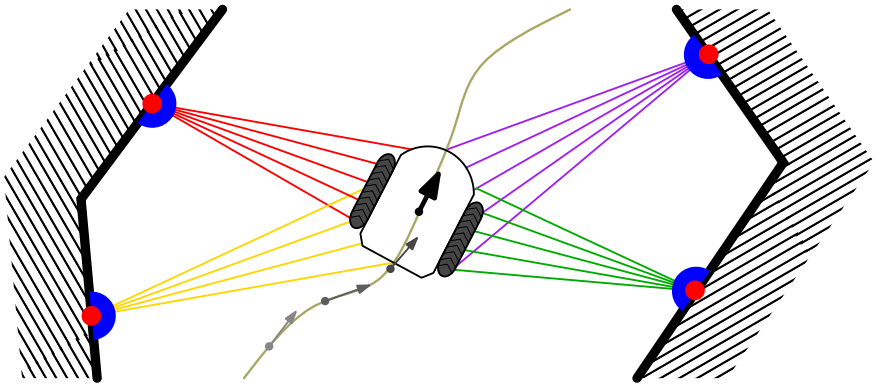
INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa



Sensor Network and Localization methods for a mobile robot

João Filipe Teles Ferreira

Dissertação para obtenção do grau de mestre em

Engenharia Electrotécnica e de Computadores

Júri

Presidente:	Doutor Carlos Jorge Ferreira Silvestre
Orientadores:	Doutor Rodrigo Martins de Matos Ventura
	Doutor Alberto Manuel Martinho Vale
Vogal:	Doutor Jorge Manuel Miranda Dias

Outubro de 2010

Resumo

Nesta tese propõe-se um sistema de localização para veículos móveis, sem sensores a bordo, no interior de edifícios. A parte sensorial consiste numa rede de scanners laser instalados em determinados pontos do edifício. Um sistema deste tipo pode ser necessário quando as características do veículo, ou da sua função, não são apropriadas para a instalação de sensores directamente na sua estrutura.

Para a parte sensorial foi desenvolvido um método para optimizar o posicionamento dos sensores no edifício. Melhorando a disposição da rede sensorial melhora-se a qualidade da informação recolhida. Tirando partido desta informação, dois métodos de localização foram testados, um baseado no filtro de Kalman estendido, o outro no filtro de particulas. Ambos foram estudados e comparados, em ambiente de simulação, quanto ao seu desempenho e fiabilidade. Foi analizado o impacto da configuração da rede sensorial no sistema, avaliando assim a robustez a diferentes disposições dos sensores e eventuais avarias em alguns deles.

Dos ensaios realizados, o filtro de particulas é a aproximação mais fiável para este tipo de aplicação, a estimação da posição do veículo é feita com uma boa precisão, é robusto a falhas de sensores e, depois de falhas generalizadas, como por exemplo uma falha de energia, localiza globalmente o veículo num curto período de tempo. Os resultados, embora simulados, são bastante promissores para uma futura aplicação numa plataforma real.

Palavras-chave: Localização, Veiculos móveis autónomos, Rede de Sensores, Scanners laser, Fusão de dados.

Abstract

This thesis propose a localization system for mobile vehicles, without sensors on board, in indoor environments. The sensory part is a network composed by laser scanner sensors placed in defined points on the building. This system is useful in cases where sensor installation directly on the vehicle is not advised, due to his or his job characteristics.
An optimization method was developed to place the sensors on the building. Improving the network disposition on the building, the quality of gathered information increases. Using this information, two localization methods were developed and tested, one based on extended Kalman filter, and other on particle filter. Both were studied and compared, in simulation environment, concerning their performance and reliability. It was analyzed the impact of sensors disposition on the building in the localization system, evaluating the robustness to different sensor configurations, including a possible fail of some of them.
From the experimental results, the approach that suits better this application is the particle filter localization method, the resulting estimations have good precision, it is robust to sensor failure and, facing a general failure, like power failure, it manages a short time global localization. Although these conclusions are based on simulation, the method shows promising results for a future real application.

Keywords: Localization, Autonomous Mobile Vehicles, Sensor network, Laser scanners, Data Fusion.

Contents

List of Tables

List of Figures

List of Abbreviations

1D	—	1 Dimension
2D	—	2 Dimensions
AGV	—	Automated Guided Vehicle
CCW	—	Counter Clock Wise
CG	—	Coverage Graph
CW	—	Clock Wise
EKF	—	Extended Kalman Filter
FoV	—	Field of View
GA	—	Genetic Algorithm
ITER	—	International Thermonuclear Experimental Reactor
KF	—	Kalman Filter
LRF	—	Laser Range Finder
MC	—	Monte Carlo
MCL	—	Monte Carlo Localization
PF	—	Particle Filter
SA	—	Simulated Annealing
TB	—	Tokamak Building
TCS	—	Transfer Cask System
VP	—	Visibility Polygon
VV	—	Vacuum Vessel
WHB	—	WareHouse Building

Chapter 1

Introduction

1.1 Motivation

One of the biggest technological challenges in the present is the production of clean energy, with small environmental impact but still supplying the ever growing consumption. Atomic fusion has potential to be a suitable alternative, it has the capability to generate abundant energy, releasing no carbon dioxide or greenhouse gases.
International Thermonuclear Experimental Reactor (ITER) is an experimental fusion reactor, used to test this process, and prove if this kind of energy generation is practicable. It is an investigation work shared by many countries, E.U.A., Russia, E.U., China, Japan, India and Republic of Korea, representing, together, more than half the world's population. ITER facilities will be built in Cadarache, Southern France, and are expected to be ready in 2019.
ITER operation is based on an central component, the tokamak, that generates energy using nuclear fusion of Deuterium and Tritium atoms, two isotopes of Hydrogen, this reaction reaches very high temperatures, forming a plasma. This plasma is kept in a Vacuum Vessel (VV) away from walls trough a very powerful magnetic field. The heat generated by the fusion reaction is used to generate electrical energy. The Tokamak must be protected, the Blanket is a layer that covers all the inside walls of VV providing protection from heat and radiation.
During ITER operation, maintenance actions, like inspections or component replacement, are necessary, and can only be made remotely, due to rad-hard conditions. The transportation of robotic operators and spare parts to the maintenance site is required and is a crucial task on tokamak maintenance.All this equipment is transported inside a shielded cask by a remote controlled vehicle, the Transfer Cask System (TCS) through narrow corridors. This is a very hard task because the operator cannot be present and can only see the vehicle through a video surveillance system. A more accurate method to drive TCS is necessary due to the tight safety margins inside the building.
The proposed solution is a localization system that gives a precise estimation on the vehicle position and orientation. The estimation can be used either, to help a human, operating remotely, or to implement an autonomous guiding system.
A specific propriety of the vehicle task, the radiation load, gives an unique specification for

the localization system, vehicle must be kept without sensors. The main radiation source is the vehicle load, so it is not advised to install sensors in the vehicle because the radiation would rapidly damage his electronic hardware.
The solution is to strategically install sensors on the building, where the radiation has lower levels due to building shield and, since the sensor is static, is easier to shield sensor parts to reduce the exposure.
The sensors selected to observe the entire operation area are Laser Range Finder (LRF) sensors, these are precise and accurate from short to long ranges, the measures have no interference from the magnetic fields on the tokamak and sensor's electronic part exposure to radiation can be minimized, leaving only the mirror exposed, which is suited to the proposed problem. Various LRF sensors are required to avoid not covered areas, occluded by scenario obstacles. The main problem to solve is to estimate the real vehicle pose based on measurements acquired by the distributed sensor network.
Remote handling systems in ITER are crucial to the entire system performance, they will influence directly the maintenance tasks duration, and with more maintenance time comes less energy production time, rising the cost of each maintenance intervention. A well designed remote handling system is required to minimizing these costs maximizing the production time.
Although the motivation is the localization in ITER scenarios, this localization system is suitable to other indoor scenarios, providing an accurate localization to autonomous vehicles.

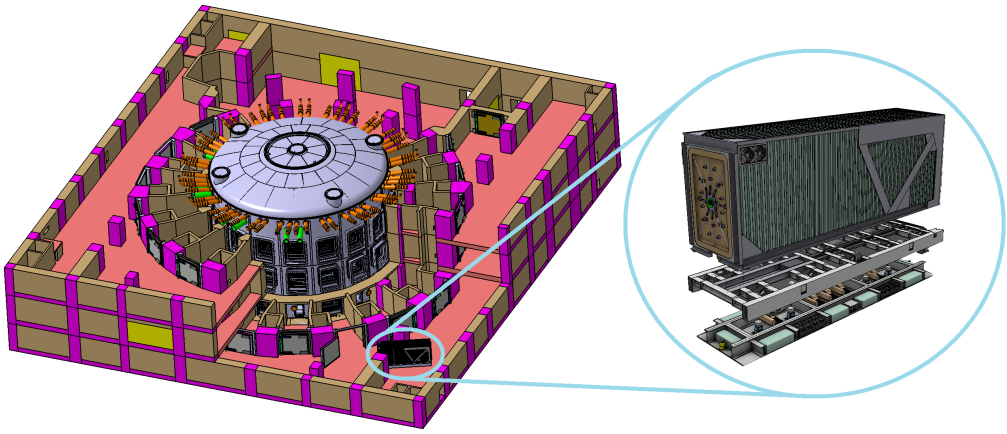


Figure 1.1: ITER Tokamak building and Transfer Cask System

1.2 Scientific Context

Localization problem is well known in robotic field due to his importance in task execution, in particular, for guidance of an autonomous vehicle.
Methods for localizing moving vehicles attracts researchers since the beginning of robotics, with the efforts applied in this area, together with sensor and processing evolutions, these methods are becoming very accurate and powerful.
Depending on the application and resources, many localization techniques are available, the Extended Kalman Filter (EKF) is often used with this propose, despite his limitations to gaussian models it has very good results in a great variety of problem, with few computational effort, [2], Monte Carlo (MC) methods such as Particle Filters (PF) ([9], [15]), are becoming the new leading techniques, with evolution in processors the computational weight of particle filters are less concerning facing the major benefices, as the non-gaussian, non-linear models supported. Grid based localization methods,[29] are not suited for this problem since they are based on discretization of the state space, to have some accuracy, require very large memories.
General applications of these methods depends on sensors mounted on board of the vehicle, the main difference to our solution is the constraint of leaving the vehicle completely sensor free. The problem shifts completely with this imposed change, normally the vehicle observes the surrounding environment, in this problem, a distributed sensor network observes the vehicle and the surroundings simultaneously. Some localization systems with this principle have been implemented, with a sensory network estimating the correct pose of a moving element, typically resorting to vision sensors.
With the evolution in the vision field there are some investigation on performing localization with a set of cameras [21], there are also current research on estimating position with other types of equipment, for instance using WiFi [3] or RFID technologies [4].
The above techniques are not suited for the present problem, the requirements dictate that an external sensor network is needed, and observing the vehicle, the camera network, WiFi or RFID implementations are not very reliable and have low accuracy.
The sensors that will be used on this work are state of the art Laser Range Finder (LRF), these sensors can measure several distances to obstacles with great accuracy at long ranges, with numerical output, easy to process than images from cameras, for instance. Other advantage is the possibility to expose only the mirror to radiation, increasing the lifetime of sensor electronic parts. These kind of sensors are applied in obstacle avoidance in industry and on localization methods as well ([31], [36]), with sensors mounted on the vehicle. But, for this approach, it is required a network of laser sensors mounted on the building.
The LRF network is a crucial part of the localization system, the layout of such network will play a major role on the system's performance, so, each sensor placement on the building should be optimized according some criteria. To optimize these placements, a very large and complex set of possible sensor positions is analyzed, requiring MC optimization methods to do so, such as Genetic Algorithms (GA) or Simulated Annealing (SA) [37]. Optimization techniques are constantly evolving as the quest for better and faster algorithms. The selection for SA, in this approach, is based on algorithm simplicity, and capability of deliver

good and fast results when compared to more complex algorithms.

1.3 Thesis Objectives

To achieve the main objective of this thesis, the development of a localization system for mobile vehicles without sensors, is required the design of a distributed sensory network that observes the vehicle at all time and an algorithm to estimate vehicle pose from data retrieved by the sensors. For network definition an algorithm for optimizing the coverage areas of sensor network is implemented, Chapter 2 focus on this optimization process, discussing evaluation criteria and optimization algorithm and results.
The main objective is the development of a robust localization system, with acceptable error margins and high performance, capable of localizing a moving vehicle with no sensors on board. Chapter 3 addresses the development and comparison of two localization algorithms to choose the better suited for this kind of implementation. The effects of sensor placement in localization performance are studied on Chapter 4, where the importance of optimization for the overall system is evaluated.

Up to date a similar work on localization with laser range finder laser network mounted on the building, with a mobile vehicle completely free of sensors, was not found, only similar approaches with cameras [21].
It is an innovative technique that allows a localization with sensor free robot, with further work may be suited to multiple vehicle localization as well, with no need for more sensors. This have huge applications on industry and logistics as the common Autonomous Guided Vehicle (AGV).With this technique, AGV have no longer a restricted trajectory, with a stripe or cable on the floor, trajectories can be changed only in virtual space keeping exactly the same sensor network and the same vehicles.
This system is also very robust to electromagnetic noise as it work with lasers, exterior perturbations have little effect on the system to, as the sensor can be shielded almost entirely leaving only the mirror exposed, this is an important feature due to major electromagnetic fields inside ITER.

1.4 Thesis Outline

Thesis is divided into three parts, the dimensioning and optimization of sensory network, the sensor data processing stage, that actually localizes the vehicle given the measurements retrieved by the sensors and the evaluation of the effects that sensor placement have on localization performances. Chapter 2 focus with the optimization of sensor placement, some criteria are discussed and the most important one, the coverage, is optimized, generating several networks for different scenarios. Simulated Annealing algorithm is explained and used to optimize sensor networks and some experimental results are shown.
Vehicle Localization is addressed in Chapter 3, using a previous dimensioned network, two methods for localization are tested, EKF and a PF, some adjusts and heuristics are developed, to achieve better performances, and the global system performance is evaluated through experimental results obtain in simulation.
Chapter 4 shows some results and comparisons between different network performance on localization, what is the effect of a non-optimal network in the localization output, variations with the number of sensors. In this chapter the importance of optimization criteria will be evaluated, based on resulting performances with different sensor networks. Conclusions and future work proposed are exposed in Chapter 5 it has a final review of the work done, his applicability on real world situations. Future work and open issues are left open for further investigation trying to increase system's capabilities and robustness.

