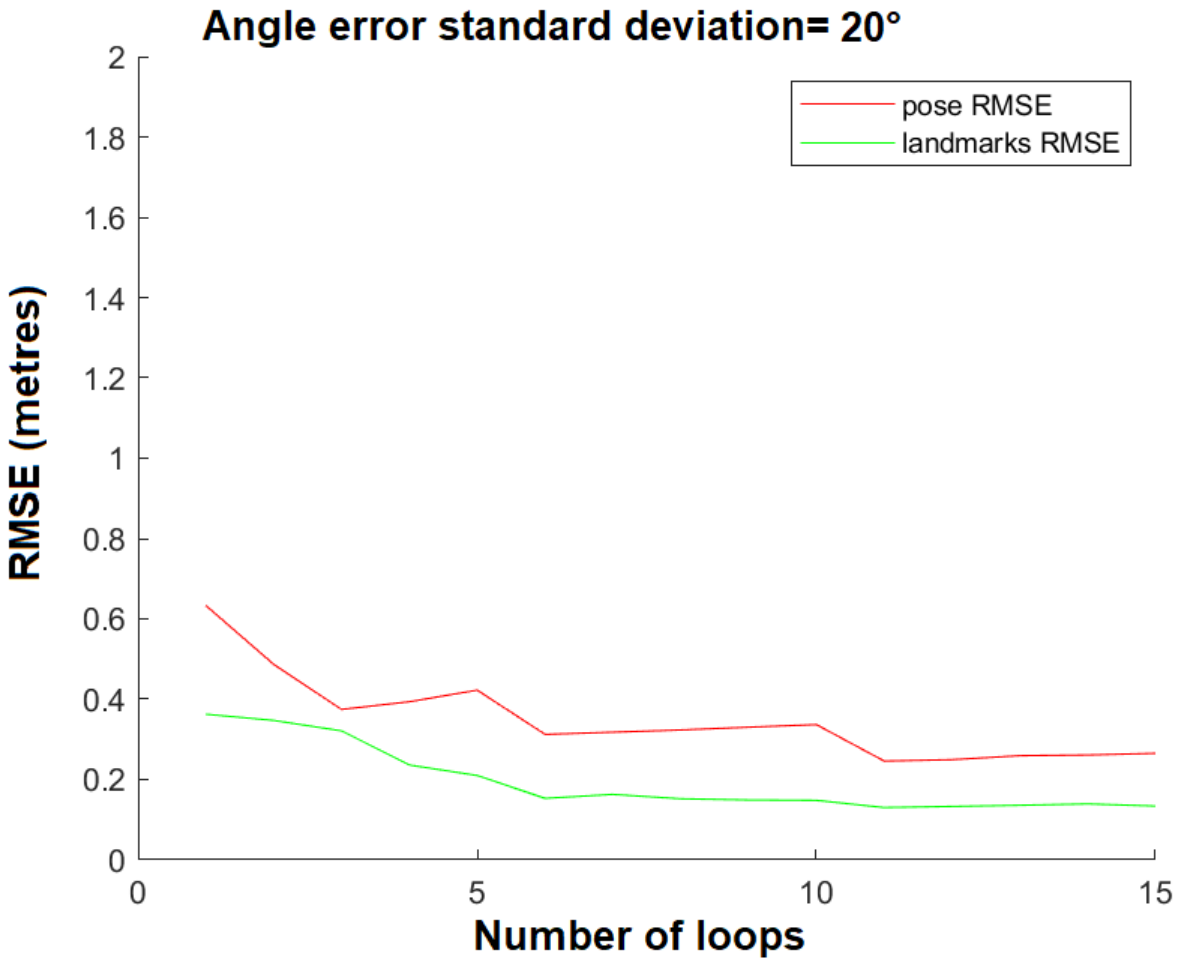

Figure 5.22: Result of simulation considering an angle error standard deviation value equal to 20°

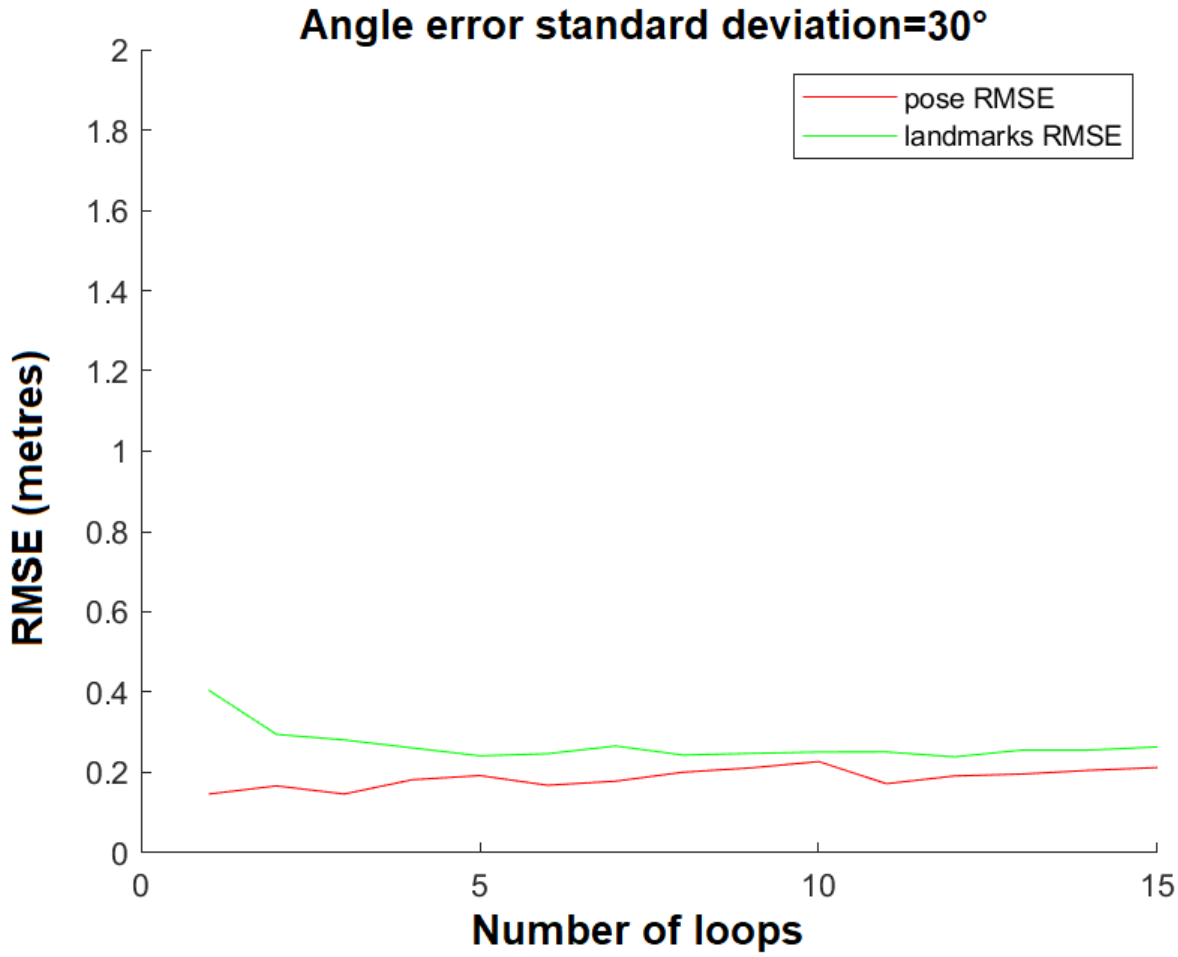

Figure 5.23: Result of simulation considering an angle error standard deviation value equal to 30°

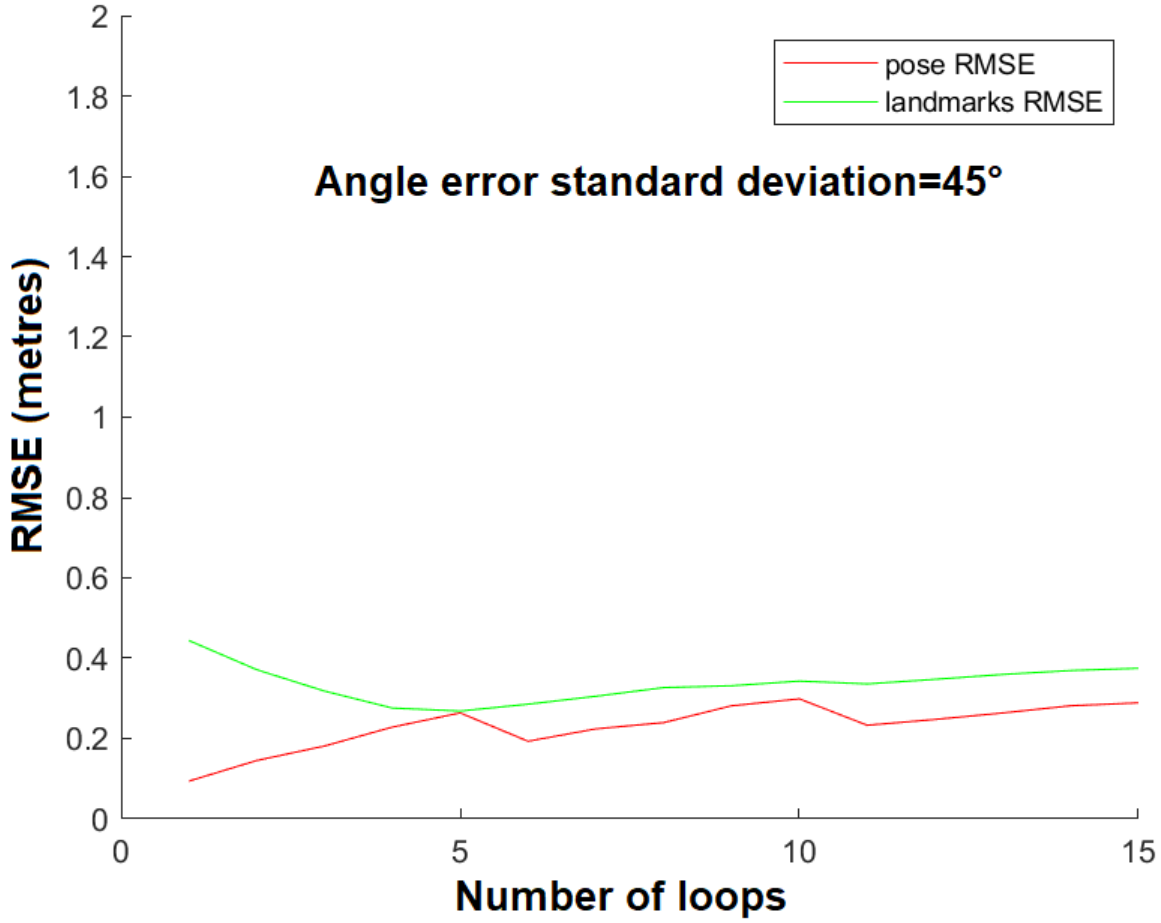

Figure 5.24: Result of simulation considering an angle error standard deviation value equal to 45°