Chapter 2

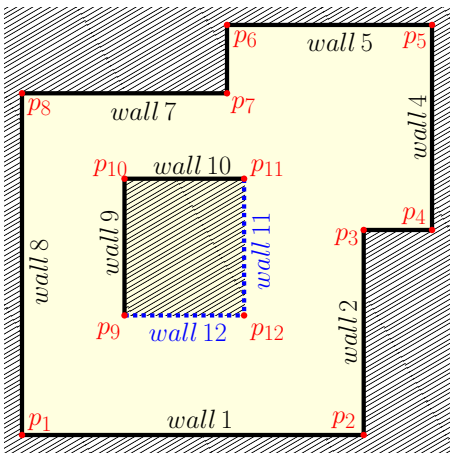
Sensor Placement Optimization

2.1 Problem Statement

Guiding autonomous vehicles to navigate to a given goal position requires a well designed positioning system. Inside a building with narrow spaces, maneuvering a vehicle is a hard problem, so the pose (position and orientation) estimation must be accurate and precise.
In general, vehicle estimates his own pose using on board sensors. The proposed solution uses building installed sensors, while the vehicle has none.
To implement this location system, it is crucial that the sensor network covers the entire scenario, avoiding occlusion situations. Indoor environments have, typically, many obstacles creating occlusions, only one sensor can never cover the entire area. To have a complete coverage of the environment is required a group of sensors with different configurations. Optimizing these configurations is crucial to the systems performance.
This Chapter proposes a method for optimizing sensors' placements, maximizing the coverage, with the goal of minimizing the amount of sensors to be installed.
The sensor coverage in an indoor environment is similar to the art gallery problem stated by [7]. This problem inquires how many observation points are necessary to cover an entire area with a given number of walls. This is a NP-problem as proved by [1]. Any solution to these problems is very fast to verify but there is no fast solution known. The difficulties in optimization on a visibility problem like this are known, and some approximate solutions have been proposed, applied usually in visual sensors [33], like cameras in surveillance systems like [10]. There are developed methods to optimize coverage areas and enhance performance of visual sensors network ([23], [24]). The improvement of visual sensor network coverage, like surveillance cameras [39] have similar optimizing difficulties, the main constraints are the same, the occlusions in indoor environments. A preliminary study on LRF network coverage is addressed in [11], it describes the optimization framework presented in this thesis.
To optimize the sensor placement is required a map of the scenario. The map is a simplified 2D layout since, usually, the vehicle operates on floor level. This way the map of the scenario consists of 2D cartesian points MP, defining the corners and a set of edges connecting these

points ML (2.1).

$$MP = \begin{bmatrix} p_1 \\ \vdots \\ p_N \end{bmatrix} = \begin{bmatrix} p_{1x} & p_{1y} \\ \vdots & \vdots \\ p_{Nx} & p_{Ny} \end{bmatrix}, ML = \begin{bmatrix} i_1 & f_1 & v_1 \\ \vdots & \vdots & \vdots \\ i_M & f_M & v_M \end{bmatrix} \tag{2.1}$$



	MP	ML		
j		i_j	f_j	v_j
1	p_1	1	2	T
2	p_2	2	3	T
3	p_3	3	4	T
4	p_4	4	5	T
5	p_5	5	6	T
6	p_6	6	7	T
7	p_7	7	8	T
8	p_8	8	1	T
9	p_9	9	10	T
10	p_{10}	10	11	T
11	p_{11}	11	12	F
12	p_{12}	12	9	F

Figure 2.1: Example of map representation

The map is built from a set of N points MP, and M lines, ML. Each point is represented by his coordinates p_x and p_y. ML is the set of walls on the map, each one with an initial and a final point, i and f are the indexes of those points in the set MP, respectively. v is a flag stating if the sensor can be installed on that wall or not. Figure 2.1 presents a simple map with the respective definition, including points, walls and flags. Map is basically a polygon with holes; the exterior polygon is a sequence of points in counter clock wise (CCW) it represents the outer walls and corners. Holes represent obstacles, smaller polygons inside the exterior one. Each one is a sequence of point in clock wise (CW).

The sensor state $\hat{s}$ (2.2), is composed by $\lambda \in [0, 1]$, a parameter of sensor position along the wall, θ_s, the sensor orientation, β, the wall where the sensor is installed and Φ_s, the sensors Field of View. With $\lambda = 0$ the sensor is installed in the initial point of wall β, as λ grows sensor moves along the wall reaching $\lambda = 1$ in the final point. This coordinate system was defined as wall coordinate system, as it references the sensor position with respect to walls on the map. On the approach followed in this work, the LRF sensor is always installed on walls, being these the static parts of the environment, so the definition of this particular coordinate system.

$$\hat{s} = \begin{bmatrix} \lambda & \beta & \theta_s & \Phi_s \end{bmatrix}^T \tag{2.2}$$

A transformation between wall coordinates and cartesian coordinates, defined as $\mathcal{T}(\hat{s})$ (2.3) that generates a new state, $\hat{s}_{cart}$ in cartesian coordinates that is useful to export the sensor

placement resulting from optimization.

$$\hat{s}_{cart} = \mathcal{T}(\hat{s}) = \begin{bmatrix} \hat{x_s} \\ \hat{y_s} \\ \theta_s \\ \Phi_s \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} p_{i_\beta} \end{bmatrix}^T \\ \theta_s \\ \Phi_s \end{bmatrix} + \lambda \begin{bmatrix} \begin{bmatrix} p_{f_\beta} - p_{i_\beta} \end{bmatrix}^T \\ 0 \\ 0 \end{bmatrix} \tag{2.3}$$

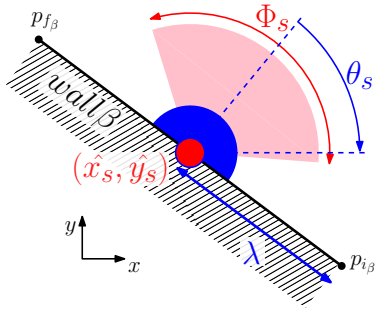


Figure 2.2: Sensor state representation

Including multiple sensors a tuple of sensor states $\hat{S}_L$ or $\hat{S}_{Lcart}$ is compiled (2.4 and 2.5) including a state for each sensor. L is the total number of sensors used by the sensory network and it is defined prior to the optimization step. The variables to optimize are only the sensors' placements. For each L there's an optimal sensory network configuration respecting given criteria, shown in section 2.2.

$$\hat{S}_L = \left\{ \ \hat{s}_1, \quad \dots \quad , \hat{s}_L \ \right\} \tag{2.4}$$

$$\hat{S}_{L\,cart} = \left\{ \ \mathcal{T}(\hat{s}_1), \quad \cdots \quad , \mathcal{T}(\hat{s}_L) \ \right\} \tag{2.5}$$

The areas where a sensor can be installed on the map must be predefined before the optimization, they represent optimization constraints that should not be neglected. A Sensor State Space $\widehat{SSS}$ defines the possible configurations for sensors. During optimization, each sensor state, $\hat{s}$, belongs to $\widehat{SSS}$.

The $\widehat{SSS}$ is defined as a path traveling trough all map walls with exception for walls marked with the flag $v = False$ represented with dashed lines on Figure 2.1. Some lines on the map could not be representing solid walls, and could not be prepared for sensor installation, they could be representing removable obstacles, doors, etc...

Given the framework of the problem, the optimization must compute the optimal placement for L sensors inside a given map $\{MP, ML\}$ with his respective restrictions.

2.2 Optimization Criteria

To optimize the sensor placement on a given map, we should define a criteria that distinguish the best solution from the others. Normally a evaluation function where the best solution corresponds to an global extreme. Defining these criteria and evaluation function is not trivial and should consider the actual objectives and application of the problem that is being optimized.
The objective of our optimization problem is to find the best placements for LRF sensors inside an indoor scenario. The application of these sensor network is to localize a vehicle traveling inside that same scenario and guarantee and accurate and precise localization prediction for all vehicle's possible positions.

2.2.1 Coverage

The most important criteria seems to be the coverage that the sensor network have on the map. Since the localization system must be valid at all possible positions, it is logic that it must cover the entire scenario or else,when the vehicle travels to an hiding zone, with no sensor readings,there is no way to localize it. This is a very dangerous situation due to the risk of collisions between the vehicle and the scenario. If the system lose track of the vehicle it can no longer control it correctly, keeping it in the correct trajectory.
We establish that the percentage of the total map area seen from the sensor network was the best way to measure coverage. The optimization process consists in the maximization of this percentage having the sensors positions has variables.

Single Sensor Coverage

Starting with a single sensor network ($L = 1$), we must define a sensor state space ($\widehat{SSS}$) to evaluate coverage. $\widehat{SSS}$ is the sensor path that contemplates the entire map wall space, with exception to the ones marked with the flag $v = False$ on ML array, dashed lines on Figure 2.3. The path direction on the exterior map walls is CCW, while in interior walls is CW. λ parameterizes translation on the path and it always grows from 0 to 1. Rotation is also contemplated by the path since the majority of LRF lasers in the market have a FoV equal or greater that 180° ($\Phi \geq 180°$). $\widehat{SSS}$ path is composed by two major segments, the translation along the walls and the rotation in corner points as shown in Figure 2.3. To evaluate coverage globally, the sensor state ($\hat{s}$) follows the path $\widehat{SSS}$, which guaranties that it visits all possible states.

$$Poly(\hat{s}) = Poly(\hat{s}, MP, ML) = \begin{bmatrix} a_1 & a_2 & \cdots & a_K \end{bmatrix}^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_K \\ y_1 & y_2 & \cdots & y_K \end{bmatrix}^T \quad (2.6)$$

The best way to measure the coverage of a sensor is building a polygon, the visibility polygon (VP), from the geometry of the problem. This defines the area covered by a sensor, respecting occlusions imposed by environment obstacles. A function $Poly(\hat{s}, MP, ML)$ (2.6) is defined to build VP from sensor state, $\hat{s}$ and map layout $\{MP, ML\}$. For notation simplicity the map parameters $\{MP, ML\}$ can be omitted. It returns an array defining the

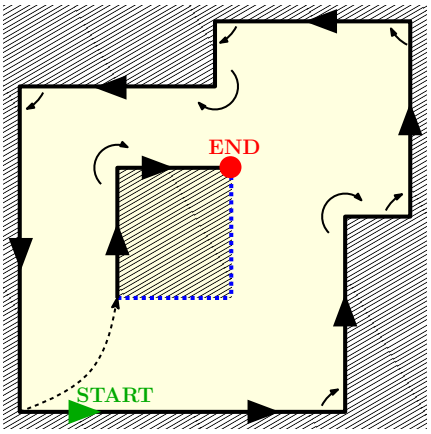


Figure 2.3: Sensor state space example

vertexes of VP in CW (Figure 2.4). Visibility polygons are always star-shaped as defined in [26], meaning that the entire polygon can be seen from at least one point inside itself, having no holes.

Function $Poly(\hat{s}, MP, ML)$ is implemented through an computational algorithm based on simple geometry and algebra. This algorithm as two major function inside:

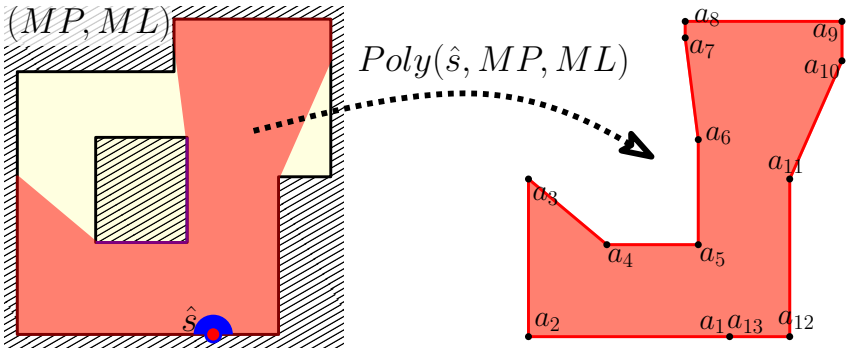
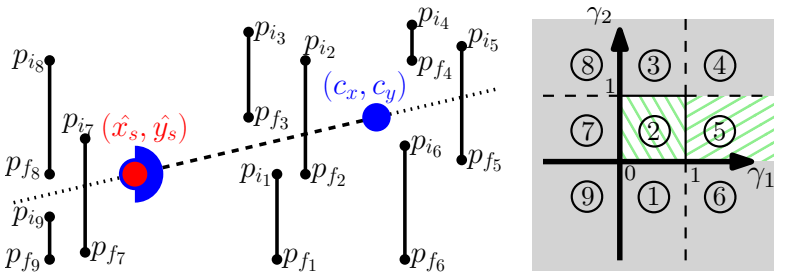


Figure 2.4: Visibility Polygon extraction

$Intersect(\hat{s}, c, w, MP, ML)$ - This function is the core of the algorithm, it makes the intersection between two line segments, one going from sensor position, defined in $\hat{s}$, to a map point defined in MP, called in this context as corner (c), and other segment defined in ML as a map wall (w). In the end it returns two values that parameterize the intersection (2.7). Evaluating the combination between these two values it is possible to know the point of intersection n (2.8) and evaluate the situation as described in Table 2.1. Figure 2.5(a) presents some possible situation with different map walls, while Figure 2.5(b) parameterizes those situation to (γ_1,γ_2) axes. From Table 2.1 it can be concluded that the areas of interest are in stripes, parameterizing, as example, walls 2 and 5. The other areas are irrelevant to the algorithm implementation as they represent walls that do not cross the line of sight between sensor and the point c.

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \end{bmatrix} = \begin{bmatrix} c_x - \hat{x_s} & p_{f_w x} - p_{i_w x} \\ c_y - \hat{y_s} & p_{f_w y} - p_{i_w y} \end{bmatrix}^{-1} \begin{bmatrix} p_{i_w x} - \hat{x_s} \\ p_{i_w y} - \hat{y_s} \end{bmatrix} \tag{2.7}$$

$$\begin{bmatrix} n_x \\ n_y \end{bmatrix} = \gamma_1 \begin{bmatrix} c_x - \hat{x_s} \\ c_y - \hat{y_s} \end{bmatrix} + \begin{bmatrix} \hat{x_s} \\ \hat{y_s} \end{bmatrix} \tag{2.8}$$



(a) *Intersect*() with several walls

(b) Solution areas on (γ_1,γ_2) axes

Figure 2.5: *Intersect*() typical situations

Table 2.1: *Intersect*() typical situations description

Walls (w)	**(γ_1,γ_2)**	**Description**
1,6	$\gamma_2 < 0$	Walls are outside the line of sight
2	$0 \leq \gamma_1 \leq 1$ $0 \leq \gamma_2 \leq 1$	Wall is in the line of sight blocking the visibility between the sensor and corner (c)
5	$\gamma_1 > 1$ $0 \leq \gamma_2 \leq 1$	Wall is in the line of sight behind corner (c)
3,4	$\gamma_2 > 1$	Walls are outside the line of sight
7,8,9	$\gamma_1 < 0$	Walls behind the sensor, outside the line of sight

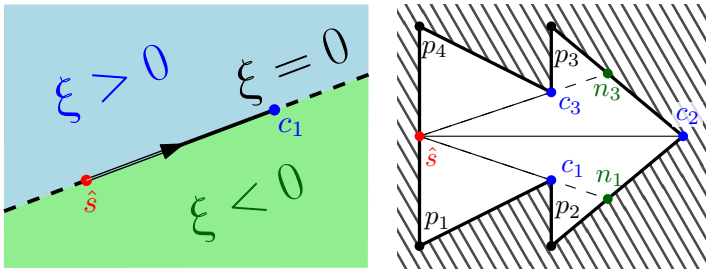
With these two values, points of the map are classified whether they should enter to VP array or not, or even if some new points should be added.

Side($\hat{s}, c, MP, ML$) - Side function is used to classify the corner (c), depending on this classification the results from *Intersect*() have different outcomes as described in Table 2.2. This classifier has also impact on the order the points are added to VP array. Figure 2.6(b) presents example of possible classifications, and match them to Table 2.2. Cross product is the main operation on this function, it tells where are the points with respect to the line of sight (2.6(a)), if they are on the right, middle or left (2.9). On the current map construction, we assume each point to be part of two walls, so it is connected directly to other two points (neighbors). These points are used to compute *Side*() that will pay a important role defining which points to include in VP and in which order.

Some situations may have $\xi = 0$ for one of the neighbors, in this case the *Side*() value

is equal the sign of the other neighbor's ξ.

$$\xi = \begin{vmatrix} c_x - \hat{x_s} & c_y - \hat{y_s} \\ p_{jx} - \hat{x_s} & p_{jy} - \hat{y_s} \end{vmatrix}, \quad \begin{cases} right, & \xi < 0 \\ middle, & \xi = 0 \\ left, & \xi > 0 \end{cases} \tag{2.9}$$



(a) *Side*() cross product (b) *Side*() example situations

Figure 2.6: *Side*() typical situations

Table 2.2: *Side*()typical situation description

Corner	**Connect**	ξ	*Side*()	**Description**
c_1	p_1	< 0	-1	Can see n_1 behind corner. Put n_1 in VP.
	p_2	< 0		First n_1 and then c_1 to respect CW order.
c_2	p_2	< 0	0	Can not see behind corner..
	p_3	> 0		Include only c_2 in VP.
c_3	p_3	> 0	1	Can see n_3 behind corner. Put n_3 in VP.
	p_4	> 0		First c_3 and then n_3 to respect CW order.

Using these functions, the algorithm to extract VP is described in Table 2.3. It returns a polygon with vertexes perfectly align with the map points, because it is extracted directly from map coordinates and not by methods of discretization, like ray-cast from the sensor position. Although much simpler, this method introduces error of discretization.
An example of the implemented algorithm is shown in Figure 2.7. It begins by adding the points defining the wall where the sensor is installed, and the sensor itself (a_{12},a_1, a_2), next it chooses c_1 as corner because it is the first in CW ordering,discovering a point behind the corner (n_1). n_1 and c_1 are added to the array by this order because $Side(\hat{s}, c_1, MP, ML) = -1$. Next it evaluates corner c_2, this is not added because it is occluded with a map wall, the same applies to c_3, c_4 is added next, alone, because it is not occluded by any wall but $Side(\hat{s}, c_4, MP, ML) = 0$, no points behind him can not be seen. Same strategy is applied to all corners in the map by CW order being the algorithm result shown in 2.10. This is only a simplified example to understand the algorithm.

$$Poly(\hat{s}, MP, ML) = \begin{bmatrix} a_1 & a_2 & n_1 & c_1 & c_4 & c_5 & n_5 & c_6 & c_7 & n_8 & c_8 & a_{12} & a_1 \end{bmatrix}^T \tag{2.10}$$

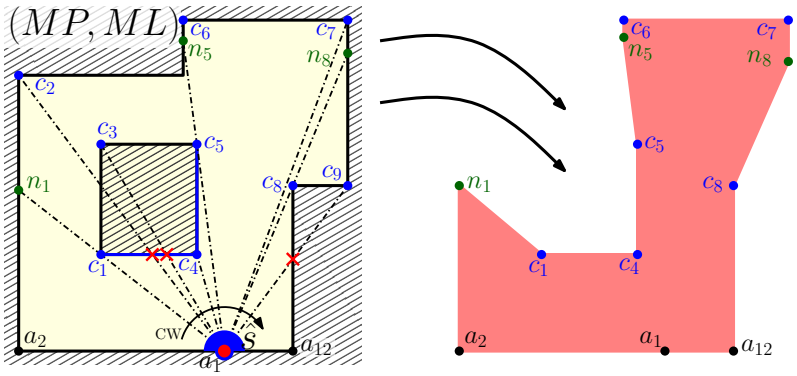


Figure 2.7: Optimization Algorithm example

This algorithm can be applied to any sensor state defined in $\widehat{SSS}$, for each one, there is an associated visibility polygon (VP) and his respective coverage area. This area ($C(\hat{s})$) is computed using VP vertexes (2.11). As the sensor state changes along $\widehat{SSS}$ the coverage is shown in a Coverage Graph (CG). CG is shown in percentage of total area, as the actual building's dimensions are not important for this evaluation.
To compute CG, $\widehat{SSS}$ must be discretized, since the visibility polygon computation is a slow operation, and should be performed off-line to accelerate the processes afterwards. This discrete set of sensor configurations is called $d\widehat{SSS}$, for each distinct state on this set, there is an index (Tag) that identifies it. Tag introduces a sense of sequence of sensor states, this is the same sequence followed by $\hat{s}$ on path $\widehat{SSS}$. CG is a graph showing the area covered by sensor depending on Tag and includes two types of segments, translation segments if the difference to next state is just a translation of sensor position, and rotation segments if the difference is in orientation, on corners. Figure 2.8 is the CG for the map example of Figure 2.3.

$$C(\hat{s}) = Area(Poly(\hat{s})) = \frac{1}{2}\sum_{i=0}^{K-1}(x_i y_{i+1} - x_{i+1} y_i) \tag{2.11}$$

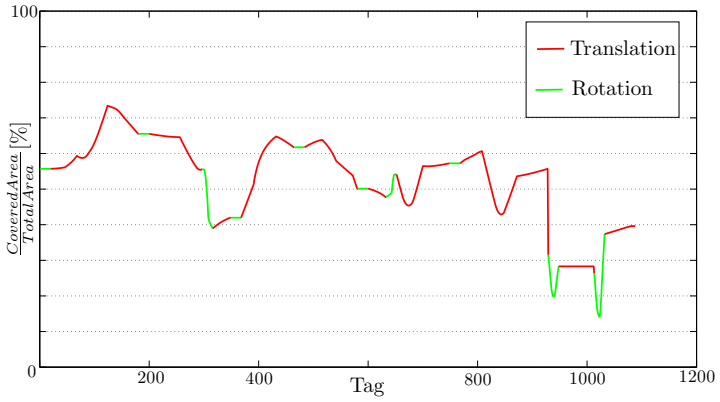


Figure 2.8: CG example

Table 2.3: $Poly(\hat{s}, MP, ML)$ Algorithm

1. Organizes MP points CW with respect to sensor position ($\hat{x_s}$,$\hat{y_s}$);
2. Excludes map points outside the FoV;
3. Initialize an empty array $Poly(\hat{s}$,MP,ML);
4. Include, the polygon limit vertexes, limited by FoV, in the array;
5. For each map point (c):
   (a) Calculates $Side(\hat{s}, c, MP, ML)$;
   (b) For each map wall (w):
      i. Calculates $Intersect(\hat{s}, c, w, MP, ML)$ between segment (from sensor to corner) and map wall (w);
      ii. If the wall occludes the corner, break cycle, if not, registers the closest intersection (n);
   (c) If the c is on a side, ($Side() = 1 \vee -1$) the corner(c) and the closest intersection (n) enter the array in CW order as described in Table 2.2, if the point is in the middle ($Side() = 0$), only himself (c) enters the array;
6. Export the array.

Multi Sensor Coverage

Due to occlusions one sensor is not enough to cover the entire area. To overcome this problem there is the need to build a sensor network formed with multiple sensors with different configurations. This network is described by a tuple[1], $\hat{S}_L$ (2.4), it places L sensors on the scenario, with placements dependent on optimization of network coverage. The set of all possible network configurations is the set of all possible tuples with length L, this results from combination of L paths $\widehat{SSS}$, called $\widehat{SSS}^L$ (2.12).
To optimize the network, the polygons obtained earlier, for a single sensor, can be used, as the possible sensor poses are the same. The main difference in this section is that a logic operation between this polygons must be performed. The coverage area (2.13) is obtained from the union between the polygons corresponding to the states in tuple $\hat{S}_L$.

$$\hat{S}_L = (s_1, ..., s_L) \in \widehat{SSS}^L, \quad if \quad s_{1,...,L} \in \widehat{SSS} \tag{2.12}$$

$$C(\hat{S}_L) = Area\left[\bigcup_{j=1}^{L} Poly(\hat{s}_j)\right], \quad \hat{s}_j \in \hat{S}_L \tag{2.13}$$

Polygon union is performed with an union boolean operator, described in [8], that return a polygon or a set of polygons corresponding to the coverage area from all L sensors. Coverage area is the main criteria to maximize in this problem, uncovered areas are not allowed in most problems, because once the vehicle enters those, there is no way to localize it.

2.2.2 Redundancy

Redundancy is important for a system that must overcome unexpected failures during operation. One of those failures can be the malfunction of a sensor, and so having areas covered by more than one sensor could be important. One way to measure redundancy on this system is exactly the area covered by more then one sensor. The problem is to compute these areas, they are normally disconnected polygons, resulting from an intersection of two or more visibility polygons. To define these areas an operator $R_D(\hat{S}_L)$, $1 < D < L$ is defined in (2.14), it depends on the network configuration ($\hat{S}_L$) and on the degree of redundancy desired (D). D defines how many sensors, at least, cover the area returned by $R_D(\hat{S}_L)$. $\hat{S}_D$ is a set of D sensor states, generated from the states in $\hat{S}_L$. $\mathcal{P}_{\kappa=D}(\hat{S}_L)$ is a set containing all possible combinations of D sensor states.
The area covered by, at least, 2 sensors (2.15) is the most direct redundancy operator, for simplicity, when D is omitted corresponds to $D = 2$. As the number of sensors grows the complexity to extract these areas grows as well, becoming very difficult to compute them. Figure 2.9 show the complexity of this operation for only three polygons.

$$R_D(\hat{S}_L) = Area\left\{\bigcup_{\hat{S}_D \in \mathcal{P}_{\kappa=D}(\hat{S}_L)}^{\#\mathcal{P}_{\kappa=D}(\hat{S}_L)} \left[\bigcap_{u}^{D} Poly(\hat{s}_u)\right]\right\}, \quad \begin{matrix} \hat{s}_u \in \hat{S}_D \subseteq \hat{S}_L \\ \hat{s}_i, \hat{s}_j \in \hat{S}_D \\ \hat{s}_i = \hat{s}_j \Rightarrow i = j \end{matrix} \tag{2.14}$$

[1]Set of sensor states, one state for each network sensor, in practice it is a vector of L *Tags*

$$R(\hat{S}_L) = R_2(\hat{S}_L) = Area\left[\bigcup_{i \neq j} Poly(\hat{s}_i) \cap Poly(\hat{s}_j)\right], \quad \hat{s}_i, \hat{s}_j \in \hat{S}_L, \tag{2.15}$$

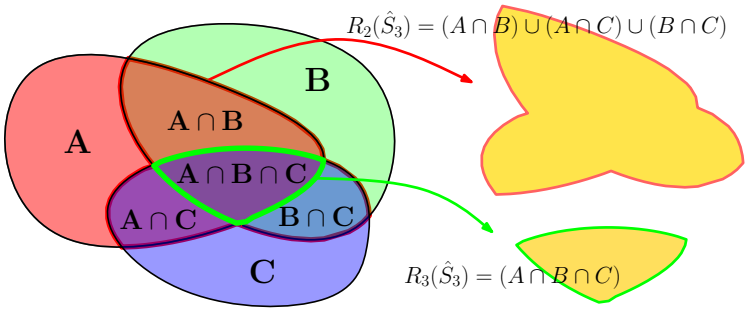


Figure 2.9: Extraction of Redundancy Polygon

As L grows, the number of combinations between polygons grows as well, making $R(\hat{S}_L)$ very hard to compute for a large number of sensors. Another way to compute this function is using a MC method, this gives a approximation of $R(\hat{S}_L)$ and is easier to compute.
This method consists of generating several points, uniformly distributed on the scenario and evaluate, for each point, if it lies inside each polygon or not. Figure 2.10 show the application of this method to an example map, black points are not covered, blue point are covered from just one sensor and the red ones are covered by two. Computing if the point is inside a polygon is a simple operation performed quickly as described in [32], in the end, the percentage of points located inside more then one polygon, is approximately equal to the percentage of area covered by more than one sensor.
The redundancy values are not optimized during this work, their values are still calculated to analyze how the redundancy behaves with the coverage optimization.

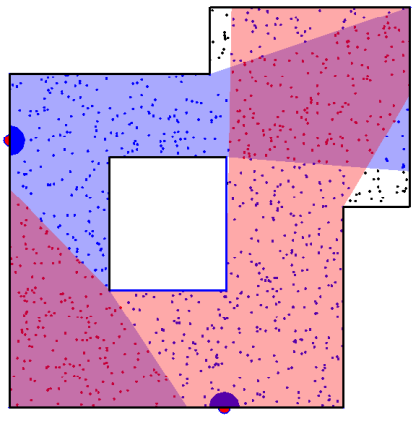


Figure 2.10: MC Redundancy measurement

2.3 Simulated Annealing Optimization

2.3.1 Introduction

Simulated Annealing (SA) is a probabilistic method for global optimization, it can give a good approximation for a global optimum in a complex function. The name is inspired on his heuristic that resembles with a process of heating up and slowly cooling down a material to align and strengthen his molecular structure. On this process, as the material cools down slowly, the atoms tend to converge for a state that minimizes the internal energy. This method was proposed in [6] and [17], it copies the slow cooling down technique, where the state variables, in this case, sensor network configuration $\hat{S}_L$, mimic the atom positions and the cost function ($F(\hat{S}_L)$) mimics the structure internal energy. Cooling down slowly will make the variables states converge to a solution where the cost function is minimal. It can be shown [13] that this method tends to converge to the cost function global minimum as the cooling down process is extended.

The process begins with a random state, and on each iteration SA considers a neighbor state ($\hat{S}'_L$) near the current state ($\hat{S}_L$), depending on the acceptance probabilities $P(\hat{S}_L, \hat{S}'_L, T)$, $\hat{S}_L$ will move to $\hat{S}'_L$ or not. This probability depends on the cost function values for each state and on the parameter T called Temperature. This iteration repeats until some criteria is met or some computation resource is exhausted. The best state found during the process ($\hat{S}^*_L$) will be the optimization result. This process is described in Table 2.4.

The main requirement for probability function $P(\hat{S}_L, \hat{S}'_L, T)$ is that, while T is positive, it must have positive values when $F(\hat{S}_L) < F(\hat{S}'_L)$, meaning that the state can move to a worse solution (move "uphill"), avoiding, this way, the algorithm to become stuck in local minimum.

As T tends to zero, $P(\hat{S}_L, \hat{S}'_L, T)$ must tend to zero if $F(\hat{S}_L) < F(\hat{S}'_L)$ and to a positive value if $F(\hat{S}_L) > F(\hat{S}'_L)$. This means that, as the temperature drops, the algorithm prefers to go "downhill", moving mainly to neighbors that are better solutions, in fact, when $T = 0$ the algorithm becomes greedy, and moves only "downhill".

Probability function can even depend on the difference $F(\hat{S}_L) - F(\hat{S}'_L)$ giving preference to small rather than big movements "uphill", it is often used the Boltzmann factor to compute this probability (2.16).

$$P(s, s', T) = \begin{cases} \propto e^{\frac{F(\hat{S}_L) - F(\hat{S}'_L)}{T}}, & if \quad F(\hat{S}_L) < F(\hat{S}'_L) \quad and \quad T > 0 \\ 0, & if \quad F(\hat{S}_L) < F(\hat{S}'_L) \quad and \quad T = 0 \\ 1, & if \quad F(\hat{S}_L) > F(\hat{S}'_L) \quad and \quad T = 0 \end{cases} \tag{2.16}$$

Coverage optimization problems are difficult, because the evaluation function, $C(\hat{S}_L)$, have a bad behavior, very local minima and maxima, discrete functions, has shown in Figure 2.8 for $C(\hat{S}_1)$ of a simple map, and, has the number of sensors increase, complexity grows exponentially.

An entire state space search would be impossible for multiple sensors, so a MC method like SA is preferred, it is very often used in discrete state spaces. It is discrete and has the capability to avoid being trapped in local minima, so it is suited to this problem. Other methods like Genetic Algorithms, [22], have these similar characteristics but they are much

Table 2.4: Simulated Annealing Algorithm

1. Initialize $\hat{S}_L$ and $\hat{S}_L^*$;
2. While computation resources available and criteria not met:
   (a) Compute a new neighbor state $\hat{S}'_L$;
   (b) If $F(\hat{S}'_L) < F(\hat{S}_L^*)$, $\hat{S}_L^* \leftarrow \hat{S}'_L$;
   (c) If $P(\hat{S}_L, \hat{S}'_L, T) > random()$*, $\hat{S}_L \leftarrow \hat{S}'_L$;
3. return $\hat{S}_L^*$

* $random()$ returns a random number from 0 to 1 generated from uniform distribution.

more complex to implement. We followed the SA approach instead other MC methods mainly due to his simplicity.