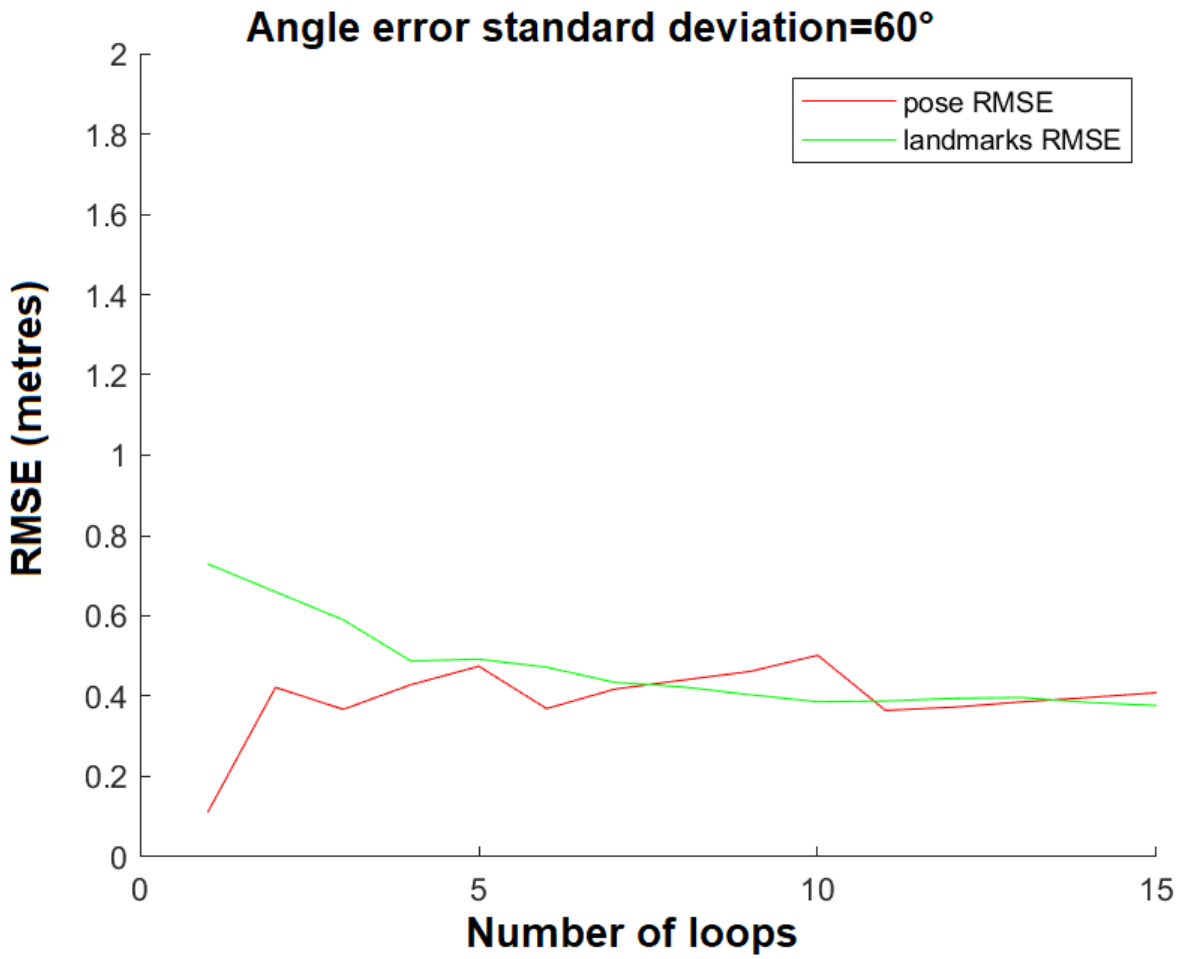

Figure 5.25: Result of simulation considering an angle error standard deviation value equal to 60°

Given the results, here are some considerations:

- As the angle error standard deviation increases, so does the pose and landmarks RMSE
- The localization of the landmarks improves as more loops are made by the robot
- The pose of the robot begins to suffer of accumulated error as the number of loops increases if the angle error is high

- Even with relatively high angle error standard deviation (up to 60°) the localization of the landmarks and the robot still gives good results (RMSE less to 1 meter).

- A realistic Wi-Fi SLAM system can be simulated by setting a high angular and distance uncertainty in the observation, considering that distance estimation based on received power (RSS) is typically very poor; an example is the one showed in figure 5.24, where the angle standard error is equal to 45° and the distance standard error is equal to 1 meter. The mapping of the landmarks after some laps converge to a good accuracy (0.5 m), while the robot pose accuracy is kept constant in time (about 0.5 m).

- Observing the figures 5.14-5.19, it can be stated that if a radio technology capable to provide distance estimations with error standard deviation up to 0.5 m is available, such as UWB RADAR SLAM, then a very good localization and mapping accuracy can be obtained (error standard deviation up to 20 cm). However, even ranging standard deviation error up to 2 metres, lead to a position estimation error of about 60 cm.

Chapter 6

Conclusions

In this thesis a general overview of the SLAM problem, his methods of resolution and the actual state of the art have been presented. Then, simulations on different technologies have been performed in order to obtain a comparison and deduce some highlights.

The EKF is a robust method for SLAM implementation, which can be applied when linear approximations and Gaussian noise approximation in both motion and measurement model can be made. Otherwise, when the aforesaid hypothesis is not applicable or the system experience very strong non linearities, the RBPF is another consolidate method.

The simulations performed through different technologies, allow to draw some conclusions:

- In the different simulations, the initial position of robots is supposed known, thus making the estimation tracking errors during the different laps almost constant, when the error standard deviation of the sensor observations is restrained. Conversely, the localization of landmarks, whose initial position is not known, tends to become more accurate as the number of laps increases. The crowd-sensing concept justifies this behaviour.

- A visual-based SLAM implementation, using an accurate camera sensor (error standard deviation up to 5 px), can provide localization of robot and mapping of the landmarks with an estimation error of about 10 cm. A LIDAR-based SLAM implementation can reach the same accuracy.

- A RADAR-based SLAM implementation can overcome the difficulties caused by NLOS conditions, typical of vision/LASER-based SLAM, and does not need mechanical moving parts; besides, it allows the implementation of a SLAM system in a more compact way and with less

power consumption.

- A Wi-Fi SLAM system has been simulated considering a high distance uncertainty, associated to the data provided by RSS, and a high angular uncertainty, assuming this information coming from an antenna array installed on a common smartphone. The achieved result shows a good accuracy in the device localization (estimation error up to 0.6 m), justifying the idea that Wi-Fi SLAM could be used in the next future to implement low-cost SLAM.

- A mmW/UWB based SLAM has been simulated assuming very low uncertainties in signal angle of arrival and distance estimation error standard deviation parameters; the results have confirmed that these technologies can achieve excellent performance; the figure 5.14, which can be associated to a mmW technology, shows that an accuracy of about 10 cm can be reached.

A possible next step, with respect to the technologies simulated in this thesis activity, is to edit the code in order to make completely unknown the initial robot position, in order to assess the robustness of SLAM in the absence of strong hypothesis on a-priori knowledge.

In this way, it could be possible to understand whether crowd-sensing can be beneficial not only to obtain an accurate localization of the mobile user/robot, but also an accurate mapping of the environment, even in presence of scarcely accurate sensors.

Another important step could be the experimental implementation of SLAM using real devices.

Appendix A

MATLAB code for visual SLAM

script configSLAM.m

```
%%USER CONFIGURATION PARAMETERS

CHOICE_2D_3D=0;
if CHOICE_2D_3D==0
    HEIGHT_SENSOR=1.5;       %height of the sensor with respect
       to robot
else
    HEIGHT_SENSOR=0.5;
end
N_LOOPS=10;              %If userDataCorridor_2 is used, the
   number of loops can be defined
NLOOPFRAME=1200;
LAST_FRAME=NLOOPFRAME*N_LOOPS;        %number of iterations
NLMKS_ROBOT=25;         %number of lmks initialized in the robot
LMK_SIZE=6;              %dimension of lmks
NPTS=5;                  %number of points for each side
CREATE_VIDEO=false;      %flag which activates creation of
   videos of the simulation
PIX_ERROR_STD=0.1;        %std error of the pinhole sensor
N_ROBOT_RND=20;         %If userDataPnt_rnd is used, the n of
   robots is defined
WIDTH=20;               %width of cloister of landmarks
POS_STD=[0.1;0.1];          %position standard error of robot in
    userDataCorridor_2
Sim_Corr=1;
Choose=-1;              %flag for the control of direction
   chosen by robot in the corridor
```

```
N_ROBOTS='1';
KIND_LMK='lin';
KIND_SENSOR='p';
a=[N_ROBOTS KIND_LMK '_' KIND_SENSOR];

if Sim_Corr==0;
    switch a
        case '1pnt_p'
            userDataPnt_1p;
        case '1pnt_o'
            userDataPnt_1o;
        case '2pnt_p'
            userDataPnt_2p;
        case '2pnt_o'
            userDataPnt_2o;
        case '2pnt_m'
            userDataPnt_2m;
        case '3pnt_p'
            userDataPnt_3p;
        case '1lin_p'
            userDataLin_1p;
        case '2lin_p'
            userDataLin_2p;
        case '3lin_p'
            userDataLin_3p;
        case 'rnd_pnt_p'
            userDataPnt_rnd;
    end
else
    userData_corridor
end
```