2.3.2 Temperature Schedule

SA performance depends critically on the temperature schedule, this is the way that the parameter T changes along the optimization process, when T is high, states change widely, as the temperature drops the search becomes more thorough. The most common approach is decreasing by a factor α, between zero and one, in each iteration, $T(k) \leftarrow \alpha T(k-1)$, being k the iteration number.
On our optimization the temperature schedule was divided in three parts (2.17), the initial temperature (T_{ini}) was kept constant in the beginning to let the algorithm search almost randomly through the entire space, then, on second part the common approach, reducing temperature by a factor α and finally some iterations with $T = 0$, to refine the solution. To perform this, we defined three parameters, k_α, k_0 and k_{end}, stating the iteration to start α decay, to put $T = 0$ and to stop the algorithm. Figure 2.11 shows a possible temperature schedule layout respecting the above description.
The method to choose neighbors ($\hat{S}'_L$) was adding a random tuple to the current one, but the norm of this tuple was different for each part. First a the algorithm performs large variations in network configuration, allowing the solution to travel widely on state space, then the random tuple decreases, in the end it just makes small adjustments.

$$T(k) = \begin{cases} T_{ini} & k < k_\alpha \\ \alpha T(k-1) & k_\alpha < k < k_0 \\ T = 0 & k_0 < k < k_{end} \end{cases} \tag{2.17}$$

2.3.3 Proposed Method

The objective of this optimization problem is to maximize the area covered by a network of L sensors installed on indoor scenarios walls. The search space, or state space for this problem is formed by all possible tuples with size L, generated from combinations of sensors

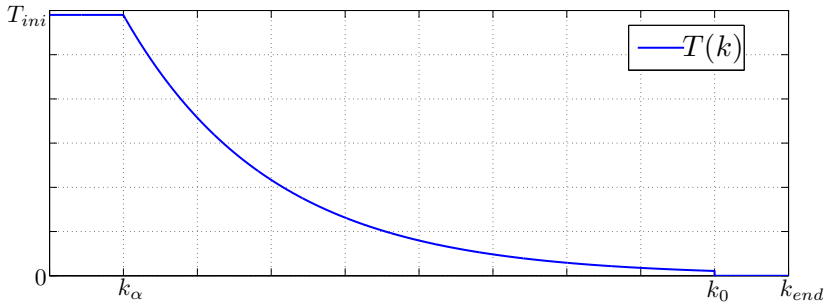


Figure 2.11: Temperature schedule layout

configurations $d\widehat{SSS}^L$. This set is resemblant to $\widehat{SSS}^L$ but for the discrete set $d\widehat{SSS}$ (2.18).

$$(s_1, ..., s_L) \in d\widehat{SSS}^L \; if \; s_{1,...,L} \in d\widehat{SSS} \tag{2.18}$$

Analyzing the search space and the cost function characteristics, SA algorithm is suitable for this problem :

- Discrete state space, $d\widehat{SSS}^L$. Normally used in SA implementations.
- non-Convex cost function, the coverage area along the possible sensor configurations has many local minima and narrow valleys, as seen in the example of Figure 2.8, SA algorithm is suited for this problem as it can avoid being trapped in local minima.
- Wide search space, being impossible to search the entire space. SA is a MC method which is needed to compute a solution in a reasonable time.
- Implementation simplicity, SA is one of the most simple to implement MC methods. As this is just a first step towards the localization system itself the method simplicity is an important criteria to choose it.

Meeting the most important criteria that we have establish, SA is the algorithm chosen to optimize sensor networks.

To compute the network with optimal coverage using SA algorithm, the cost function is actually a fitness function because it s maximized instead of minimized. This is not a big challenge and is done on the algorithm implementation itself. From now on the SA is "inverted" and tries to maximize the fitness function. This function is the coverage of the network $C(\hat{S}_L)$ (2.13). This function is evaluated on the search space $d\widehat{SSS}^L$ during each run of SA. The best solution $(\hat{S}_L^*)$ is the tuple (2.20) for which the coverage function gets the higher value $(C(\hat{S}_L^*))$. The objective of the optimization process is to find the global maximum of this function (2.19).

$$C(\hat{S}_L^*) = max_{\hat{S}_L} \, C(\hat{S}_L) \tag{2.19}$$

$$(s_1, ..., s_L) = \hat{S}_L^* = argmax_{\hat{S}_L} \, C(\hat{S}_L) \tag{2.20}$$

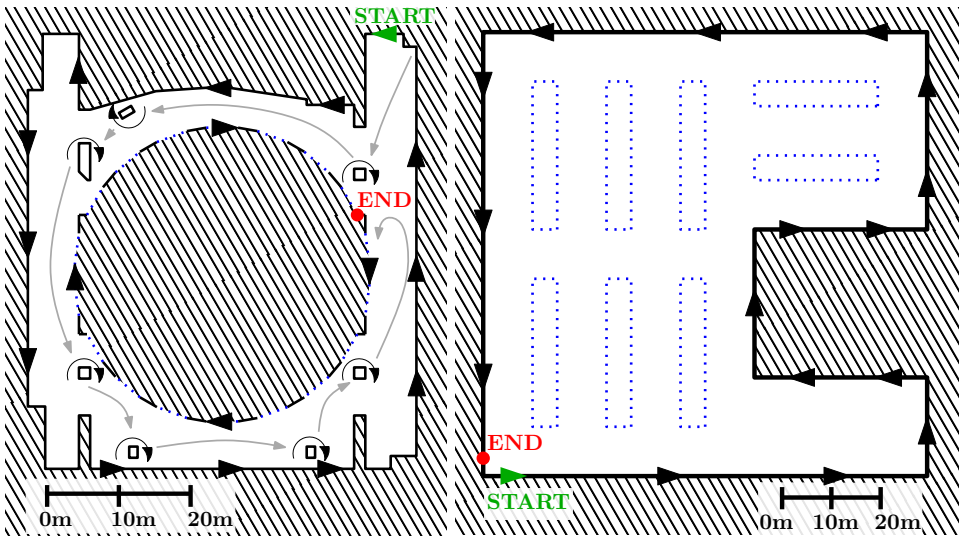
The output of the algorithm is not always the global maximum, but for multiple runs[2] of SA, the probability of finding the global maximum rises. With more runs of the algorithm

[2]Executions of SA with the same state space, which increases the probability of finding the global maximum

more certainty we have about the location of the maximum.
The tuple that gives rise to the maximum coverage contains the configurations for the sensors on the network, placing them with those configurations guarantees the optimal coverage for a network with L sensors.

2.4 Experimental Results

For experimental proposes two maps were used, the first is the indoor map of a floor of the TB in ITER shown on Figure 2.12(a) where the solid edges are walls and dashed ones are VV doors. The second map is a general propose warehouse building on Figure 2.12(b), synthesized to prove the effectiveness of this method in other environments, solid edges are walls and dashed edges are shelves. Sensors can only be mounted on solid walls. Figures 2.12(a) and 2.12(b) show the path used to create the $d\widehat{SSS}$ set, this was done with a discretization step of 0.2 m for translation and $\pi/60$ for rotations.
The CG for these two maps are shown in Figure 2.13(a) and 2.13(b) , this is sufficient to notice the behavior of this functions for a network of just one single sensor, very hard to optimize as the sensor network grows.



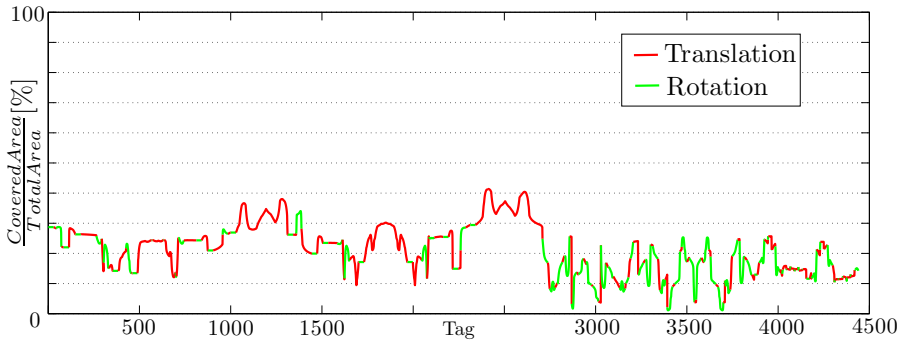
(a) Tokamak map (b) Warehouse map

Figure 2.12: TB and WHB with path along $d\widehat{SSS}$

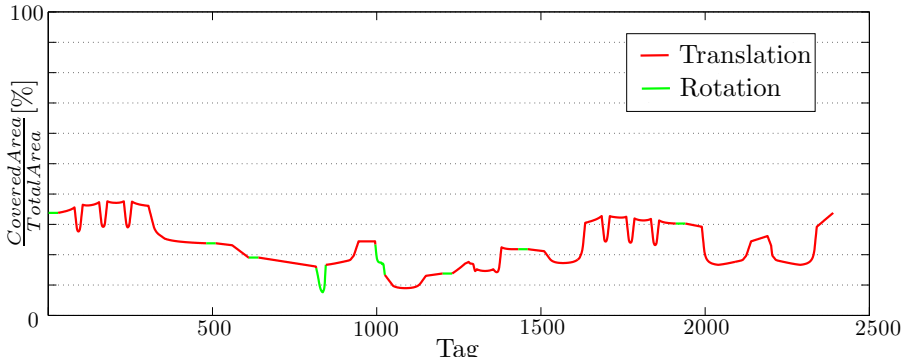
The optimization results were obtained using temperature schedule presented in section 2.3.2, fixed the parameters $(k_\alpha, k_0, k_{end}) = (.05k_{end}, .95k_{end}, 5000)$ the number of iterations is fixed, equal to 5000. This was establish taking in consideration the trade-off between optimization time and fitness value achieved on each run of SA. For each L, the SA algorithm runs ten times, and the best result obtained in these runs is assumed to be the optimal $(\hat{S}_L^*)$.

Figure 2.15 shows the coverage $C(\hat{S}_L^*)$ and redundancy $R(\hat{S}_L^*)$ depending on L, for the network with maximum coverage. The graph also show the minimum coverage $(min(C(\hat{S}_L)))$, the minimum and the maximum redundancy $(min(R(\hat{S}_L))$ and $max(R(\hat{S}_L)))$ obtained in the ten SA runs., the behavior of maximum coverage as the number of sensors grow is similar on both maps, and is typical for indoor scenarios. The incremental gain (2.21) is a good evaluator to choose the number of sensor forming the network if the problem depends only on the coverage.

$$G(L) = C(\hat{S}_L^*) - C(\hat{S}_{L-1}^*) \tag{2.21}$$



(a) Tokamak building



(b) Warehouse building

Figure 2.13: Coverage Graph for both maps

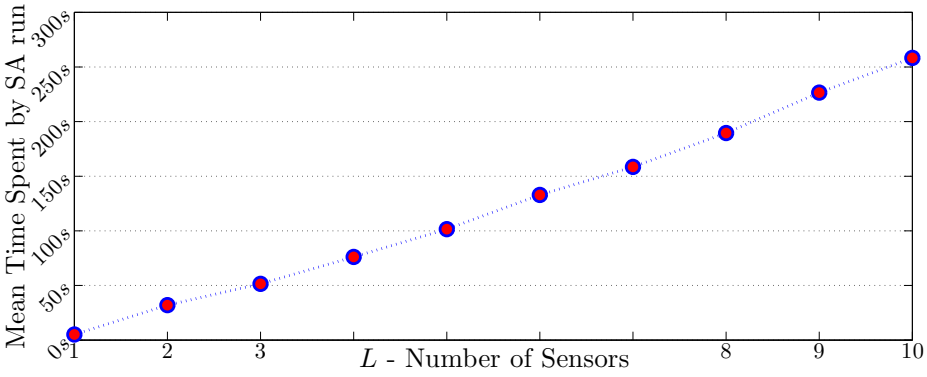


Figure 2.14: Optimization time depending on number of sensors installed

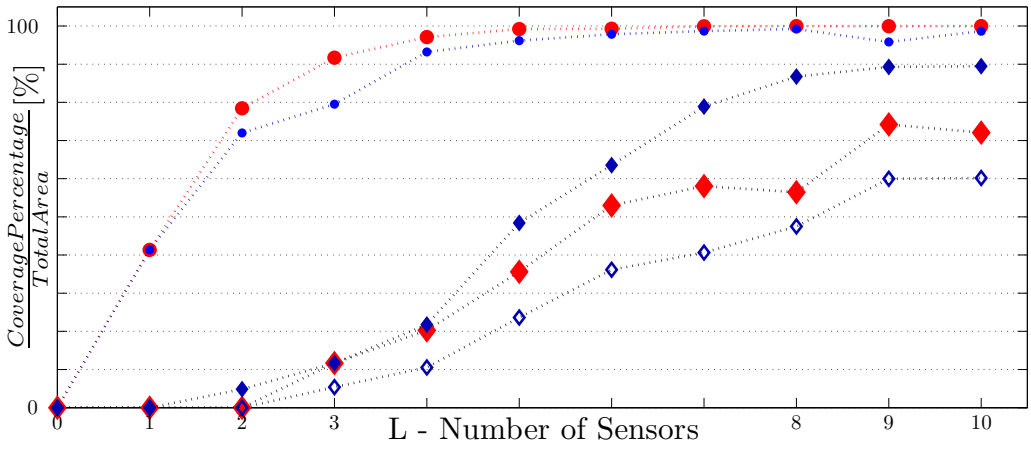
For scenarios where there is no tolerance for failure, redundancy of the system is very important, although not optimized on this work. The redundancy grows with the coverage, but since it is not optimized has more variation then the coverage along the different runs. In some situations it is better not to choose the maximum coverage network, because there are solutions where, losing some coverage, the network gains a lot of redundancy. For the TB values (2.15(a)) for a network with eight sensors ($L = 8$), the coverage is almost the same on all runs of SA, but the maximum coverage network has a redundancy below 60% while there is a network with almost 90% redundancy, with nearly the same coverage, which is a better solution considering the two criteria. Naturally, in this case the best solution does not coincide with the maximum coverage criteria.

Detailed values for coverage and redundancy, respecting the optimal network configurations $\hat{S}_L$, are given on Tables 2.5 and 2.6, including redundancy in higher levels, from $D = 2$ to $D = 6$. Optimal coverage network configurations $\hat{S}_L^*$ are shown in Figure 2.16 and 2.17, to notice the behavior when a new sensor is added, the tendency is to stay away from the others to maximize coverage over the entire scenario. When L is high, the network attributes a sensors for very specific areas, where the others can not cover, and so the separation between them is less visible.

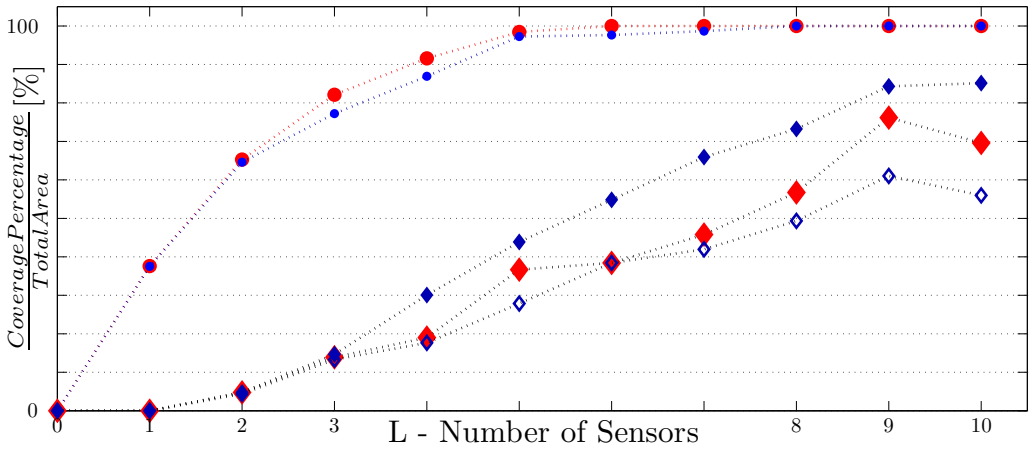
Redundancy with $D > 2$ is a indicator of coverage from multiple sensors, this can have an impact on the quality of the localization system prediction, since the system has more observations on the vehicle when it travels through these areas.

The redundancy is not optimized on this work because it takes a long time computing, the present values, are processed off-line, after the SA algorithm completes the coverage optimization.

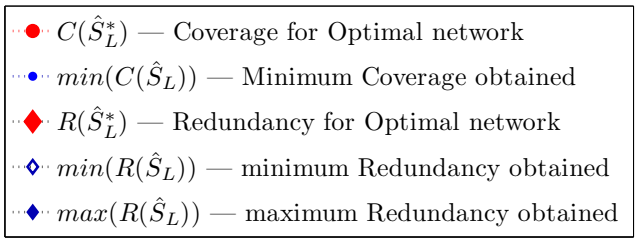
The polygon logic operations take some time, as L grows, the complexity rises, initially visibility polygons are star-shaped, making the union operation very simple, but the result from this operation is a, no longer star-shaped, polygon, it can contain holes, or even become a set of disconnected polygons. So logic union operation is the critical operation in the algorithm. Figure 2.14 shows the mean processing time, in all ten runs of SA, time values are not very important, since they depend on the processor used, but still, they yield that the operation complexity depends linearly on the number of sensors L.



(a) Tokamak building



(b) Warehouse building



(c) Legend

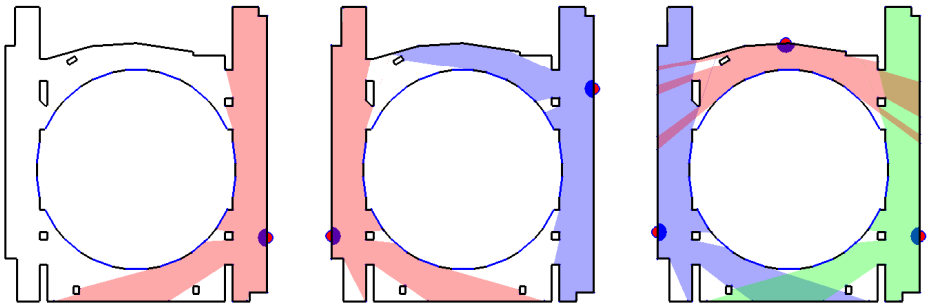
Figure 2.15: Coverage and Redundancy depending on number of sensors installed

Table 2.5: $C(\hat{S}_L)$ and $R_D(\hat{S}_L)$ for Tokamak Building

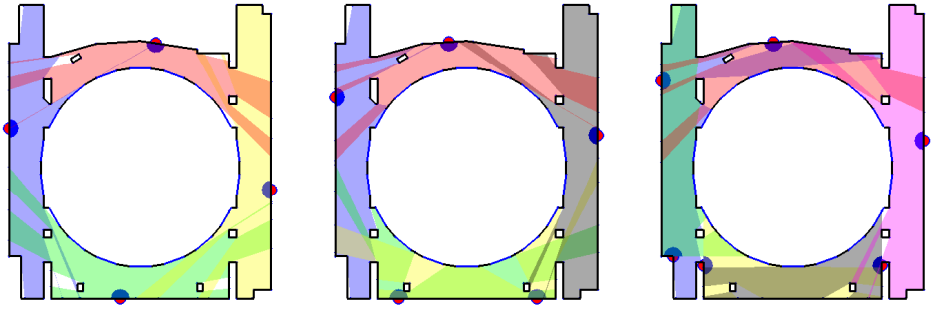
L	$C(\hat{S}_L)$	$R_D(\hat{S}_L)$				
		$D=2$	$D=3$	$D=4$	$D=5$	$D=6$
1	41.4%					
2	78.4%	0%				
3	91.7%	12%	0%			
4	97.1%	20%	0%	0%		
5	99.2%	36%	1%	0%	0%	
6	99.3%	53%	5%	0%	0%	0%
7	99.9%	58%	11%	0%	0%	0%
8	100.0%	56%	16%	2%	0%	0%
9	100.0%	74%	31%	8%	0%	0%
10	100.0%	72%	38%	7%	0%	0%

Table 2.6: $C(\hat{S}_L)$ and $R_D(\hat{S}_L)$ for Warehouse Building

L	$C(\hat{S}_L)$	$R_D(\hat{S}_L)$				
		$D=2$	$D=3$	$D=4$	$D=5$	$D=6$
1	37.6%					
2	65.3%	5%				
3	82.1%	14%	0%			
4	91.6%	19%	1%	0%		
5	98.4%	37%	9%	0%	0%	
6	100.0%	38%	5%	0%	0%	0%
7	100.0%	46%	13%	4%	1%	0%
8	100.0%	57%	36%	17%	0%	0%
9	100.0%	76%	30%	4%	0%	0%
10	100.0%	70%	39%	16%	7%	2%

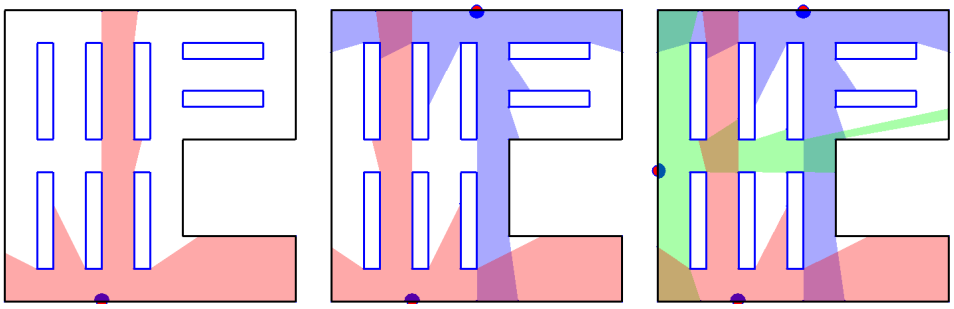


(a) $C(\hat{S}_1) = 41\%$ (b) $C(\hat{S}_2) = 78\%$ (c) $C(\hat{S}_3) = 92\%$

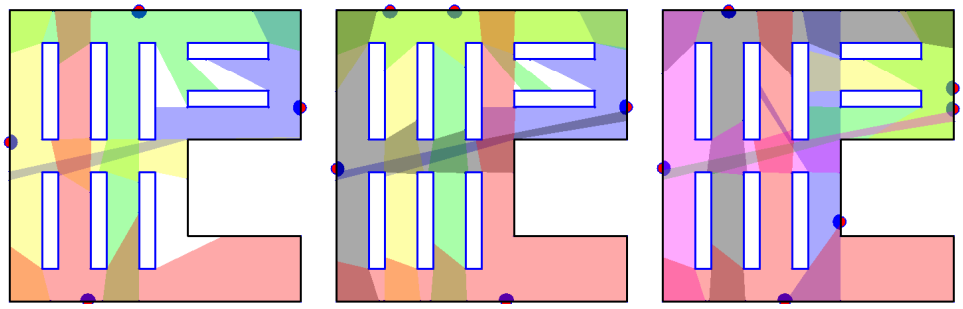


(d) $C(\hat{S}_4) = 97\%$ (e) $C(\hat{S}_5) = 99\%$ (f) $C(\hat{S}_6) = 99\%$

Figure 2.16: Solutions for Tokamak building, $1 < L < 6$



(a) $C(\hat{S}_1) = 38\%$ (b) $C(\hat{S}_2) = 65\%$ (c) $C(\hat{S}_3) = 82\%$



(d) $C(\hat{S}_4) = 92\%$ (e) $C(\hat{S}_5) = 98\%$ (f) $C(\hat{S}_6) = 100\%$

Figure 2.17: Solutions for Warehouse building, $1 < L < 6$

How many sensor should be mounted in a scenario to ensure a good performance for the localization system, and where to put those sensors are the main questions to answer on this chapter.

- **How many sensors**, taking only the coverage in consideration, a possible rule to decide L is defined in 2.22, using the incremental gain (2.21). If these gain drops below a certain threshold (ξ) means that there is no gain in adding more sensors. In a practical situation more criteria could be considered, as the redundancy or areas of interest, some areas could be uncovered but there is no way for the vehicle to reach them, so there is no interest on them. The redundancy has an important role on the robustness of the system and in some situations it is profitable to give up on some coverage to gain redundancy as explain for the TB example in Figure 2.15(a) for $L = 8$.

$$L = min \;\; n : G(n) < \xi \tag{2.22}$$

- **Where to put the sensors**, with L fixed and considering only the coverage, the optimized network $\hat{S}_L^*$ is the best way to distribute the sensors on the scenario. If we choose a non-optimal solution, for example, to enhance the redundancy, the sensor placement must always respect the configurations on the tuple $\hat{S}_L$.
  The actual placement on the real scenario can generate deviations from the predefined positions. This deviations can affect the performance of the localization system adding static errors.
  An expeditious method to avoid these errors is a process of automatic calibration after the mounting process. These method is not yet implemented and is left as an open issue to address in future work.

Chapter 3

Vehicle Localization

3.1 Problem Statement

The localization problem is well known in robotics. In order to plan or follow a trajectory, to reach a certain goal, the knowledge of the correct position of the vehicle is crucial. Either by remote control handling with human operators or by autonomous guiding systems, this is a crucial part during remote handling operations in ITER buildings.
The requirement in a known environments is to localize the vehicle given the map layout. Usually this is made using on board sensors, with methods like EKF Localization or Monte Carlo Localization (MCL) ([14]). In this work, the main difference is the installation of sensors in the scenario instead, leaving the vehicle just as an actuator. On this approach, the environment observes the vehicle and not the other way around.
This specification brings new challenges, in common methods, with the vehicle movement, all measures are expected to change, since they are observations of the environment. In this application some change and others stay the same because some observe the vehicle and others do not. There is no trivial way to decide if all measures are interesting or just the ones hitting the vehicle, and no trivial way to classify them as hitting or not hitting the vehicle. The main goal is the creation of a localizations system that, overcoming this issues, integrates the sensor measurements and estimate the vehicle pose correctly with a certain accuracy. The vehicle pose (3.1) tells the position of vehicles' center and respective orientation.

$$x_t = \begin{bmatrix} x_r^t & y_r^t & \theta_r^t \end{bmatrix}^T \tag{3.1}$$

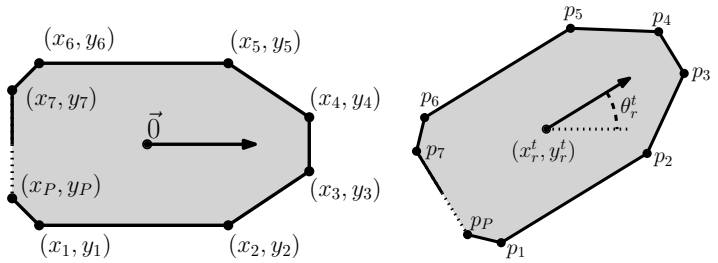
$$\hat{x}_t = \begin{bmatrix} \hat{x}_r^t & \hat{y}_r^t & \hat{\theta}_r^t \end{bmatrix}^T \tag{3.2}$$

Localization problem consists on estimating the correct pose, returned by the vector $\hat{x}_t$ (3.2), knowing the surrounding environment. This includes the map layout, the vehicle layout and the correct sensors positions. Map layout representation is equal to the one described in Chapter 2 on section 2.1.
Vehicle layout is described by a polygon V_0 defining the corner positions in CCW when the vehicle pose is $x_t = [0\ 0\ 0]$ (3.3) on Figure 3.1(a). $V(x_t)$ (3.4) on Figure 3.1(b) is the vehicle

in pose $x_t = [x_r^t, y_r^t, \theta_r^t]^T$.

$$V_0 = \begin{bmatrix} x_1 & x_2 & \cdots & x_P \\ y_1 & y_2 & \cdots & y_P \end{bmatrix} \tag{3.3}$$

$$V(x_t) = [p_1\ p_2\ \cdots\ p_P] = \begin{bmatrix} \cos\theta_r^t & -\sin\theta_r^t \\ \sin\theta_r^t & \cos\theta_r^t \end{bmatrix} V_0 + \begin{bmatrix} x_r^t & x_r^t & \cdots \\ y_r^t & y_r^t & \cdots \end{bmatrix} \tag{3.4}$$



(a) On origin (b) After transformation

Figure 3.1: Vehicle layout

Sensor measurements come from a network of LRF sensors [16], each one with specific configurations (s_i) (3.5). x_s^i, y_s^i and θ_s^i are the pose of the i-th sensor, Φ_s^i is the FoV, δ_s^i the angular resolution and σ_s^i the standard deviation for distance measurement errors. Some of these parameters are imported from the optimization problem of Chapter 2, only δ_s^i and σ_s^i are established here depending on equipment installed.

The integration of several sensors is mandatory to cover all possible vehicle positions, the sensor network is defined in S (3.6) containing L sensors.

$$s_i = \begin{bmatrix} x_s^i & y_s^i & \theta_s^i & \Phi_s^i & \delta_s^i & \sigma_s^i \end{bmatrix}^T = \begin{bmatrix} [\mathcal{T}(\hat{s}_i)]^T & \delta_s^i & \sigma_s^i \end{bmatrix}^T \tag{3.5}$$

$$S = [s_1\ \ s_2\ \ \ldots\ \ s_L] = \begin{bmatrix} x_s^1 & y_s^1 & \theta_s^1 & \Phi_s^1 & \delta_s^1 & \sigma_s^1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_s^L & y_s^L & \theta_s^L & \Phi_s^L & \delta_s^L & \sigma_s^L \end{bmatrix}^T \tag{3.6}$$

LRF sensors measure distances to obstacles around them, for each angular position there is a distance. Measurements acquired from each sensor s_i are arranged in the array $z(s_i)$ (3.7) where d_j^i is a distance corresponding to laser beam with direction φ_j^i acquired by the i-th sensor in scenario. The number of measurements for each sensor depends on his FoV and angular resolution. These can be converted into spacial points in Cartesian coordinates using the tranformation described in 3.8. The measurements available from all network come from all the L sensors installed in the scenario (3.9).

$$z(s_i) = \begin{bmatrix} d_1^i & \cdots & d_{P_i}^i \\ \varphi_1^i & \cdots & \varphi_{P_i}^i \end{bmatrix}^T \tag{3.7}$$

$$z_{cart}(s_i)_j = \begin{bmatrix} d_j^i \cos\varphi_j^i + x_s^i \\ d_j^i \sin\varphi_j^i + y_s^i \end{bmatrix}^T, \qquad z_{cart}(s_i) = \begin{bmatrix} z_{cart}(s_i)_1 \\ \vdots \\ z_{cart}(s_i)_{P_i} \end{bmatrix}, \qquad P_i = \left\lfloor \frac{\Phi_s^i}{\delta_s^i} \right\rfloor + 1 \tag{3.8}$$

$$Z = \begin{bmatrix} z(s_1) & \cdots & z(s_L) \end{bmatrix} \tag{3.9}$$

Measurements coming from LRF are corrupted with noise (Figure 3.3), mainly in the range measured, the error in angular position is very small and neglected in this work. The error in ranges can be modeled as corrupted with zero mean gaussian noise as shown in [40], for a widely used commercial sensor. The variance for this noise is described for each sensor by his standard deviation, σ_s^i.

Based on these problem framework, two Baysian methods of estimation are tested, EKF and PF [34], to choose which is better suited to the task. They will integrate sensor measurements with dynamic information, like velocity commands, to achieve an estimation with a certain accuracy.