script slamtb.m

```
% SLAMTB  An EKF-SLAM algorithm with simulator and graphics.
%
%   This script performs multi-robot, multi-sensor, multi-landmark 6DOF
%   EKF-SLAM with simulation and graphics capabilities.
%
%   Please read slamToolbox.pdf in the root directory thoroughly before
%   using this toolbox.
%
%   - Beginners should not modify this file, just edit USERDATA.M and enter
%   and/or modify the data you wish to simulate.
```

```
11 %
12 %   - More advanced users should be able to create new landmark
       models, new
13 %   initialization methods, and possibly extensions to multi-map
       SLAM. Good
14 %   luck!
15 %
16 %   - Expert users may want to add code for real-data
       experiments.
17 %
18 %   See also USERDATA, USERDATAPNT, USERDATALIN.
19 %
20 %   Also consult slamToolbox.pdf in the root directory.
21
22 %   Created and maintained by
23 %   Copyright 2008, 2009, 2010 Joan Sola @ LAAS-CNRS.
24 %   Copyright 2011, 2012, 2013 Joan Sola.
25 %   Programmers (for parts of the toolbox):
26 %   Copyright David Marquez and Jean-Marie Codol @ LAAS-CNRS
27 %   Copyright Teresa Vidal-Calleja @ ACFR.
28 %   See COPYING.TXT for full copyright license.
29
30 %% OK we start here
31
32 % clear workspace and declare globals
33
34 global Map
35
36 %% I. Specify user-defined options - EDIT configSLAM.m
37
38 configSLAM
39 SPS=1/Time.dt;
40 SENSE=pi/4*SPS;
41 clockwise=1;                %Counterclockwise initialization of
       sense
42 vel_ang=SENSE*clockwise;
43
44 %% II. Initialize all data structures from user-defined data in
       userData.m
45 % SLAM data
46 Position=Robot{1}.position(1:2);            %Saving initial
       position of the first robot
47
48 Orientation=Robot{1}.orientationDegrees(3); %Saving initial
       orientation of the first robot
49
50 [Rob,Sen,Raw,Lmk,Obs,Tim]     = createSlamStructures(...
51     Robot,...
52     Sensor,...      % all user data
```

```
53      Time,...
54      Opt);
55
56  % Simulation data
57  [SimRob,SimSen,SimLmk,SimOpt] = createSimStructures(...
58      Robot,...
59      Sensor,...      % all user data
60      World,...
61      SimOpt);
62
63  % Graphics handles
64  [MapFig,SenFig]                = createGraphicsStructures(...
65      Rob, Sen, Lmk, Obs,...       % SLAM data
66      SimRob, SimSen, SimLmk,...   % Simulator data
67      FigOpt);                     % User-defined graphic options
68
69  %%initialiting the vector of waypoints of this algorithm to use
        them in the
70  %%lidar EKF SLAM and initializing the state of the robots that
        are used to
71  %%get the waypoints and to get the MSE related to the positions
        of the robots
72  v=zeros(1,NLMKS_ROBOT);
73  wp=[];
74  jj=1;
75  real_state=cell(1,length(MapFig.Rob));
76  est_state=cell(1,length(MapFig.Rob));
77
78  %%saving the landmarks of this algorithm to use them in the
        lidar EKF SLAM
79  if isequal(KIND_LMK,'pnt')
80      lm=World.points(1:2,:);
81  end
82
83  %% III. Initialize data logging
84  % TODO: Create source and/or destination files and paths for
        data input and
85  % logs.
86  % TODO: do something here to collect data for post-processing or
87  % plotting. Think about collecting data in files using fopen,
        fwrite,
88  % etc., instead of creating large Matlab variables for data
        logging.
89
90  % Clear user data - not needed anymore
91  clear Robot Sensor World Time   % clear all user data
92  S=0;
93  %% IV. Main loop
94  original=SimRob.con.u;
```

```
95
96  for currentFrame = Tim.firstFrame : Tim.lastFrame
97      % 1. SIMULATION
98      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
99
100     % Simulate robots
101 %     pause
102     for rob = [SimRob.rob]
103
104         SS=locate_robot(SimRob(rob).state.x);
105         POS=SS(1:2);
106         SS(1:2)=[];
107         ORI=SS;
108         Det_Dir
109
110 % %         All landmarks are set visible
111 %         for i=1:length(Obs)
112 %             Obs(i).vis=true;
113 %         end
114
115         SimRob(rob) = simMotion(SimRob(rob),Tim);
116
117         % Simulate sensor observations
118         for sen = SimRob(rob).sensors
119
120             % Observe simulated landmarks
121             Raw(sen) = simObservation(SimRob(rob), SimSen(sen),
                    SimLmk, SimOpt) ;
122
123         end % end process sensors
124
125     end % end process robots
126
127     % 2. ESTIMATION
128     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
129
130     % Process robots
131     for rob = [Rob.rob]
132
133         % Robot motion
134         % NOTE: in a regular, non-simulated SLAM, this line is
                not here and
135         % noise just comes from the real world. Here, the
                estimated robot
136         % is noised so that the simulated trajectory can be made
                perfect
137         % and act as a clear reference. The noise is additive to
                the
138         % control input 'u'.
```

```
139         Rob(rob).con.u = SimRob(rob).con.u + Rob(rob).con.uStd.*
                randn(size(Rob(rob).con.uStd));
140         Rob(rob) = motion(Rob(rob),Tim);
141         Map.t = Map.t + Tim.dt;
142
143         % Process sensor observations
144         for sen = Rob(rob).sensors
145 %             DetVisLmk
146             % Observe known landmarks
147             [Rob(rob),Sen(sen),Lmk,Obs(sen,:),v,D,sss] =
                    correctKnownLmks( ...
148                 Rob(rob),   ...
149                 Sen(sen),   ...
150                 Raw(sen),   ...
151                 Lmk,        ...
152                 Obs(sen,:), ...
153                 Opt, SimLmk, SimRob,v) ;
154 %             pause
155             % Initialize new landmarks
156             ninits = Opt.init.nbrInits(1 + (currentFrame ~= Tim.
                    firstFrame));
157 %             DetVisLmk
158
159             for i = 1:ninits
160                 [Lmk,Obs(sen,:),v] = initNewLmk(...
161                     Rob(rob),   ...
162                     Sen(sen),   ...
163                     Raw(sen),   ...
164                     Lmk,        ...
165                     Obs(sen,:), ...
166                     Opt,v,SimLmk,SimRob,D,sss) ;
167             end
168
169         end % end process sensors
170
171     end % end process robots
172
173
174     % 3. VISUALIZATION
175     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
176
177     if currentFrame == Tim.firstFrame ...
178             || currentFrame == Tim.lastFrame ...
179             || mod(currentFrame,FigOpt.rendPeriod) == 0
180
181         % Figure of the Map:
182         MapFig = drawMapFig(MapFig,  ...
183             Rob, Sen, Lmk,  ...
184             SimRob, SimSen, ...
```

```
185             FigOpt);
186
187         if FigOpt.createVideo
188             makeVideoFrame(MapFig, ...
189                 sprintf('map-%05d.png',currentFrame), ...
190                 FigOpt, ExpOpt);
191         end
192
193         % Figures for all sensors
194         for sen = [Sen.sen]
195             SenFig(sen) = drawSenFig(SenFig(sen), ...
196                 Sen(sen), Raw(sen), Obs(sen,:), ...
197                 FigOpt);
198
199
200             if FigOpt.createVideo
201                 makeVideoFrame(SenFig(sen), ...
202                     sprintf('sen%02d-%05d.png', sen,
                            currentFrame),...
203                     FigOpt, ExpOpt);
204             end
205
206         end
207
208         % Do draw all objects
209         drawnow;
210     end
211
212
213     % 4. DATA LOGGING
214     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
215     % TODO: do something here to collect data for post-
            processing or
216     % plotting. Think about collecting data in files using fopen
            , fwrite,
217     % etc., instead of creating large Matlab variables for data
            logging.
218
219     %Real and estimated positions of the robots
220     for i=1:length(MapFig.Rob)
221         real_state{i}=[real_state{i}  SimRob(i).state.x];
222         est_state{i} =[est_state{i}   Rob(i).state.x];
223     end
224
225     if mod(currentFrame,20)==0
226         wp=[wp real_state{i}(1:2,currentFrame)];
227     end
228     if mod(currentFrame,NLOOPFRAME)==0
229         calc_RMSE
```

```
230         dataRMSE_lmk(jj)=RMSE_lmk;
231         dataRMSE_pos(jj)=RMSE_pos;
232         jj=jj+1;
233     end
234 end
235 %% V. Post-processing
236 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
237 % Enter post-processing code here
238 %savinf variables for the other algorithm
239 wp=[Position [Orientation;0] wp];
240 % save ('new_data.mat','lm','wp')
241 DIST_AP=5;
242 lim_x=LENGTH/2;
243 AP_x=-LENGTH/2+DIST_AP
244 AP_lm=[];
245 while AP_x<lim_x
246 AP_lm=[   AP_lm    [AP_x;-WIDTH/2+CORR/2]    [AP_x;+WIDTH/2-CORR
    /2]    ];
247 AP_x=AP_x+DIST_AP;
248 end
249 %%Calculating the MSE
250 calc_RMSE
251
252 %%Creating a video of the simulation related to the first 2
    robots
253 if CREATE_VIDEO==true
254     CreateVideoMap
255     CreateVideoSen1
256     CreateVideoSen2
257 end
258
259 save ('LMWP.mat','lm','wp','N_LOOPS','NLOOPFRAME','WIDTh','
    LENGTH','CORR','NDOOR_SIDE','AP_lm')
260 save (['new_data' num2str(PIX_ERROR_STD) '.mat' ],'dataRMSE_lmk'
    ,'dataRMSE_pos')
261
262 % ========== End of function - Start GPL license ==========
263
264
265 %   # START GPL LICENSE
266
267 %
    ---------------------------------------------------------------------
268 %
269 %   This file is part of SLAMTB, a SLAM toolbox for Matlab.
270 %
271 %   SLAMTB is free software: you can redistribute it and/or
    modify
```

```
272 %   it under the terms of the GNU General Public License as
    published by
273 %   the Free Software Foundation, either version 3 of the
    License, or
274 %   (at your option) any later version.
275 %
276 %   SLAMTB is distributed in the hope that it will be useful,
277 %   but WITHOUT ANY WARRANTY; without even the implied warranty
    of
278 %   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
    the
279 %   GNU General Public License for more details.
280 %
281 %   You should have received a copy of the GNU General Public
    License
282 %   along with SLAMTB.  If not, see <http://www.gnu.org/licenses
    />.
283 %
284 %   ------------------------------------------------------------------------
```

```
285
286 %   SLAMTB is Copyright 2007, 2008, 2009, 2010, 2011, 2012
287 %   by Joan Sola @ LAAS-CNRS.
288 %   SLAMTB is Copyright 2009
289 %   by Joan Sola, David Marquez and Jean Marie Codol @ LAAS-CNRS
    .
290 %   See on top of this file for its particular copyright.
291
292 %   # END GPL LICENSE
```