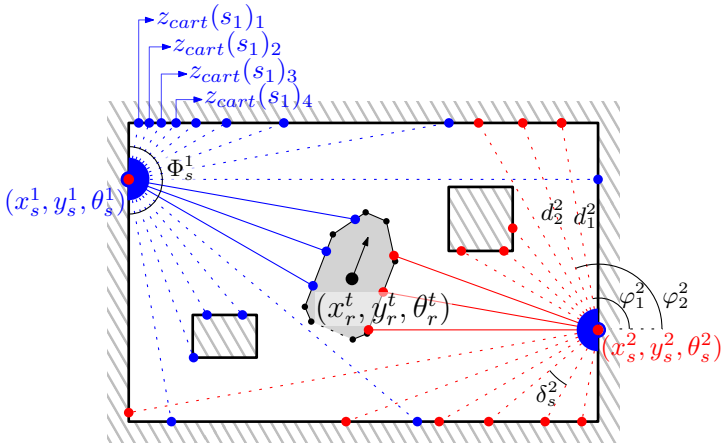


Figure 3.2: Vehicle localization framework

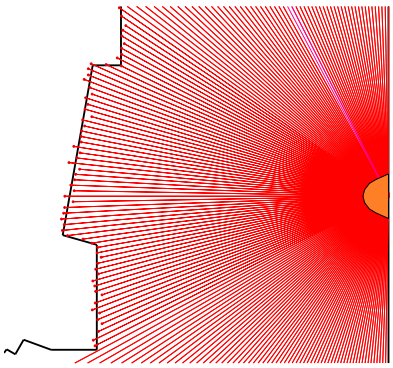


Figure 3.3: Measurement noise

3.2 Extended Kalman Filter

3.2.1 Introduction

Kalman filter (KF) is a very useful tool in robotics field, it is used in data fusion and sensor fusion, his typical application are the estimation of state variables based on noisy measurements. It has popularity in localization, tracking in computer vision etc.
It is a discrete-time filter having two major steps per iteration, prediction and update. For each time step t, KF has a belief over the state x_t, this belief is computed based on all previous information, measurements (z_t) and control inputs (u_t). On prediction step, KF estimates a prior belief (3.11), based on previous belief (3.10), control inputs and transition model (called movement model later on). The update step uses the sensor observations and, from baysian inference rule, computes a posteriori belief (3.12) using an observation model, updating the the prior belief. KF is a gaussian process, so all beliefs, at time t, are represented by mean μ_t and covariance Σ_t.

$$bel(x_{t-1}) = p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) \sim \mathcal{N}(\mu_{t-1}, \Sigma_{t-1}) \tag{3.10}$$

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \sim \mathcal{N}(\overline{\mu}_t, \overline{\Sigma}_t) \tag{3.11}$$

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) = \eta p(z_t|x_t)p(x_t|z_{1:t-1}, u_{1:t}) \sim \mathcal{N}(\mu_t, \Sigma_t) \tag{3.12}$$

KF is a dynamic filter using predefined linear models, a state transition model (3.13) and an observation model (3.14). A_t is the state transition matrix, B_t is the control-input matrix and C_t is the observation model's matrix. Zero mean gaussian noise is added to linear models each iteration, Q_t and R_t are the process and the observation noise covariance matrices, respectively.

$$x_t = A_t x_{t-1} + B_t u_t + w_t, \qquad w_t \sim \mathcal{N}(0, Q_t) \tag{3.13}$$

$$z_t = C_t x_t + v_t, \qquad v_t \sim \mathcal{N}(0, R_t) \tag{3.14}$$

Using this framework, equations 3.15 and 3.16 form the prediction step, yielding a prior belief ($\overline{\mu}_t$, $\overline{\Sigma}_t$), an optimal Kalman gain (K_t) is computed (3.17)taking in consideration the certainty of measurements and ambiguity in current predicted estimation. This gain is useful to the update step, after measurement acquisition both mean and covariance are updated (3.18 and 3.19) making the posteriori belief (μ_t, Σ_t).

$$\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t \tag{3.15}$$

$$\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + Q_t \tag{3.16}$$

$$K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + R_t)^{-1} \tag{3.17}$$

$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t \bar{\mu}_t) \tag{3.18}$$

$$\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t \tag{3.19}$$

Under certain initial conditions and respecting the gaussian and linear assumptions, KF tends to minimize the mean-squared error between predicted and real states [28].

The main problem with KF is the limitation to linear systems, in real world this have very limited applications.

EKF is an improvement from simple KF, through a linearization via Taylor expansion about the current mean and covariance, EKF supports non-linear systems using the same filter framework. In this case the transition (3.20) and observation (3.21) models have no need to be linear, instead, they must be differentiable.

$$x_t = f(x_{t-1}, u_t) + w_t, \qquad w_t \sim \mathcal{N}(0, Q_t) \tag{3.20}$$

$$z_t = h(x_t) + v_t, \qquad v_t \sim \mathcal{N}(0, R_t) \tag{3.21}$$

During filter operation, the steps are identical to KF, the EKF algorithm is synthesized on Table 3.1 .To keep the same mathematical treatment, a linearization of model functions is needed. The first order Taylor expansion is the linearization method applied, F_t and H_t (3.22) are the jacobians, around the current prediction state, of transition and observation model, respectively.

Table 3.1: EKF algorithm

1. Initialize with current belief at time $t = 0$, (μ_0, Σ_0)
2. For each time step t, $t = t + 1$, new commands u_t and measurements z_t
   (a) Prediction step:
   $\bar{\mu}_t = f(\mu_{t-1}, u_t)$
   $\bar{\Sigma}_t = F_t \Sigma_{t-1} F_t^T + Q_t$
   (b) Compute Kalman gain:
   $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + R_t)^{-1}$
   (c) Update step:
   $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$
   $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$

$$F_t = \left.\frac{\partial f}{\partial x}\right|_{\mu_{t-1}, u_t} \qquad H_t = \left.\frac{\partial h}{\partial x}\right|_{\bar{\mu}_t} \tag{3.22}$$

In this case the filter's estimation corresponds to vehicle's pose, $\mu_t = [\hat{x}_r^t \;\; \hat{y}_r^t \;\; \hat{\theta}_r^t]^T$ which is estimated state, at time t. Although by linearization, the EKF overcomes the greater limitation of KF, the linear models, it has some problems with non-smooth function, but it works well in many cases ([19], [30]). For this research, with laser range finder sensors it was interesting to test the behavior of EKF, by his simple implementation and great adaptability to many problems.

3.2.2 Proposed Method

The method proposed is a direct implementation of EKF, where μ_t is the filter estimation of vehicle pose, $\hat{x}_t = \mu_t$, the prediction step is based on the movement model computing a

prior belief of vehicle pose, $\bar{x}_t$, using previous estimate and control inputs, u_t. The update step is based on observation model, it integrates the sensor's measurements to update the prior belief and estimate x_t with minimal error.

Movement model

The movement model $f(\hat{x}_{t-1}, u_t)$ predicts the pose of the vehicle in the current time step, using the previous pose and the input-control commands. In this case the movement model is non-linear and depends on vehicle kinematics. From vehicle kinematics and current pose prediction is possible to extract a velocity vector for the center of the vehicle, $g_t = \left[v_x^t, v_y^t, v_\theta^t\right]^T$. To compute the priori belief, is necessary a discrete integration of this velocities (3.43), with time steps corresponding to algorithm time step ΔT.

$$\bar{x}_t = \hat{x}_{t-1} + \Delta T\, g_t + w_t, \qquad w_t \sim \mathcal{N}(0, Q_t) \qquad Q_t = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{3.23}$$

Where Q_t is the covariance matrix for the process noise, associated with movement errors, assuming source of noise independent for each movement component.

On the current application the vehicle is the TCS, his driving system is composed by two wheels that can have different directions and rotation velocities. The actual control commands given to the robot's controllers are the velocities and headings for these two wheels as shown in Figure 3.4 $u_t = [v_F^t, v_R^t, \alpha_F^t, \alpha_R^t]^T$. The linearization around the current prediction $\bar{x}_t$ is described in (3.24). These are specific kinematics for the TCS vehicle but the method is suited for any kind of vehicle.

$$g_t = \begin{bmatrix} v_x^t \\ v_y^t \\ v_\theta^t \end{bmatrix} = \begin{bmatrix} v_R^t \cos(\hat{\theta}_r^t + \alpha_R^t) + v_F^t \cos(\hat{\theta}_r^t + \alpha_F^t) \\ v_R^t \sin(\hat{\theta}_r^t + \alpha_R^t) + v_F^t \sin(\hat{\theta}_r^t + \alpha_F^t) \\ \frac{1}{bW}\left[-v_R^t \sin(\alpha_R^t) + v_F^t \sin(\alpha_F^t)\right] \end{bmatrix} \tag{3.24}$$

v_F^t and v_R^t are the velocities for front and rear wheels, α_F^t and α_R^t are the respective headings with reference to robot orientation θ_r^t, $\hat{x}_t$ is the current prediction and bW is the distance between wheels' center.

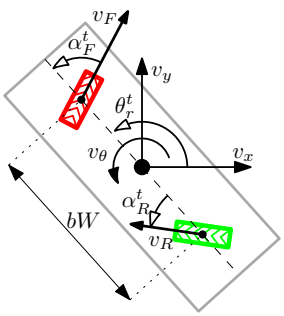
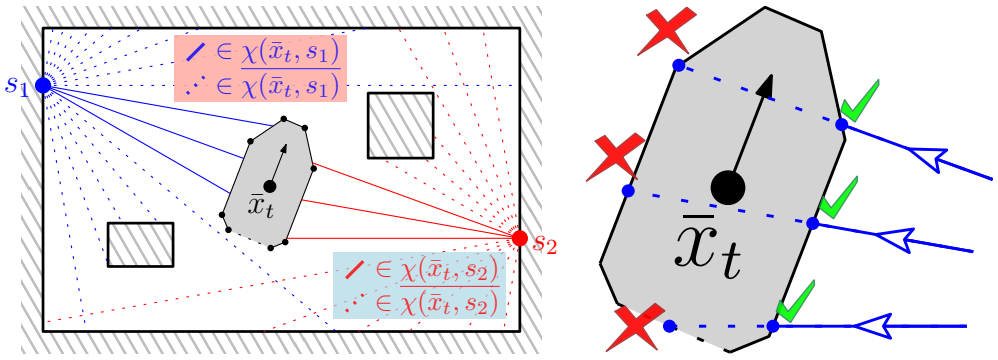


Figure 3.4: TCS vehicle kinematics

Observation model

The observation model $h(\bar{x}_t)$ is a non-linear function that predicts measurements based on a estimation of vehicle pose $\bar{x}_t$. This observation model predicts, for each sensor, the distances measured by a laser ray casted in predefined angular position φ_j^i. These angles are predefined by the sensor orientation θ_s^i, angular resolution δ_s^i and FoV Φ_s^i.



(a) $\chi(x_t, s_i)$ and $\overline{\chi(x_t, s_i)}$ sets example

(b) Measurement distance prediction

Figure 3.5: EKF observation model

A ray is casted in each direction φ_j^i, all the possible intersections with vehicle's edges are considered in 3.25, where p_k is a point from the vehicle layout $V(x_t)$. A laser ray only hits the vehicle if $0 < c_e < 1$ and $l_e > 0$, being e the index of the vehicle edge. There are always more than one intersection with the vehicle (Figure 3.5(b)), the correct range, $\bar{d}_j^i$, corresponds to the intersection with minimal l_e because, first edge occludes the ones behind. The index of the edge facing the ray is denoted $\hat{e}$.

Since the laser measurements hitting the map walls are not dynamic with the vehicle pose, there is no need for an observation model for them, so for each sensor, the model prediction, $h_i(\bar{x}_t)$ (3.26), is an array of distances, $\bar{d}_j^i$, one for each direction, φ_j^i, that hits the vehicle. The length of this array, $J = \#\chi(\bar{x}_t, s_i)$, corresponds to the number of laser rays hitting the vehicle, and depends on s_i and $\bar{x}_t$.

$\chi(\bar{x}_t, s_i)$ is a set of angular positions, such that laser rays casted in those directions, from sensor s_i, hit the vehicle with pose $\bar{x}_t$. $\overline{\chi(\bar{x}_t, s_i)}$ is the set of angular positions which rays do not hit the vehicle. Figure 3.5(a) shows the parameters and variables used in observation model, for a general vehicle layout $V(\bar{x}_t)$ (3.4). Combining all sensor predictions, $h(\bar{x}_t)$ is an array with prediction of Z but only for measures that hit the vehicle.

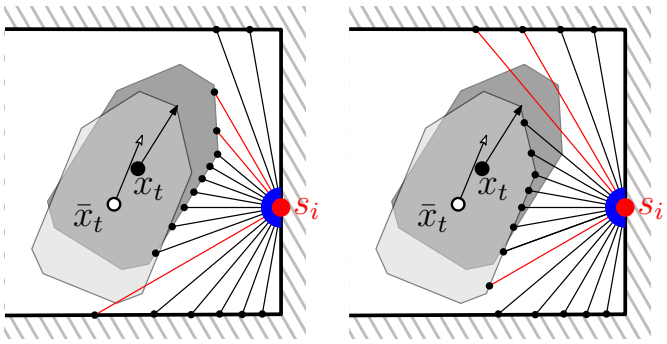
$$\begin{cases} A(\varphi_j^i, p_k) = \begin{bmatrix} \cos\varphi_j^i & x_k - x_{k-1} \\ \sin\varphi_j^i & y_k - y_{k-1} \end{bmatrix}, \qquad p_k = \begin{pmatrix} x_k \\ y_k \end{pmatrix} \\ \begin{bmatrix} l_1 \\ c_1 \\ \vdots \\ l_{P-1} \\ c_{P-1} \end{bmatrix} = \begin{bmatrix} A(\varphi_j^i, p_2) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & A(\varphi_j^i, p_P) \end{bmatrix}^{-1} \begin{bmatrix} p_2 - \begin{pmatrix} x_s^i \\ y_s^i \end{pmatrix} \\ \vdots \\ p_P - \begin{pmatrix} x_s^i \\ y_s^i \end{pmatrix} \end{bmatrix} \\ \bar{d}_j^i = l_{\hat{e}} = \min \begin{Bmatrix} l_1 & \cdots & l_{P-1} \end{Bmatrix} \qquad l_e > 0,\ 0 < c_e < 1 \end{cases} \tag{3.25}$$

$$h_i(x_t) = \begin{bmatrix} \bar{d}_1^i & \cdots & \bar{d}_{J_i}^i \\ \varphi_1^i & \cdots & \varphi_{J_i}^i \end{bmatrix} \tag{3.26}$$

$$h(x_t) = \begin{bmatrix} h_1(x_t) & \cdots & h_L(x_t) \end{bmatrix} \tag{3.27}$$

The problem with these approach is that, at time t, the EKF does not know the true pose of the vehicle, x_t, it knows the ápriori estimation, given by the movement model, $\bar{x}_t$. This will have some impact in the observation model, since in reality, the laser rays hitting the robot are not the same as the ones hitting in estimation.

There are four distinct sets of measurements, the ones hitting the robot in real pose and in estimated pose, $\chi(x_t, s_i)$ and $\chi(\bar{x}_t, s_i)$, and the ones hitting map walls or fixed obstacles in real and estimated pose, $\overline{\chi(x_t, s_i)}$ and $\overline{\chi(\bar{x}_t, s_i)}$, respectively. Figure 3.6 presents an example where the estimation is not far from real pose, but the sets does not coincide. This



(a) Ranges for pose x_t (b) Ranges for pose $\bar{x}_t$

Figure 3.6: Measurement model non-gaussian, non-smooth behavior

situation represents a non-gaussian and non-smooth behavior of the measurement model, since for small differences in vehicle pose there are wide changes in some measurements. EKF assumes always gaussian error on measurements and also computes the kalman gain matrix using a linearization of the observation model around the estimated point, assuming smooth behavior of the model. For this particular measurements, on the vehicle limits, these assumptions don't apply. The observation model function is not globally differentiable, in this cases it has no derivative, being this a problem for this method.

The solution proposed is to ensure that the observation model function stays in his differentiability domain, which means, in this case, that the filter only counts with measurements such that $\varphi_j^i \in [\chi(x_t, s_i) \cap \chi(\bar{x}_t, s_i)]$, this approach eliminates useful information from the filter, since it is neglecting some measurements.

For these implementation the sets $\chi(x_t, s_i)$ and $\chi(\bar{x}_t, s_i)$ must be computed on each EKF update step. $\chi(\bar{x}_t, s_i)$ is easy to achieve, since $\bar{x}_t$ is well known, but for the real pose, it is not so trivial, there is no absolute certain way to distinguish the measurements hitting the vehicle and the ones hitting the map. One way to guarantee the knowledge of this set with a certain degree of confidence is by implementing a virtual barrier around the map wall, and classify the measurements depending if the point measured is inside or outside the barrier polygon(3.28). Figure 3.7 illustrates this classification. The thickness of this barrier depends on the standard deviation for the error of laser range measurements σ_s^i, and on

the certainty degree of the barrier itself. As the range measurement error is assumed to be gaussian a barrier with thickness of $3\sigma_s^i$ will ensure that 99% of measurements would be classified correctly.
The objective with this sets is to minimize the impact of non-smoothness of observation model, including only measurements hitting both predicted and real vehicle body. The problem is not yet solved but it is minimized.

$$\begin{cases} \text{if } z_{cart}(s_i)_j \quad \text{inside barrier polygon}, & \varphi_j^i \in \overline{\chi(x_t, s_i)} \\ \text{if } z_{cart}(s_i)_j \quad \text{outside barrier polygon}, & \varphi_j^i \in \chi(x_t, s_i) \end{cases} \tag{3.28}$$

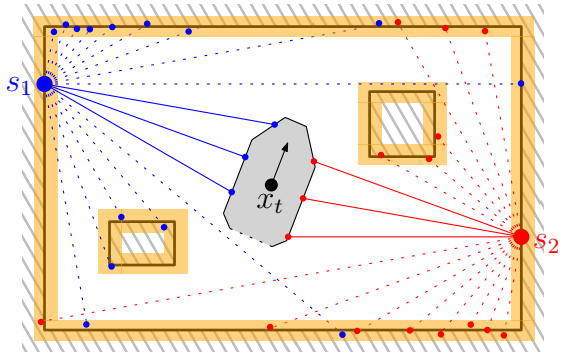


Figure 3.7: Barrier around map walls

To compute the crucial EKF update step, the kalman gain matrix K_t requires a linearization of the observation model function, the simplest way is using the jacobian H_t (3.29) around the current prediction, $\bar{x}_t$. Jacobian is, for each laser ray, the derivative of the distance measured with respect to vehicle pose (3.30). The derivative for each measurement such that $\varphi_j^i \in [\chi(x_t, s_i) \cap \chi(\bar{x}_t, s_i)]$, depend on vehicle pose, $\bar{x}_t$, sensor pose, s_i and direction, φ_j^i.
Although the measures include both range and angle, the jacobian only has derivatives on measured ranges because the angle is static, and gaussian error is only considered on range measurement and not on angular distances .

$$H_t = \left[\begin{array}{ccc} \frac{\partial h(x_t)}{\partial x_r^t} & \frac{\partial h(x_t)}{\partial y_r^t} & \frac{\partial h(x_t)}{\partial \theta_r^t} \end{array} \right]^T \Bigg|_{\overline{x}_t} \tag{3.29}$$

$$H_t(\varphi_j^i) = \frac{d}{dx_t} [1\ 0] \left[A(\varphi_j^i, p_{\hat{e}+1})^{-1} \left[p_{\hat{e}+1} - \begin{pmatrix} x_s^i \\ y_s^i \end{pmatrix} \right] \right] \Bigg|_{\overline{x}_t} \tag{3.30}$$

The implementation presented maps directly the influence of measures on the vehicle pose, there is not the intermediate step of feature extraction, like line extraction ([38], [25], [18]) or matching of the vehicle layout to measurements ([35]), since these approaches require some computational effort, and normally perform poorly when facing symmetry, which is typical in vehicle layouts.

The output of this filter should be a good approximation of the correct vehicle pose,x_t, returning for each time step t, a estimation $\hat{x}_t = [\hat{x}_r^t \ \ \hat{y}_r^t \ \ \hat{\theta}_r^t]^T$ and a covariance matrix, Σ_t representing the current belief. These matrix is a good indicator of the precision of the

method. Typically if variances are high, the filter has poor precision, if they are low means more precision.

3.2.3 Fault Detection, Global Localization and Pose Ambiguity

The only way to detect that prediction is wrong, going away from the real pose, is by measurement analyzes. Those are the only information available on-line. The number of measurements integrated by KF is a good evaluator of the similarity between sensor real and estimated poses.

To reach a numerical value for this similarity, it is defined a similarity ratio, *sml*, in 3.31, where # defines the number of elements of the set, i.e., $\#\chi(x_t, s_i)$, define the number of laser rays falling outside the barrier. *sml* is a number between 0 and 1, being 0 for total dissimilarity and 1 for complete similar. Figures 3.8(a) and 3.8(b) illustrates the principle, the idea is to have an on-line certainty evaluator, this way the method can implement heuristics to enhance his performance, like enlarging the covariance matrix, or restart the method with a new initial belief.

$$sml = \frac{2\#\left[\chi(x_t, s_i) \cap \chi(\bar{x}_t, s_i)\right]}{\#\chi(x_t, s_i) + \#\chi(\bar{x}_t, s_i)} \tag{3.31}$$

$$cm = \frac{1}{\#\chi(x_t, s_i)} \sum \left[z_{cart}(s_i)_j\right]^T, \qquad for \;\; \varphi_j^i \in \chi(x_t, s_i) \tag{3.32}$$

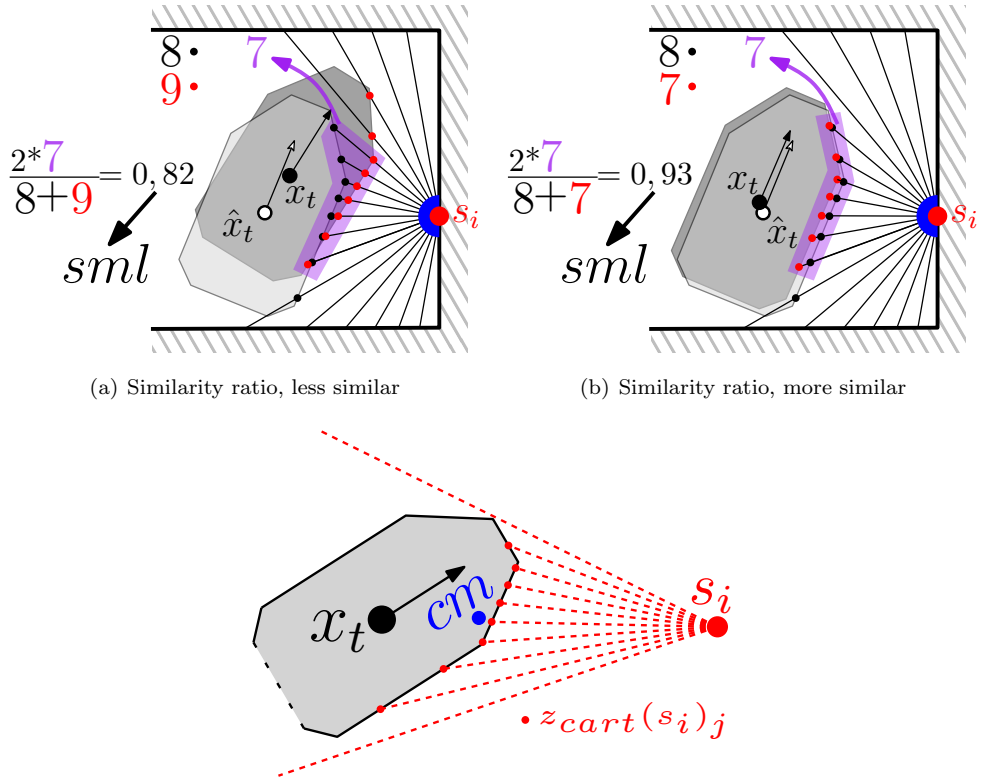
One possible use for this feature is the global localization, on a kidnapping situation, in this case this is equivalent to a situation where the system fails and loses track of the vehicle. With these method the algorithm perceives the vehicle loss and generates a new belief centered in the most probable position for the vehicle.

The most probable position is where the sensors are known to be hitting something rather than the wall. Using the barrier described in section 3.2.2, a position $cm = [cm_x \;\; cm_y]^T$ is computed based on the measures which angular positions $\varphi_j^j \in \chi(x_t, s_i)$ (3.32). The filter is restarted, with $\hat{x}$ position equal tocm with a random orientation, and with a predefined, sufficiently large, covariance matrix, that should be defined considering vehicle's dimensions. The point *cm* has a high probability to fall inside the vehicle layout, (Figure 3.8(c)), and so, if the covariance ellipse include the vehicle center, the method has big chances to converge based on the observation models only.

During EKF operation, with certain kind of vehicle layout, there is a possibility of pose ambiguity, due to symmetries. For example with a rectangular vehicle, like TCS, there is a 180° ambiguity, the only way to distinguish which is the front and rear of the vehicle is by dynamics integrations. This is a hard problem, because the error given by the dynamics in each iteration is absorbed by the noise random variable, w_t.

EKF belief is modeled with a gaussian, there is no way to have multi-modal distributions. Once the algorithm locks with one orientation it is stuck with it, since the movement model noise absorb the error caused by dynamics, and from the sensors point of view, due to symmetry, the measurements are correct.

This problem was not solved in the EKF framework, but in section 3.3.3 some heuristics are proposed to solve it.



(a) Similarity ratio, less similar

(b) Similarity ratio, more similar

(c) High probability pose, *cm*

Figure 3.8: Global localization with EKF

3.3 Particle Filter

3.3.1 Introduction

PF is, like EKF, a bayesian state estimation method, proposed, the first time, in [12]. This method is comparable with EKF, since it is a discrete-time estimator, it work recursively and has the same two steps, prediction and update. The major difference is in belief representation. Beliefs, at time t, are no longer represented by gaussian moments, but instead, with a set of weighted particles Ψ_t (3.33), each one is a hypothetical realization, $x_t^{[n]} = [x_r^{[n]} \; y_r^{[n]} \; \theta_r^{[n]}]^T$ of the state to estimate, x_t. These way there is no assumption of gaussian processes, and both transition and observation models can be non-linear and non-gaussian.

The prior belief is represented, at time t, by the set $\overline{\Psi}_t$ (3.35) it is composed by N_p particles. The method computes belief $bel(x_t)$ recursively from $bel(x_{t-1})$, so for the representation with particle sets, the method estimates Ψ_t from Ψ_{t-1}.

For each iteration, the prediction step is done using a transition model, this model maps the effect of control inputs on each particle state $x_{t-1}^{[n]}$. From previous belief Ψ_{t-1} (3.34) and control inputs (u_t), it generates the prior belief particle set $\overline{\Psi}_t$. During update step, the algorithm uses measurement model and computes the weights of each particle, $w_t^{[n]}$. This weight (3.37) is the likelihood of measurements predicted for state $x_t^{[n]}$ given the real measurements z_t. This weighted set of particles, Ψ_t represent the posterior belief (3.36 since

by bayesian inference and markov assumption the posterior belief is related to prior belief by 3.38 . Intuitively it is expected that the particles with greater likelihood are the ones closer to correct state.

$$\Psi_t = \left\{ x_t^{[1]},\ x_t^{[2]},\ \cdots,x_t^{[N_p]} \right\} \tag{3.33}$$

$$bel(x_{t-1}) = p(x_{t-1}|z_{1:t-1}, u_{1:t-1}) \sim \Psi_{t-1} \tag{3.34}$$

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}) \sim \overline{\Psi}_t \tag{3.35}$$

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t}) \sim \Psi_t \tag{3.36}$$

$$w_t^{[n]} = p(z_t|x_t^{[n]}) \tag{3.37}$$

$$bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t) = \eta p(z_t|x_t)p(x_t|z_{t-1}, u_t) \tag{3.38}$$

The algorithm used by PF is described in Table 3.2, the resample step is the main introduction with respect to EKF, this step samples a new set Ψ_t from the set $\overline{\Psi}_t$, it maps the particle weights into their spacial distribution. It is like sampling particles from $bel(x_t)$ 3.38, the weights are reseted, and a new iteration begins from the last spacial distribution. Resampling forces particles' states to be close to the real one, otherwise, particles would start to go away from real state. The weights should accumulate along the various iterations 3.39, and the majority of particles would have very low weights, without resampling the method need much more particles and normally performs poorly.

$$w_t^{[n]} = p(z_t|x_t^{[n]})w_{t-1}^{[n]} \tag{3.39}$$

Table 3.2: PF algorithm

1. Initialize with current belief at time $t = 0$, Ψ_0
2. For each time step t, $t = t + 1$, new commands u_t and measurements z_t
   (a) Prediction step, for each particle n:
   Sample $x_t^{[n]} \sim p(x_t|z_{t-1}, u_t)$
   (b) Update step, for each particle n:
   Weight attribution $w_t^{[n]} \propto p(z_t|x_t^{[n]})$
   (c) Normalize weight distribution: $\sum_{i=1}^{N_p} w_t^{[n]} = 1$
   (d) Resample: Sample $x_t^{[n]} \sim \eta p(z_t|x_t)p(x_t|z_{t-1}, u_t)$, forming Ψ_t

An application of PF to vehicle localization, is the MCL method, ([9], [15]). In MCL algorithm the prediction step depends on a transition model, $f(x_t, u_t)$ that, for each particle state, $x_{t-1}^{[n]}$ will compute his new state based in commands, u_t, (3.40), since it is applied to a mobile vehicle, in this situation, it is also called movement model. In this context, the states predicted for particles representing the prior belief $\overline{\Psi}_t$, are addressed as $\bar{x}_t^{[n]}$, being $\overline{\Psi}_t = \left\{ \bar{x}_t^{[1]},\ \bar{x}_t^{[2]},\ \cdots,\bar{x}_t^{[N_p]} \right\}$.
The update step uses a sensor model, or observation model, to compute the measurement

likelihood $p(z_t|x_t^{[n]})$. This observation model $h(x_t)$ (3.41) is used to attribute the weights for each particle.

$$\bar{x}_t^{[n]} = f(x_{t-1}^{[n]}, u_t) \tag{3.40}$$

$$w_t^{[n]} = h(\bar{x}_t^{[n]}) \tag{3.41}$$

The resample step is done in a particular way, it uses the current particle states, drawing no new states. This method, in fact, clones several times the same particle to the next distribution. The probability of drawing a particle in the next distribution is proportional to his weight. This can be done because in next prediction step, the error added to movement prediction has an independent random value, so it will separate particles states, keeping them in the same space region. This method has no gaussian assumptions or linearizations, this can be an advantage since EKF had some problems with non-smoothness and non-linearity of the observation model function. MCL algorithm is described in Table 3.3.

Table 3.3: MCL algorithm

1. Initialize with current belief at time $t = 0$, Ψ_0
2. For each time step t, $t = t + 1$, new commands u_t and measurements z_t
   (a) Prediction step, for each particle n:
   $\bar{x}_t^{[n]} = f(x_{t-1}^{[n]}, u_t)$
   (b) Update step, for each particle n:
   $w_t^{[n]} = h(\bar{x}_t^{[n]})$
   (c) Normalize weight distribution: $\sum_{i=1}^{N_p} w_t^{[n]} = 1$
   (d) Resample: Draw particle n with probability $w_t^{[n]}$

3.3.2 Proposed method

Movement model

The movement model $f(x_t, u_t)$ is very similar to the one described in 3.2.2 in page 48 and some parameters used here are defined there. The main differences is that now it must be computed for each particle separately and the noise, associated to the process, must be added on the prediction.