script userData_corridor.m

```
1  % USERDATA  User data for SLAMTB.
2  %   Edit this script to enter the information you need for SLAM.
       Variable
3  %   names and comments should make this file easy to understand.
       Follow
4  %   these guidelines:
5  %
6  %   * Specify site and estimation details for the current run in
       ExpOpt.
7  %   * Specify sampling time and start and end frames in Tim.
8  %   * Use as many robots and sensors as you wish with Robot{}
    and Sensor{}.
9  %   * Assign sensors to robots via Sensor{i}.robot.
10 %   * Use field Sensor{i}.distortion for radial distortion
```

```
    parameters if
11  %   desired.
12  %   * Use the field Opt.map.numLmk and .lmkSize to specify the
    maximum
13  %   number of landmarks that the SLAM map must support.
14  %   * Use Opt.init.initType to select the type of landmarks to
    use. Try
15  %   with one in this list:
16  %       'idpPnt', 'hmgPnt', 'ahmPnt', 'plkLin', 'ahmLin'.
17  %   * Use World.points and World.segments to create artificial
    worlds of
18  %   points or segments. Check functions THICKCLOISTER and HOUSE.
19  %
20  %   See further comments within the file for more detailed
    information.
21  %
22  %   NOTE: You can have multiple copies of this file with
    different names to
23  %   store different simulation conditions. Just modify the call
    in SLAMTB
24  %   to point to the particular 'USERDATA' file you want.
25  %
26  %   See also SLAMTB, EULERANGLES, THICKCLOISTER, HOUSE,
    USERDATAPNT,
27  %   USERDATALIN.
28
29  %   Copyright 2008-2009 Joan Sola @ LAAS-CNRS.
30
31  % Time variables
32  %   - sampling time, first and last frames
33
34  global LENGTH
35  global WIDTh
36  global HEIGHT
37  global CORR
38  global NDOOR_SIDE
39
40  LENGTH     = 40;
41  WIDTh      = 20;
42  HEIGHT     = 4;
43  CORR       = 6;
44  NDOOR_SIDE = 3;
45
46
47  Time = struct(...
48    'dt',                   .1,...          % sampling time,
       seconds
49    'firstFrame',           1,...           % first frame #
50    'lastFrame',            LAST_FRAME);           % last frame #
```

```
51
52 % Simulated world
53 % - Simulation landmark sets, playground dimensions
54
55
56 lm=[[-LENGTH/2;WIDTh/2;HEIGHT/2] [-LENGTH/2;-WIDTh/2;HEIGHT
       /2]...%EXTERNAL WALLS EDGES
57      [LENGTH/2;WIDTh/2;HEIGHT/2] [LENGTH/2;-WIDTh/2;HEIGHT/2]...
58      [-LENGTH/2+CORR;WIDTh/2-CORR;HEIGHT/2] [-LENGTH/2+CORR;-
           WIDTh/2+CORR;HEIGHT/2]...%INTERNAL WALLS EDGES
59      [-CORR/2;WIDTh/2-CORR;HEIGHT/2] [-CORR/2;-WIDTh/2+CORR;
           HEIGHT/2]...
60      [LENGTH/2-CORR;WIDTh/2-CORR;HEIGHT/2] [LENGTH/2-CORR;-WIDTh
           /2+CORR;HEIGHT/2]...
61      [+CORR/2;WIDTh/2-CORR;HEIGHT/2] [+CORR/2;-WIDTh/2+CORR;
           HEIGHT/2]];
62 door_lm=linspace(-LENGTH/2,LENGTH/2,NDOOR_SIDE+2);
63 door_lmy=linspace(-WIDTh/2,WIDTh/2,round(NDOOR_SIDE/2+2));
64 door_lm=door_lm(2:end-1);
65 door_lmy=door_lmy(2:end-1);
66 door_lma=door_lm;
67 door_lmay=door_lmy;
68 door_lm=[door_lm;-WIDTh/2*ones(1,length(door_lm));HEIGHT/2*ones
       (1,length(door_lm))];
69 door_lma=[door_lma;WIDTh/2*ones(1,length(door_lma));HEIGHT/2*
       ones(1,length(door_lma))];
70
71 door_lmy=[-LENGTH/2*ones(1,length(door_lmy));door_lmy;HEIGHT/2*
       ones(1,length(door_lmy))];
72 door_lmay=[LENGTH/2*ones(1,length(door_lmay));door_lmay;HEIGHT
       /2*ones(1,length(door_lmay))];
73 lm=[lm door_lm door_lma door_lmy door_lmay];
74
75
76
77 World = struct(...
78   'points',           lm,... % 3d point landmarks - see
         THICKCLOISTER. GenLmk(CHOICE_2D_3D,NPTS,WIDTH)
79   'segments',         Corridor_Ing_Ces(LENGTH,WIDTh,HEIGHT,CORR,
         door_lm));                  % 3D segments - see HOUSE. house
         (-2,-2,0)
80
81 % Robot things with their controls
82 %   - each robot's type and initial configuration, and controls.
83 %   - motion models (add new model strings if you need more):
84 %       'constVel'    6D Constant velocity model
85 %       'odometry'    6D Odometry model
86 %       'inertial'    6D IMU-based model
87 %   - See EULERANGLES for orientations specifications.
```

```
88
89
90  Robot{1} = struct(...                    % ODOMETRY EXAMPLE
91    'id',                  1,...            % robot identifier
92    'name',                'Dala',...       % robot name
93    'type',                'atrv',...       % type of robot
94    'motion',              'odometry',...   % motion model
95    'position',            [LENGTH/2-CORR;-(WIDTh-CORR)/2;0],...
            % robot position in map
96    'orientationDegrees', [0;0;0],...      % orientation, in
         degrees, [roll; pitch; yaw].
97    'positionStd',         [POS_STD;0],...     % position error,
         std
98    'orientationStd',      [0;0;0],...      % orient. error, std, in
          degrees
99    'dx',                  [.08;0;0],...      % position increment
100   'daDegrees',           [0;0;1.8],...      % angle increment,
         degrees
101   'dxStd',               0.005*[1;1;0],...  % odo linear error
         std
102   'daStd',               0.05*[1;1;1]);      % odo ang error std,
          degrees
103
104 % Sensor things
105 %   - each sensor's type and parameters, noise, non-measurable
       prior.
106 %   - Sensor types (add new type strings if you need more):
107 %       'pinHole'   Pin-hole camera
108 %   - See EULERANGLES for orientations specifications.
109
110 Sensor{1} = struct(...
111   'id',                  1,...            % sensor identifier
112   'name',                'Micropix',...   % sensor name
113   'type',                'pinHole',...    % type of sensor
114   'robot',               1,...            % robot where it is
         mounted
115   'position',            [0;0;HEIGHT_SENSOR],...    % position in
          robot
116   'orientationDegrees', [-90;0;-90],... % orientation in robot,
         [roll; pitch; yaw]
117   'positionStd',         [0;0;0],...      % position error std
118   'orientationStd',      [0;0;0],...      % orient. error std
119   'imageSize',           [640;480],...    % image size
120   'pixErrorStd',         PIX_ERROR_STD,...          % pixel error
         std
121   'intrinsic',           [320;240;320;320],... % intrinsic params
          [u0 v0 au av]
122   'distortion',          [-0.3;0.1],...           % distortion
         params
```

```
123   'frameInMap',          false,...       % add sensor frame in
         slam map?
124   'imGrid',                struct(...      % grid for Active
         Search
125     'numCells',          [8;6],...         % number of H and V
           grid cells
126     'skipOuter',         true));           % skip outer cells for
           initialization?
127
128
129 % Omnidirectional camera model -> MegaPixel Fish Eye lens as
      example (FoV is ~190 deg )
130 % k           = [668 438 1 0 0]';
131 % invProjDist = [4.473e+2 -0.000e+0 -1.068e-3 1.184e-6 -1.856e
      -9]';
132 % projDist    = [6.447e+2 -3.410e+2 -2.901e+1 -5.770e+1 1.849e+1
       5.415e+0 5.065e+1 -5.614e+1 1.591e+1 0 0]';
133 % Sensor{1} = struct(...
134 %   'id',                  1,...           % sensor identifier
135 %   'name',                'FrontCam',...  % sensor name
136 %   'type',                'omniCam',...   % type of sensor
137 %   'robot',               1,...           % robot where it is
      mounted
138 %   'position',            [0.2;0;1.2],... % position in robot
139 %   'orientationDegrees', [-120;0;-90],...% orientation in robot
      , [roll; pitch; yaw].
140 %   'positionStd',         [0;0;0],...     % position error std
141 %   'orientationStd',      [0;0;0],...     % orient. error std
142 %   'imageSize',           [1280;800],...  % image size
143 %   'pixErrorStd',         1.0,...         % pixel error std
144 %   'intrinsic',           k,...           % intrinsic params: [
      xc yc c d e]';
145 %   'distortion',          projDist,...    % distortion params ->
       polynom for projection to cam sensor
146 %   'invDistortion',       invProjDist,... % distortion params ->
       polynom for inv proj from cam sensor
147 %   'frameInMap',          false,...       % add sensor frame in
      slam map?
148 %   'imGrid',              struct(...      % grid for Active
      Search
149 %     'numCells',          [8;6],...       % number of H and V
      grid cells
150 %     'skipOuter',         true));         % skip outer cells for
       initialization?
151
152
153
154 % Estimation options
155 Opt = struct(...
```

```
156   'map',              struct(...    % options for the map
157     'numLmks',        NLMKS_ROBOT,...         % number of 3d
          landmarks
158     'lmkSize',        LMK_SIZE),...          % Size of landmark
159   'correct',          struct(...    % options for lmk correction
160     'reprojectLmks',  true,...       % reproject lmks after
          active search?
161     'reparametrize',  true,...       % reparametrize lmk?
162     'nUpdates',       10,...         % max simultaneus updates
163     'MD2th',          9,...          % Threshold on Mahalanobis
          distance squared
164     'linTestIdp',     0.1,...        % threshold on IDP
          linearity test
165     'lines',          struct(...     % options for line
          corrections
166       'innType',      'ortDst',...    % innovation type for
            lines
167       'extPolicy',    false,...       % line extending policy ?
168       'extSwitch',    10)),...        % extension policy switch
            point in pixels
169   'init',             struct(...    % Options for initialization
170     'nbrInits',       [1 1],...      % number of inits [
          firstFrame, otherFrames]
171     'initType',       'idpPnt',...   % Type of lmk to use for
          init
172     'idpPnt',         struct(...     % options for lmk
          initialization
173       'nonObsMean',   .01,...         % mean of non obs
174       'nonObsStd',    .5),...         % std of non obs
175     'idpLin',         struct(...     % opt. for Plucker and
          anchored Plucker lines init
176       'nonObsMean',   [.1;0],...      % mean of non obs
177       'nonObsStd',    [.25;1])),...   % std of non obs
178   'obs',              struct(...    % Observation options
179     'lines',          struct(...     % lines options
180       'minLength',    20)));          % minimum segment length
181
182
183 % Simulation options
184 %   - random
185 SimOpt = struct(...
186   'random',           struct(...      % random generator options
187     'newSeed',        false,...         % select new random seed
          ?
188     'fixedSeed',      208948,...            % random seed for
          non-random runs
189     'seed',           []),...          % current seed
190   'obs',              Opt.obs);       % Observation options
191
```

```
192
193
194 % Figure options
195 %   - view, projection, video, ellipses.
196 %   - figure projections - mapProj:
197 %       'persp'     Perspective
198 %       'ortho'     Orthographic
199 %   - 3D figure views - mapView - see MAPOBSERVER.
200 %       [a,e,f]     Custom azimuth/elevation/FOV vector.
    Distance automatic
201 %       [a,e,f,d]   custom az/el/fov/distance vector.
202 %   - 3D figure predefined views (edit mapObserver.m to create/
    edit views):
203 %       'top'       Top view
204 %       'side'      Side view
205 %       'view'      Generic view
206 %       'normal'    Normal view
207 %   - objects colors - two options for color specification:
208 %       'rgbcmykw'  1-char predifined Matlab colors
209 %       [r g b]     RGB color vector. [0 0 0] is black, [1 1 1]
    is white.
210 FigOpt = struct(...
211   'renderer',       'zbuffer',...    % renderer
212   'rendPeriod',     1,...           % frames to skip for faster
      processing
213   'createVideo',    CREATE_VIDEO,...       % create video
      sequences?
214   'map',            struct(...      % map figure options
215     'size',         [1280 960],...   % map figure size
216     'lims',         struct(...      % playground limits
217       'xMin',           -LENGTH/2-5,...
218       'xMax',            LENGTH/2+5,...
219       'yMin',           -WIDTh/2-5,...
220       'yMax',            WIDTh/2+5,...
221       'zMin',           -HEIGHT/2-5,...
222       'zMax',            HEIGHT/2+5),...
223     'proj',         'persp',...     % projection of the 3d
        figure
224     'view',         'view',...      % viewpoint of the 3d figure
         [30 45 40 20]
225     'orbit',        [0 0],...       % Azimuth and Elevation
        orbit angle increments - use to animate figure
226     'showSimLmk',   true,...        % show simulated landmarks?
227     'showEllip',    true,...        % show ellipsoids?
228     'colors',       struct(...      % map figure colors
229       'border',     [1 1 1],...      %   [r g b]
230       'axes',       [0 0 0],...      % with:
231       'bckgnd',     [1 1 1],...      %   [0 0 0] black
232       'simLmk',     .3*[1 1 1],...   %   [1 1 1] white
```

```
233       'defPnt',     struct(...       % euclidean point colors
234         'mean',     'b',...          % mean dot
235         'ellip',    [.7 .7 1]),...   % ellipsoid
236       'othPnt',     struct(...       % other point colors
237         'mean',     'r',...          % mean dot
238         'ellip',    [1 .7 .7]),...   % ellipsoid
239       'defLin',     struct(...       % Plucker line colors
240         'mean',     [0 .8 0],...     % mean line
241         'ellip',    [.6 1 .6]),...   % ellipsoid
242       'othLin',     struct(...       % Plucker line colors
243         'mean',     [.8 0 0],...     % mean line
244         'ellip',    [1 .6 .6]),...   % ellipsoid
245       'simu',       'b',...          %   or 'r', 'b', etc.
246       'est',        'g',...          % estimated robots and
            sensors
247       'ground',     [.8 .8 .8],...   % simulated robots and
            sensors
248       'label',      [.0 .5 0])),...  % landmark ID labels
249     'sensor',       struct(...      % sensor figures options
250       'size',       [320 240],...    % sensor figure size
251       'showEllip',  false,...        % show ellipses?
252       'colors',     struct(...       % Sensor figure colors:
253         'border',   .8*[1 1 1],...   %
254         'axes',     [0 0 0],...      %
255         'bckgnd',   [1 1 1],...      %
256         'raw',      .3*[1 1 1],...   %
257         'defPnt',   struct(...       % Default point colors
258           'updated',  'c',...        % updated
259           'predicted','b'),...       % predicted
260         'othPnt',   struct(...       % other point colors
261           'updated',  'r',...        % updated
262           'predicted','m'),...       % predicted
263         'defLin',   struct(...       % Default line colors
264           'meas',   'b',...          % measurement
265           'mean',   'g',...          % mean line
266           'ellip',  'y'),...         % ellipsoid
267         'othLin',   struct(...       % other line colors
268           'meas',   'b',...          % measurement
269           'mean',   'm',...          % mean line
270           'ellip',  'r'),...         % ellipsoid
271         'label',    [.5 .5 .5])));   %
272
273
274 % Experiment options
275 %   - site name, series gathered, estimation run number
276 ExpOpt = struct(...
277   'root',         '~/SLAM/',...         % root directory
278   'site',         'simu',...            % Name of the site
279   'dataRun',      1,...                 % Run # on this site
```

```
280   'estimateRun',  1,...                  % slam run for data and
          site
281   'lmkTypes',     Opt.init.initType,... % types of landmarks
          used
282   'sensingType',  'mono',...             % sensing mode
283   'mappingType',  'single');             % mapping mode
284
285
286
287 % ========== End of function - Start GPL license ==========
288
289
290 %   # START GPL LICENSE
291
292 %
      ---------------------------------------------------------------------
293 %
294 %   This file is part of SLAMTB, a SLAM toolbox for Matlab.
295 %
296 %   SLAMTB is free software: you can redistribute it and/or
      modify
297 %   it under the terms of the GNU General Public License as
      published by
298 %   the Free Software Foundation, either version 3 of the
      License, or
299 %   (at your option) any later version.
300 %
301 %   SLAMTB is distributed in the hope that it will be useful,
302 %   but WITHOUT ANY WARRANTY; without even the implied warranty
      of
303 %   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
      the
304 %   GNU General Public License for more details.
305 %
306 %   You should have received a copy of the GNU General Public
      License
307 %   along with SLAMTB.  If not, see <http://www.gnu.org/licenses
      />.
308 %
309 %
      ---------------------------------------------------------------------
310
311 %   SLAMTB is Copyright 2007, 2008, 2009, 2010, 2011, 2012
312 %   by Joan Sola @ LAAS-CNRS.
313 %   SLAMTB is Copyright 2009
314 %   by Joan Sola, David Marquez and Jean Marie Codol @ LAAS-CNRS
      .
```

```
315 %   See on top of this file for its particular copyright.
316
317 %   # END GPL LICENSE
```

script Corridor_Ing_Ces.m

```
1  function Corr_Seg=Corridor_Ing_Ces(len,width,height,corr,doors)
2  hold on
3
4  %Defining Points
5  A=[-len/2;-width/2;0];
6  B=[len/2;-width/2;0];
7  C=[len/2;-width/2;height];
8  D=[-len/2;-width/2;height];
9  E=[-len/2;width/2;0];
10 F=[len/2;width/2;0];
11 G=[len/2;width/2;height];
12 H=[-len/2;width/2;height];
13 I=[-len/2+corr;-width/2+corr;height];
14 J=[-corr/2;-width/2+corr;height];
15 K=[-corr/2;width/2-corr;height];
16 L=[-len/2+corr;width/2-corr;height];
17 M=[-len/2+corr;-width/2+corr;0];
18 N=[-corr/2;-width/2+corr;0];
19 O=[-corr/2;width/2-corr;0];
20 P=[-len/2+corr;width/2-corr;0];
21 Q=[corr/2;-width/2+corr;height];
22 R=[len/2-corr;-width/2+corr;height];
23 S=[len/2-corr;width/2-corr;height];
24 T=[corr/2;width/2-corr;height];
25 U=[corr/2;-width/2+corr;0];
26 V=[len/2-corr;-width/2+corr;0];
27 Z=[len/2-corr;width/2-corr;0];
28 Y=[corr/2;width/2-corr;0];
29
30 %Defining segments; DH AE GC FB must be commented to simulate a
       uninterrupted corridor
31 Corr_Seg=[[A;D] [A;B] [A;E] [B;C] [B;F] [C;G] [C;D] [D;H] [E;H]
       [E;F] [F;G] [G;H],...
32            [M;N] [M;P] [M;I] [I;J] [J;N] [J;K] [I;L] [N;O] [L;K]
                  [O;K] [O;P] [P;L],...
33            [T;S] [S;R] [R;Q] [T;Q] [Q;U] [U;V] [Z;V] [Z;Y] [T;Y]
                  [U;Y] [R;V] [S;Z],...
34            ];
35 end
```

script locate_robot.m

```
function SS = locate_robot(state)

%This function is used to determine the state of the robot in the map in
%terms of orientation and position; this is then used to determine the
%actions to take to move the robot in the proper way along the corridor

global LENGTH
global WIDTh
global HEIGHT
global CORR
global NDOOR_SIDE

LENGTH     = 40;
WIDTh      = 20;
HEIGHT     = 4;
CORR       = 6;
NDOOR_SIDE = 3;


%Determining orientation
ang=q2e(state(4:7));
ang=rad2deg(ang(3));


if ang>-5 && ang<5
    ORI='ORI0';
elseif ang>40 && ang<50
    ORI='ORI45';
elseif ang>85 && ang<95
    ORI='ORI90';
elseif ang>130 && ang<140
    ORI='ORI135';
elseif (ang>175 && ang<185)||(ang>-185 && ang<-175)
    ORI='ORI180';
elseif ang>-50 && ang<-40
    ORI='ORI-45';
elseif ang>-95 && ang<-85
    ORI='ORI-90';
elseif ang>-140 && ang<-130
    ORI='ORI-135';
else
    ORI='ORIOTH';
```

```
end

%determining position along the axis
if state(2)<-WIDTh/2+CORR
    POSY='A';
elseif state(2)<WIDTh/2-CORR
    POSY='B';
else
    POSY='C';
end
if state(1)<-LENGTH/2+CORR
    POSX='A';
elseif state(1)<-CORR/2
    POSX='B';
elseif state(1)<CORR/2
    POSX='C';
elseif state(1)<LENGTH/2-CORR
    POSX='D';
else
    POSX='E';
end

SS=[POSX POSY ORI];
end
```

script Det_Dir.m

```
% This code is used to determine the direction of the robot in
   the case
% of 'odometry' model of motion

% Original control action is saved
SimRob.con.u=original;
        switch POS  %When the robot is not at the corners, it
           must go forward without changing direction
            case 'CA'
                original=SimRob.con.u;
                SimRob.con.u(6)=0;
            case 'CC'
                original=SimRob.con.u;
                SimRob.con.u(6)=0;
            case 'BA'
                original=SimRob.con.u;
                SimRob.con.u(6)=0;
            case 'DA'
                original=SimRob.con.u;
                SimRob.con.u(6)=0;
```

```
            case 'EB'
                original=SimRob.con.u;
                SimRob.con.u(6)=0;
            case 'DC'
                original=SimRob.con.u;
                SimRob.con.u(6)=0;
            case 'BC'
                original=SimRob.con.u;
                SimRob.con.u(6)=0;
            case 'AB'
                original=SimRob.con.u;
                SimRob.con.u(6)=0;

            case 'AA'     %when the robot arrives at the corners
                , it must change direction
                switch ORI   %until the right direction is
                    obtained, then the direction must be kept
                    case 'ORI0'
                                original=SimRob.con.u;
                                SimRob.con.u(6)=0;
                end
            case 'EA'
                switch ORI
                    case 'ORI90'
                                original=SimRob.con.u;
                                SimRob.con.u(6)=0;
                end
            case 'EC'
                switch ORI
                    case 'ORI180'
                                original=SimRob.con.u;
                                SimRob.con.u(6)=0;
                end
            case 'AC'
                switch ORI
                    case 'ORI-90'
                                original=SimRob.con.u;
                                SimRob.con.u(6)=0;
                end
        end
```

script DetVisLmk.m

```
%This code disables landmarks that are not visibile in the real
    environment
sss=[];
%Bottom corridor
```

```
4  if SimRob.state.x(1)<LENGTH/2-CORR && SimRob.state.x(2)<-WIDTh
       /2+CORR
5      sss=1;
6      cat2=(-WIDTh/2+CORR)-SimRob.state.x(2);
7      cat1=LENGTH/2-CORR-SimRob.state.x(1);
8      ang_vis=atan(cat2/cat1);
9      D=-WIDTh/2+CORR*(1+tan(ang_vis));
10     for i=1:length(Obs)
11         if ~isempty(Obs(i).lid)
12             if SimLmk.points.coord(2,Obs(i).lid)>D
13                 v(i)=1;
14             else
15                 v(i)=0;
16             end
17         end
18         if v(i)==1
19             Obs(i).vis=false;
20         end
21     end
22 end
23
24 if SimRob.state.x(1)>-LENGTH/2+CORR && SimRob.state.x(2)>WIDTh
       /2-CORR
25     sss=3;
26     cat2=SimRob.state.x(2)-(WIDTh/2-CORR);
27     cat1=SimRob.state.x(1)-(-LENGTH/2+CORR);
28     ang_vis=atan(cat2/cat1);
29     D=WIDTh/2-CORR*(1+tan(ang_vis));
30     for i=1:length(Obs)
31         if ~isempty(Obs(i).lid)
32             if SimLmk.points.coord(2,Obs(i).lid)<D
33                 v(i)=1;
34             else
35                 v(i)=0;
36             end
37         end
38         if v(i)==1
39             Obs(i).vis=false;
40         end
41     end
42 end
43
44 if SimRob.state.x(2)<WIDTh/2-CORR && SimRob.state.x(1)>LENGTH/2-
       CORR
45     sss=2;
46     cat2=SimRob.state.x(1)-LENGTH/2+CORR;
47     cat1=WIDTh/2-CORR-SimRob.state.x(2);
48     ang_vis=atan(cat2/cat1);
49     D=LENGTH/2-CORR*(1+tan(ang_vis));
```

```
50      for i=1:length(Obs)
51          if ~isempty(Obs(i).lid)
52              if SimLmk.points.coord(1,Obs(i).lid)<D
53                  v(i)=1;
54              else
55                  v(i)=0;
56              end
57          end
58          if v(i)==1
59              Obs(i).vis=false;
60          end
61      end
62  end
63
64  if SimRob.state.x(2)>-WIDTh/2+CORR && SimRob.state.x(1)<-LENGTH
        /2+CORR
65      sss=4;
66      cat2=-LENGTH/2+CORR-SimRob.state.x(1);
67      cat1=SimRob.state.x(2)-(-WIDTh/2+CORR);
68      ang_vis=atan(cat2/cat1);
69      D=-LENGTH/2+CORR*(1+tan(ang_vis));
70      for i=1:length(Obs)
71          if ~isempty(Obs(i).lid)
72              if SimLmk.points.coord(1,Obs(i).lid)>D
73                  v(i)=1;
74              else
75                  v(i)=0;
76              end
77          end
78          if v(i)==1
79              Obs(i).vis=false;
80          end
81      end
82  end
83
84  % debug
85  c=zeros(2,length(Obs));
86  for i=1:length(c)
87      if Obs(i).vis==true
88          c(1,i)=0;
89      else
90          c(1,i)=1;
91      end
92      if ~isempty(Obs(i).lid)
93      c(2,i)=Obs(i).lid;
94      end
95  end
96  % c
```

script calc_RMSE.m

```
num_MSE_lmk=0;
num_MSE_pos=0;

for i=1:length(SimLmk.points.coord)
    if ~isempty(MapFig.Lmk(i).mean.XData)
        coord_meas(1,str2num(MapFig.Lmk(i).label.String))=MapFig
            .Lmk(i).mean.XData;
        coord_meas(2,str2num(MapFig.Lmk(i).label.String))=MapFig
            .Lmk(i).mean.YData;
    end

end
err_lmk=SimLmk.points.coord(1:2,:)-coord_meas;
for i=1:length(SimLmk.points.coord)
    num_MSE_lmk=num_MSE_lmk+err_lmk(1,i)^2+err_lmk(2,i)^2;%+
        err_lmk(3,i)^2
end
RMSE_lmk=sqrt(num_MSE_lmk/length(SimLmk.points.coord))

err_pos=cell(1,length(MapFig.Rob));
for i=1:length(MapFig.Rob)
    err_pos{i}=real_state{i}-est_state{i};
end

for i=1:length(real_state{i})
    for j=1:length(MapFig.Rob)
    num_MSE_pos=num_MSE_pos+err_pos{j}(1,i)^2+err_pos{j}(2,i)^2;
        %+err_pos{j}(3,i)^2
    end
end

RMSE_pos=sqrt(num_MSE_pos/(length(real_state{1}*length(MapFig.
    Rob))))
```

Appendix B

MATLAB code for LASER SLAM

script configfile.m

```
%%% Configuration file
%%% Permits various adjustments to parameters of the SLAM
    algorithm.
%%% See ekfslam_sim.m for more information
SCALE_FACTOR=1
ORIENTATION=Orientation*SCALE_FACTOR;
POSITION=Position*SCALE_FACTOR;
wp=wp.*SCALE_FACTOR;
lm=lm.*SCALE_FACTOR;
% control parameters
V= 1; % m/s
MAXG= 360*pi/180; % radians, maximum steering angle (-MAXG < g <
     MAXG)
RATEG= 180*pi/180; % rad/s, maximum rate of change in steer
    angle
WHEELBASE= 4; % metres, vehicle wheel-base
SCALE_VEHICLE=4;%scaling vehicle animation
DT_CONTROLS= 0.025; % seconds, time interval between control
    signals

% control noises
sigmaV= 0.1; % m/s
sigmaG= (0.1*pi/180); % radians
Q= [sigmaV^2 0; 0 sigmaG^2];

% observation parameters
MAX_RANGE= 5.0; % metres
```

```
24 DT_OBSERVE= 8*DT_CONTROLS; % seconds, time interval between
       observations
25
26 % observation noises
27 sigmaR= 1; % metres
28 sigmaB= (10*pi/180); % radians
29 R= [sigmaR^2 0; 0 sigmaB^2];
30
31 % data association innovation gates (Mahalanobis distances)
32 GATE_REJECT= 4.0; % maximum distance for association
33 GATE_AUGMENT= 25.0; % minimum distance for creation of new
       feature
34 % For 2-D observation:
35 %   - common gates are: 1-sigma (1.0), 2-sigma (4.0), 3-sigma
       (9.0), 4-sigma (16.0)
36 %   - percent probability mass is: 1-sigma bounds 40%, 2-sigma
       86%, 3-sigma 99%, 4-sigma 99.9%.
37
38 % waypoint proximity
39 AT_WAYPOINT= 0.1; % metres, distance from current waypoint at
       which to switch to next waypoint
40 NUMBER_LOOPS= 3; % number of loops through the waypoint list
41
42 % switches
43 SWITCH_CONTROL_NOISE= 1; % if 0, velocity and gamma are perfect
44 SWITCH_SENSOR_NOISE = 1; % if 0, measurements are perfect
45 SWITCH_INFLATE_NOISE= 0; % if 1, the estimated Q and R are
       inflated (ie, add stabilising noise)
46 SWITCH_HEADING_KNOWN= 0; % if 1, the vehicle heading is observed
        directly at each iteration
47 SWITCH_ASSOCIATION_KNOWN= 1; % if 1, associations are given, if
       0, they are estimated using gates
48 SWITCH_BATCH_UPDATE= 1; % if 1, process scan in batch, if 0,
       process sequentially
49 SWITCH_SEED_RANDOM= 0; % if not 0, seed the randn() with its
       value at beginning of simulation (for repeatability)
50 SWITCH_USE_IEKF= 1; % if 1, use iterated EKF for updates, if 0,
       use normal EKF
51 SWITCH_PROFILE= 0; % if 1, turn on MatLab profiling to measure
       time consumed by simulator functions
52 SWITCH_GRAPHICS= 1; % if 0, avoids plotting most animation data
       to maximise simulation speed
```

script ekfslam_sim.m

```
1 function data= ekfslam_sim(lm, wp,WIDTh,LENGTH,CORR)
2 %function data= ekfslam_sim(lm, wp)
```

```
3  %
4  % INPUTS:
5  %   lm - set of landmarks
6  %   wp - set of waypoints
7  %
8  % OUTPUTS:
9  %   data - a data structure containing:
10 %       data.true: the vehicle 'true'-path (ie, where the
       vehicle *actually* went)
11 %       data.path: the vehicle path estimate (ie, where SLAM
       estimates the vehicle went)
12 %       data.state(k).x: the SLAM state vector at time k
13 %       data.state(k).P: the diagonals of the SLAM covariance
       matrix at time k
14 %
15 % NOTES:
16 %   This program is a SLAM simulator. To use, create a set of
       landmarks and
17 %   vehicle waypoints (ie, waypoints for the desired vehicle
       path). The program
18 %   'frontend.m' may be used to create this simulated
       environment - type
19 %   'help frontend' for more information.
20 %       The configuration of the simulator is managed by the
       script file
21 %   'configfile.m'. To alter the parameters of the vehicle,
       sensors, etc
22 %   adjust this file. There are also several switches that
       control certain
23 %   filter options.
24 %
25 % Tim Bailey and Juan Nieto 2004.
26 % Version 2.0
27
28 format compact
29
30 Orientation = wp(1,2)
31 Position    = wp(:,1)
32 wp=wp(:,3:end)
33 configfile; % ** USE THIS FILE TO CONFIGURE THE EKF-SLAM **
34
35 % Setup plots
36 fig=figure;
37 hold on
38 plot([-LENGTH/2, -LENGTH/2, LENGTH/2, LENGTH/2, -LENGTH/2],...
39     [-WIDTh/2, WIDTh/2, WIDTh/2, -WIDTh/2, -WIDTh/2],'k')
40 plot([-LENGTH/2+CORR, -LENGTH/2+CORR, -CORR/2, -CORR/2, -LENGTH
       /2+CORR],...
41     [-WIDTh/2+CORR, WIDTh/2-CORR, WIDTh/2-CORR, -WIDTh/2+CORR, -
```

```
        WIDTh/2+CORR],'k')
42 plot([LENGTH/2-CORR, LENGTH/2-CORR, CORR/2, CORR/2, LENGTH/2-
    CORR],...
43      [WIDTh/2-CORR, -WIDTh/2+CORR, -WIDTh/2+CORR, WIDTh/2-CORR,
        WIDTh/2-CORR],'k')
44
45 plot(lm(1,:),lm(2,:),'b*')
46 hold on, axis equal, axis([-30 30 -20 20])
47 plot(wp(1,:),wp(2,:), 'g', wp(1,:),wp(2,:),'g.')
48
49 xlabel('metres'), ylabel('metres')
50 set(fig, 'name', 'EKF-SLAM Simulator')
51 h= setup_animations;
52 veh= [0 -WHEELBASE/SCALE_VEHICLE -WHEELBASE/SCALE_VEHICLE; 0 -2/
    SCALE_VEHICLE 2/SCALE_VEHICLE]; % vehicle animation
53 plines=[]; % for laser line animation
54 pcount=0;
55
56 % Initialise states and other global variables
57 global XX PX DATA
58 % xtrue=zeros(3,1);
59 xtrue= POSITION;
60 xtrue(3,1)=ORIENTATION;
61
62 XX= xtrue;
63 PX= zeros(3);
64 DATA= initialise_store(XX,PX,XX); % stored data for off-line
65
66 % Initialise other variables and constants
67 dt= DT_CONTROLS;        % change in time between predicts
68 dtsum= 0;               % change in time since last observation
69 ftag= 1:size(lm,2);     % identifier for each landmark
70 da_table= zeros(1,size(lm,2)); % data association table
71 iwp= 1;                 % index to first waypoint
72 G= 0;                   % initial steer angle
73 QE= Q; RE= R; if SWITCH_INFLATE_NOISE, QE= 2*Q; RE= 2*R; end %
    inflate estimated noises (ie, add stabilising noise)
74 if SWITCH_SEED_RANDOM, rand('state',SWITCH_SEED_RANDOM), randn('
    state',SWITCH_SEED_RANDOM), end
75
76 if SWITCH_PROFILE, profile on -detail builtin, end
77
78 % Main loop
79 while iwp ~= 0
80
81     % Compute true data
82     [G,iwp]= compute_steering(xtrue, wp, iwp, AT_WAYPOINT, G,
        RATEG, MAXG, dt);
83
```

```
84      if iwp==0 & NUMBER_LOOPS > 1, pack; iwp=1; NUMBER_LOOPS=
            NUMBER_LOOPS-1; end % perform loops: if final waypoint
            reached, go back to first
85      xtrue= vehicle_model(xtrue, V,G, WHEELBASE/SCALE_VEHICLE,dt)
            ;
86
87      [Vn,Gn]= add_control_noise(V,G,Q, SWITCH_CONTROL_NOISE);
88
89      % EKF predict step
90      predict (Vn,Gn,QE, WHEELBASE/SCALE_VEHICLE,dt);
91
92      % If heading known, observe heading
93      observe_heading(xtrue(3), SWITCH_HEADING_KNOWN);
94
95      % Incorporate observation, (available every DT_OBSERVE
            seconds)
96      dtsum= dtsum + dt;
97      if dtsum >= DT_OBSERVE
98          dtsum= 0;
99
100         % Compute true data
101         [z,ftag_visible]= get_observations(xtrue, lm, ftag,
                MAX_RANGE);
102         % z contiene la distanza esatta da tutti landmark
                visibili nel
103         % semicerchio orientato secondo x(3)
104         z= add_observation_noise(z,R, SWITCH_SENSOR_NOISE);
105         % aggiunge rimore gaussiano con varianza R
106
107         % EKF update step
108         if SWITCH_ASSOCIATION_KNOWN == 1
109             [zf,idf,zn, da_table]= data_associate_known(XX,z,
                    ftag_visible, da_table);
110         else
111             [zf,idf, zn]= data_associate(XX,PX,z,RE, GATE_REJECT
                    , GATE_AUGMENT);
112         end
113
114         if SWITCH_USE_IEKF == 1
115             update_iekf(zf,RE,idf, 5);
116         else
117             update(zf,RE,idf, SWITCH_BATCH_UPDATE);
118         end
119         augment(zn,RE);
120     end
121
122     % Offline data store
123     store_data(XX, PX, xtrue);
124
```

```
125     % Plots
126     xt= transformtoglobal(veh, xtrue);
127     set(h.xt, 'xdata', xt(1,:), 'ydata', xt(2,:))
128
129     if SWITCH_GRAPHICS
130         xv= transformtoglobal(veh, XX(1:3));
131         pvcov= make_vehicle_covariance_ellipse(XX,PX);
132         set(h.xv, 'xdata', xv(1,:), 'ydata', xv(2,:))
133         set(h.vcov, 'xdata', pvcov(1,:), 'ydata', pvcov(2,:))
134
135         pcount= pcount+1;
136         if pcount == 120 % plot path infrequently
137             pcount=0;
138             set(h.pth, 'xdata', DATA.path(1,1:DATA.i), 'ydata',
                    DATA.path(2,1:DATA.i))
139         end
140
141         if dtsum==0 & ~isempty(z) % plots related to
                observations
142             set(h.xf, 'xdata', XX(4:2:end), 'ydata', XX(5:2:end)
                    )
143             plines= make_laser_lines (z,XX(1:3));
144             set(h.obs, 'xdata', plines(1,:), 'ydata', plines
                    (2,:))
145             pfcov= make_feature_covariance_ellipses(XX,PX);
146             set(h.fcov, 'xdata', pfcov(1,:), 'ydata', pfcov(2,:)
                    )
147         end
148     end
149     drawnow
150
151 end % end of main loop
152
153 if SWITCH_PROFILE, profile report, end
154
155 data = finalise_data(DATA);
156 set(h.pth, 'xdata', data.path(1,:), 'ydata', data.path(2,:))
157
158 clear global DATA
159 clear global XX
160 clear global PX
161
162 %
163 %
164
165 function h= setup_animations()
166 h.xt= patch(0,0,'b','erasemode','xor'); % vehicle true
167 h.xv= patch(0,0,'r','erasemode','xor'); % vehicle estimate
168 h.pth= plot(0,0,'k.','markersize',2,'erasemode','background'); %
```

```
    vehicle path estimate
h.obs= plot(0,0,'y','erasemode','xor'); % observations
h.xf= plot(0,0,'r+','erasemode','xor'); % estimated features
h.vcov= plot(0,0,'r','erasemode','xor'); % vehicle covariance
    ellipses
h.fcov= plot(0,0,'r','erasemode','xor'); % feature covariance
    ellipses

%
%

function p= make_laser_lines (rb,xv)
% compute set of line segments for laser range-bearing
    measurements
if isempty(rb), p=[]; return, end
len= size(rb,2);
lnes(1,:)= zeros(1,len)+ xv(1);
lnes(2,:)= zeros(1,len)+ xv(2);
lnes(3:4,:)= transformtoglobal([rb(1,:).*cos(rb(2,:)); rb(1,:).*
    sin(rb(2,:))], xv);
p= line_plot_conversion (lnes);

%
%

function p= make_vehicle_covariance_ellipse(x,P)
% compute ellipses for plotting vehicle covariances
N= 10;
inc= 2*pi/N;
phi= 0:inc:2*pi;
circ= 2*[cos(phi); sin(phi)];

p= make_ellipse(x(1:2), P(1:2,1:2), circ);

function p= make_feature_covariance_ellipses(x,P)
% compute ellipses for plotting feature covariances
N= 10;
inc= 2*pi/N;
phi= 0:inc:2*pi;
circ= 2*[cos(phi); sin(phi)];

lenx= length(x);
lenf= (lenx-3)/2;
p= zeros (2, lenf*(N+2));

ctr= 1;
for i=1:lenf
    ii= ctr:(ctr+N+1);
    jj= 2+2*i; jj= jj:jj+1;
```

```
213
214     p(:,ii)= make_ellipse(x(jj), P(jj,jj), circ);
215     ctr= ctr+N+2;
216 end
217
218 %
219 %
220
221 function p= make_ellipse(x,P,circ)
222 % make a single 2-D ellipse
223 r= sqrtm_2by2(P);
224 a= r*circ;
225 p(2,:)= [a(2,:)+x(2) NaN];
226 p(1,:)= [a(1,:)+x(1) NaN];
227
228 %
229 %
230
231 function data= initialise_store(x,P, xtrue)
232 % offline storage initialisation
233 data.i=1;
234 data.path= x;
235 data.true= xtrue;
236 data.state(1).x= x;
237 %data.state(1).P= P;
238 data.state(1).P= diag(P);
239
240 %
241 %
242
243 function store_data(x, P, xtrue)
244 % add current data to offline storage
245 global DATA
246 CHUNK= 5000;
247 len= size(DATA.path,2);
248 if DATA.i == len % grow array exponentially to amortise
       reallocation
249     if len < CHUNK, len= CHUNK; end
250     DATA.path= [DATA.path zeros(3,len)];
251     DATA.true= [DATA.true zeros(3,len)];
252     pack
253 end
254 i= DATA.i + 1;
255 DATA.i= i;
256 DATA.path(:,i)= x(1:3);
257 DATA.true(:,i)= xtrue;
258 DATA.state(i).x= x;
259 %DATA.state(i).P= P;
260 DATA.state(i).P= diag(P);
```

```
261
262 %
263 %
264
265 function data = finalise_data(data)
266 % offline storage finalisation
267 data.path= data.path(:,1:data.i);
268 data.true= data.true(:,1:data.i);
```

script calc_RMSE_lidar.m

```
1  %%CALCULATING MSE OF LIDAR EKF ALGORITHM
2  %Transducing the state vector into a matrix (2xN_landmarks)
3  state_lm=DATA.state(end).x(4:end);
4  est_lm=[];
5  for i=1:2:length(state_lm)
6      est_lm=[est_lm state_lm(i:i+1)];
7  end
8
9  %Calculating MSE related to the landmarks positions
10 num_MSE=0;
11 for i=1:length(est_lm)
12     new_dist_2=zeros(1,length(est_lm));
13     corr=zeros(1,length(est_lm));
14     for j=1:length(est_lm)
15         new_dist_2(j)=(lm(1,i)-est_lm(1,j))^2+(lm(2,i)-est_lm(2,
               j))^2;
16     end
17     corr(i)=find(new_dist_2==min(new_dist_2));
18     num_MSE=num_MSE+new_dist_2(corr(i));
19 end
20 MSE_lm=num_MSE/length(est_lm);
21
22 %Calculating MSE related to the robot positions
23 num_MSE=0;
24 xpos_true=DATA.true(1:2,:);
25 xpos_est =DATA.path(1:2,:);
26 diff=xpos_true-xpos_est;
27 MSE_pos=sum(sum(diff.^2))/length(diff);
28
29 RMSE_pos=sqrt(MSE_pos);
30 RMSE_lm=sqrt(MSE_lm);
31 db_RMSE_pos=[db_RMSE_pos RMSE_pos];
32 db_RMSE_lm=[db_RMSE_lm RMSE_lm];
```

script execute_ekf.m

```
close all
clear all
clc
load('LMWP.mat');

s=1;
ss=1;
for i=1:length(AP_lm)
    if s<3 && ss>0
        AP_lm(2,i)=-1+AP_lm(2,i);
        s=s+1;
    elseif ss>0
        s=-1;
        ss=-1;
    end
    if s>-3 && ss<0
        AP_lm(2,i)=1+AP_lm(2,i);
        s=s-1;
    elseif ss<0
        s=1;
        ss=1;
    end
end

db_rmse_pos=[];
db_rmse_lm=[];
PERM_ANG_DIST=1
if PERM_ANG_DIST==0
    std_err_metres=[0.01,0.1,0.5,1,1.5,2];
    std_err_angle=5;
else
    std_err_metres=0.5;
    std_err_angle=[5,10,20,30,45,60];
end

if PERM_ANG_DIST==0
    for ee=1:length(std_err_metres)
        [RMSE_pos,RMSE_lm,data]= ekfslam_sim(AP_lm, wp,WIDTh,
           LENGTH,CORR,std_err_metres(ee),std_err_angle);
        db_rmse_pos=[db_rmse_pos;RMSE_pos]
        db_rmse_lm=[db_rmse_lm;RMSE_lm]
    end
else
    for ee=1:length(std_err_angle)
        [RMSE_pos,RMSE_lm,data]= ekfslam_sim(AP_lm, wp,WIDTh,
```

```
            LENGTH,CORR,std_err_metres,std_err_angle(ee));
45          db_rmse_pos=[db_rmse_pos;RMSE_pos]
46          db_rmse_lm=[db_rmse_lm;RMSE_lm]
47      end
48  end
49  tabless
50  plotgraphics_lidar
```

List of Tables

List of Figures

Bibliography

[1] Davide Dardari, *Bayesian filtering for distributed estimation and control lectures*, University of Bologna, 2018

[2] David M. Rosen, Julian Mason and John J. Leonard, *Towards Lifelong Feature-Based Mapping in Semi-Static Environments*, 2016 IEEE International Conference on Robotics and Automation (ICRA) Stockholm, Sweden, May 16-21, 2016 , Massachusetts, 2nd edition, 1994.

[3] Kai M. Wurm, Cyrill Stachniss, Giorgio Grisetti, *Bridging the gap between feature- and grid-based SLAM*,Robotics and Autonomous Systems journal homepage: www.elsevier.com/locate/robot, 22nd September 2009

[4] Zhang Heng, *TOA based Localization and Tracking in Indoor Multipath Environment*, School of Electrical and Electronic Engineering, doctoral thesis, 2018

[5] Hugh Durrant-Whyte,Tim Bailey, *Simultaneous Localization And Mapping: Part I*, IEEE Robotics & Automatics Magazine, June 2006

[6] Hugh Durrant-Whyte,Tim Bailey, *Simultaneous Localization And Mapping: Part II*, IEEE Robotics & Automatics Magazine, June 2006

[7] Simo Ssarka, *Bayesian Filtering and Smoothing*, Cambridge University Press, 2013

[8] D. Dardari, P. Closas, P.M. Djuric , *Indoor Tracking: Theory, Methods and Technologies*, IEEE Transactions on Vehicular Technologies, Vol. 64, no.4, pp. 1263-1278, April 2015

[9] Tim Bailey, *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments*, Doctor of Philosophy thesis, Australian Centre for Field Robotics, Department of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, 2002

[10] C. Cadena and L. Carlone and H. Carrillo and Y. Latif and D. Scaramuzza and J. Neira and I. Reid and J.J. Leonard, *Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age*, in IEEE Transactions on Robotics 32 (6) pp 1309-1332, 2016

[11] Patric Jensfelt, *Approaches to Mobile Robot Localization in Indoor Environments*, Doctoral thesis, Department of Signals, Sensors and Systems, Royal Institute of Technology, Stockholm, Sweden, 2001

[12] Khalid Yousif, Alireza Bab-Hadiashar, Reza Hoseinnezhad, *An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics Intelligent Industrial Systems*, 2015, Volume 1, Number 4, Page 289

[13] Juan D. Tardós, José Neira, Paul M. Newman and John J. Leonard, *Robust Mapping and Localization in Indoor Environments Using Sonar Data*, The International Journal of Robotics Research, 2002

[14] Angelos Mallios, *Sonar scan matching for simultaneous localization and mapping in confined underwater environments*, Doctoral thesis, Universitat de Girona, 2014

[15] Tobias Deißler1, Malgorzata Janson2, Rudolf Zetik, Jörn Thielecke, *Infrastructureless Indoor Mapping Using A Mobile Antenna Array*, 19th International Conference on Systems, Signals and Image Processing (IWSSIP), 2012

[16] Ebi Jose, Martin D. Adams, *Relative RADAR Cross Section based Feature Identification with Millimetre Wave RADAR for Outdoor SLAM*, School of Electrical and Electronic Engineering, Nanyang Technological University, January 2004

[17] Tobias Deißler, Jörn Thielecke, *UWB SLAM with Rao-Blackwellized Monte Carlo Data Association*, 2010 International Conference on Indoor Positioning and Indoor Navigation (IPIN), 15-17 September 2010, Zürich, Switzerland

[18] Anna Guerra, Francesco Guidi, and Davide Dardari, *Millimeter-Wave Personal Radars for 3D Environment Mapping*, IEEE Asilomar Conference on Signals, Systems, and Computers, At Pacific Grove, USA, November 2014

[19] Erik Leitinger, Florian Meyer, Franz Hlawatsch, Klaus Witrisal, Fredrik Tufvesson, Moe Z.Win, *A Belief Propagation Algorithm for Multipath-Based SLAM*, Graz University of Technology, January 2018

[20] Joan Sola,*An EKF-SLAM toolbox in Matlab*, LAAS-CNRS, December 2013