The model makes a discrete integration of the center velocities, derived from the vehicle kinematic model. Now the center velocity is different for each particle, $g_t^{[n]}$ (3.42), since it depends on the heading of the prediction. Process noise is modeled by a zero mean gaussian with covariance matrix Q_t. Let us define a random vector $err_t^{[n]}$ drawn from $\mathcal{N}(0, Q_t)$ for each particle n, at time t.

$$g_t^{[n]} = \begin{bmatrix} v_x^t \\ v_y^t \\ v_\theta^t \end{bmatrix} = \begin{bmatrix} v_R^t \cos(\theta_r^{t\,[n]} + \alpha_R^t) + v_F^t \cos(\theta_r^{t\,[n]} + \alpha_F^t) \\ v_R^t \sin(\theta_r^{t\,[n]} + \alpha_R^t) + v_F^t \sin(\theta_r^{t\,[n]} + \alpha_F^t) \\ \frac{1}{bW}\left[-v_R^t \sin(\alpha_R^t) + v_F^t \sin(\alpha_F^t)\right] \end{bmatrix} \tag{3.42}$$

$$\bar{x}_t^{[n]} = x_{t-1}^{[n]} + \Delta T\, g_t^{[n]} + err_t^{[n]}, \qquad Q_t = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_\theta^2 \end{bmatrix} \tag{3.43}$$

This approach moves each particle state, respecting input u_t, with independent random error. This is useful to keep a good particle distribution in the prior belief and to separate cloned particle from last iteration. In resample step some particle states are cloned, and the addition of independent error, makes each particle state unique.
The error is assumed to be gaussian on the vehicle pose to estimate, x_t, this assumption can have some impact on the the method's performance, since the errors are actually in the wheels heading and velocities. During this work there is not a carefully inspection of the movement error characteristics for this kinematic configuration, it might help the model for a real world implementation, but, for now, the error in vehicle pose is assumed to absorb all the possible errors in movement.

Observation model

PF uses the observation model to attribute weights for each particle, observation model is a likelihood function, $h(\bar{x}_t^{[n]}) = p(z_t|\bar{x}_t^{[n]})$, it evaluates, for each particle, the resemblance between measurement coming from sensors, and predicted for particle.
The measurements coming from real sensors Z are described in section 3.1 in page 43, the predictions for these measurements are computed individually for each particle. Using a similar method to the one described in 3.25, the main difference is that, now there is no restriction on the model, and the method predicts all possible measurements, $\varphi_j^i \in \chi() \cup \overline{\chi()}$, the predicted range is given by $\bar{d}_j^{i\,[n]}$ for every angular position, φ_j^i, making $y_i(x_t^{[n]})$ (3.44) the same length as $z(s_i)$ (3.7).
Intuitively the $y_i(x_t^{[n]})$ is the sensor, s_i, observation if the vehicle was in pose $x_t^{[n]}$, comparing $y_i(x_t^{[n]})$ with $z_i(x_t)$ for all sensors, will give a good evaluator if particle n state is close to the true state. The closer the state is, the higher the measurements' likelihood and the higher is the particle weight $w_t^{[n]}$.

$$y_i(\bar{x}_t^{[n]}) = \begin{bmatrix} \bar{d}_1^{i\,[n]} & \cdots & \bar{d}_{P_i}^{i\,[n]} \\ \varphi_1^i & \cdots & \varphi_{P_i}^i \end{bmatrix} \tag{3.44}$$

$$w_j^{i\,[n]} = p(d_j^i|\bar{x}_t^{[n]}) = \frac{1}{\sqrt{2\pi\sigma_{range}^2}} e^{\frac{-(d_j^i - \bar{d}_j^{i\,[n]})^2}{2\sigma_{range}^2}} \tag{3.45}$$

Laser range finder sensors have a gaussian error on ranges measured, so a way to compute the likelihood of a measurement prediction ($w_j^{i\,[n]}$) is a 1D gaussian, with variance σ_{range}^2 and mean $\bar{d}_j^{i\,[n]}$. The likelihood of this measurement is given by 3.45
The weight of a particle ($w_t^{[n]}$), is proportional to the likelihood $p(Z|\bar{x}_t^{[n]})$, the sum of all particle weights must be normalized to sum one (3.47), this way, the set of weights are a discrete probability distribution. To obtain this distribution, each measurement is assumed to be an independent random variable, so the joint probability is given by 3.46. Figure

3.9 shows the appliance of this method, retrieving weights for several measures hitting the vehicle, for map walls, the procedure is the same.

$$w_t^{[n]} = p(Z|\bar{x}_t^{[n]}) = \eta \prod_{i=1}^{L} \prod_{j=1}^{P_i} p(d_j^i|\bar{x}_t^{[n]}) \tag{3.46}$$

$$w_t^{[n]} = \frac{w_t^{[n]}}{\sum_{n=1}^{N_p} w_t^{[n]}} \tag{3.47}$$

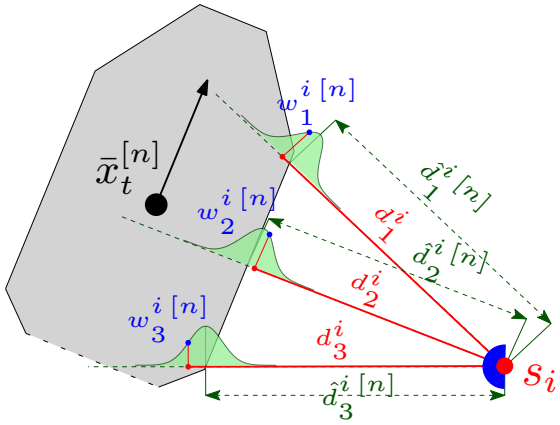


Figure 3.9: Gaussian likelihood function appliance

This approach to compute the likelihood has some problems, since the observation concept is different from a normal situation. Normally sensor is on board, and measurements change smoothly with the vehicle movement. On this approach, there are observations of the vehicle and observations of the map on the same reading, which has a non-smooth behavior. For a very small change of vehicle pose, measurements can have abrupt changes, as illustrated in Figure 3.6, where a small difference between x_t and $\bar{x}_t$, have a major impact on some ranges. The use of gaussian function (3.45),will result in a very low likelihood for these measurements and consequently, since we use 3.46 to estimate the particle weight, a very low weight to the respective particle. This situation is similar to have many particles around the correct pose, but their likelihood is very low, obviously the simple gaussian function is not suited for this situation, or else very few particles would have a sufficient weight to survive the resample step, making the PF perform very poorly.

The idea to overcome this problem is the use of a new likelihood function, instead of a gaussian centered in predicted range. This function will have in account this discontinuity and will smooth it. To implement it, classifying the measurement type is crucial.

Returning the previous set notation from Kalman filter implementation, $\chi(x_t, s_i)$ is a set of angular positions, such that laser rays casted in those directions, from sensor s_i, hit the vehicle with pose x_t. $\overline{\chi(x_t, s_i)}$ is the set of angular positions such that rays do not hit the vehicle. Applied to PF, there are several of these sets, for each particle predicted state ($\chi(\bar{x}_t^{[n]}, s_i)$ and $\overline{\chi(\bar{x}_t^{[n]}, s_i)}$, and for real pose ($\chi(x_t, s_i)$ and $\overline{\chi(x_t, s_i)}$). Section 3.2.2 explains how to obtain these sets,for predicted particle poses, the same approach used with kalman predicted pose. There is no need for compute the sets $\chi(x_t, s_i)$ and $\overline{\chi(x_t, s_i)}$, this effort is avoided by introducing the new likelihood function.

The likelihood function for each ray (3.48) is a weighted sum of several likelihood functions and it depends on which set the ray's angular position belongs to.

- $\varphi_j^i \in \chi(\bar{x}_t^{[n]}, s_i)$: In the predicted state, $\bar{x}_t^{[n]}$, the measurements with these angular positions hit the vehicle. But in real state, x_t, those same rays can hit or miss it, since the correct pose is unknown:
  - $\mathcal{L}_1(d_j^i) \longrightarrow \varphi_j^i \in \chi(\bar{x}_t^{[n]}, s_i) \cap \chi(x_t, s_i)$:for this angular position φ_j^i the ray hits the vehicle in predicted and in reality. It is implemented a gaussian likelihood function like the one before, 3.45 with σ^2_{range} variance and mean $\bar{d}_j^{i\,[n]}$;
  - $\mathcal{L}_2(d_j^i) \longrightarrow \varphi_j^i \in \chi(\bar{x}_t^{[n]}, s_i) \backslash \chi(x_t, s_i)$: for this angular position φ_j^i the ray hits the vehicle in predicted pose but in reality hits the wall behind it. Since the sensor is fixed, the range to the wall behind is a constant, D_j^i, we can use this information and implement a gaussian likelihood function, 3.45 with σ^2_{range} variance and mean D_j^i, this accounts for a certain probability that some measures miss the vehicle and hit the wall.
- $\varphi_j^i \in \overline{\chi(\bar{x}_t^{[n]}, s_i)}$: In the predicted state, $\bar{x}_t^{[n]}$, the measurements with these angular positions hit the map walls:
  - $\mathcal{L}_3(d_j^i) \longrightarrow \varphi_j^i \in \chi(x_t, s_i) \backslash \chi(\bar{x}_t^{[n]}, s_i)$, for this angular position φ_j^i the ray hits a map wall in predicted pose but in reality hits the vehicle. This case is similar to $\varphi_j^i \in \chi(\bar{x}_t^{[n]}, s_i) \backslash \chi(x_t, s_i)$, but here the distance to vehicle is unknown. We just know that there is a probability for the ray to be blocked between the sensor and the wall. This is modeled by a uniform distribution between range=0 and range=D_j^i;
  - $\mathcal{L}_4(d_j^i) \longrightarrow \varphi_j^i \in \overline{\chi(\bar{x}_t^{[n]}, s_i)} \cap \overline{\chi(x_t, s_i)}$, for this angular position φ_j^i the ray hits a map wall in predicted pose and reality. This is modeled with a gaussian with σ^2_{range} variance and mean D_j^i.
- $\mathcal{L}_5(d_j^i) \longrightarrow$ for all rays there is a probability of erroneous measurements, problems with sensor hardware or communication, to account for this it can be added a uniform distribution from range=0 to maximum range.

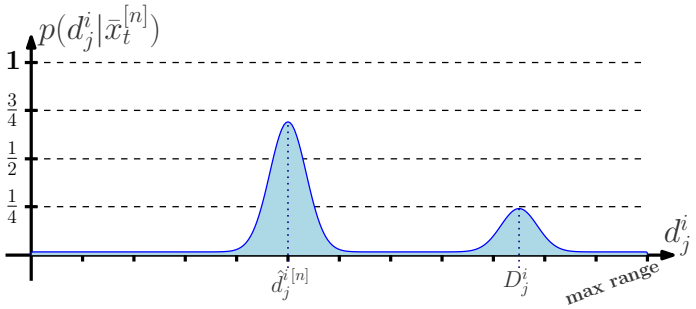
$$\mathcal{L}_1(d_j^i) = \frac{1}{\sqrt{2\pi\sigma^2_{range}}} e^{\frac{-(d_j^i - \bar{d}_j^{i\,[n]})^2}{2\sigma^2_{range}}} \qquad \mathcal{L}_2(d_j^i) = \frac{1}{\sqrt{2\pi\sigma^2_{range}}} e^{\frac{-(d_j^i - D_j^{i\,[n]})^2}{2\sigma^2_{range}}}$$

$$\mathcal{L}_3(d_j^i) = \frac{1}{D_j^i} \qquad \mathcal{L}_4(d_j^i) = \frac{1}{\sqrt{2\pi\sigma^2_{range}}} e^{\frac{-(d_j^i - D_j^{i\,[n]})^2}{2\sigma^2_{range}}} \qquad \mathcal{L}_5(d_j^i) = \frac{1}{max\ range}$$

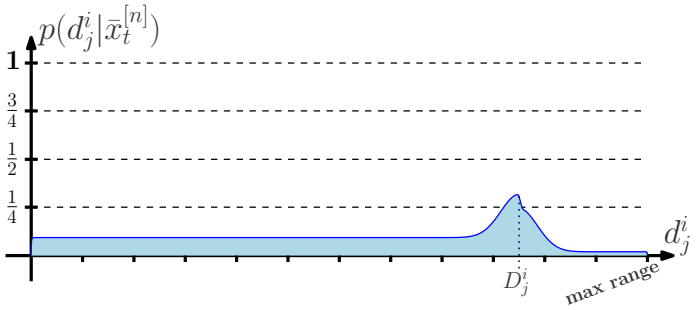
$$w_j^{i\,[n]} = p(d_j^i | \bar{x}_t^{[n]}) = \begin{cases} a_V \mathcal{L}_1(d_j^i) + a_w \mathcal{L}_2(d_j^i) + a_{err} \mathcal{L}_5(d_j^i) & \text{if} \quad \varphi_j^i \in \chi(\bar{x}_t^{[n]}, s_i) \\ \\ a_v \mathcal{L}_3(d_j^i) + a_W \mathcal{L}_4(d_j^i) + a_{err} \mathcal{L}_5(d_j^i) & \text{if} \quad \varphi_j^i \in \overline{\chi(\bar{x}_t^{[n]}, s_i)} \end{cases}$$

$$a_V + a_w + a_{err} = 1 \qquad a_w + a_W + a_{err} = 1 \tag{3.48}$$

The two branches of $p(d_j^i|\bar{x}_t^{[n]})$ function, are drawn in Figure 3.10, and their appliance is shown in Figure 3.11 similarly to gaussian model in Figure 3.9.
The weighting of these likelihood function must respect some rules, the first is that the weights must sum up one, this is to keep $p(d_j^i|\bar{x}_t^{[n]})$ as a probability distribution over d_j^i, the other is the adjustment of the weights. To fit the propose of returning a higher likelihood for particles closer to real state, a_V must be higher than a_w, this returns a higher value for closer poses than for erroneous ones, but at the same time smooths the weighting, giving a reasonable weight for poses slightly different.



(a) Likelihood for $\varphi_j^i \in \chi(\bar{x}_t^{[n]}, s_i)$



(b) Likelihood for $\varphi_j^i \in \overline{\chi(\bar{x}_t^{[n]}, s_i)}$

Figure 3.10: Likelihood functions for PF

$$w_t^{[n]} = p(Z|\bar{x}_t^{[n]}) = \eta \prod_{i=1}^{L} \prod_{j=1}^{P_i} w_j^{i\,[n]} \tag{3.49}$$

$$\log w_t^{*[n]} = \sum_{i=1}^{L} \sum_{j=1}^{P_i} \log w_j^{i\,[n]} \tag{3.50}$$

$$w_t^{[n]} = \frac{\exp\left(\log w_t^{*[n]} - lw_{min}\right)}{\sum_n^{N_p} \exp\left(\log w_t^{*[n]} - lw_{min}\right)}, \qquad lw_{min} = \min\left\{\log w_t^{*[1]}, \cdots w_t^{*[N_p]}\right\} \tag{3.51}$$

To weight the particles, the same method is used, assuming that measurements are independent (3.49). Given the great amount of measurements, the operation 3.49 is hard to compute given the limit of floating point number. To solve this problem, instead of computing the likelihood function it is computed his logarithm (3.50). Let $w_t^{*[n]}$ denote the

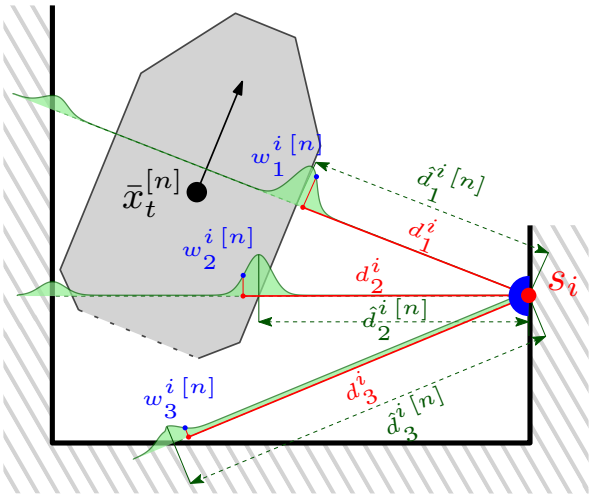


Figure 3.11: PF Likelihood functions appliance

unnormalized weight of particle n, in normalization, a mathematical manipulation is done to avoid reaching the numerical floating point limits (3.51).

Observation model establishes a likelihood for each particle state, this way the algorithm can have a posteriori belief over the state space, composed by a distribution of weighted particles.

Notice that with the introduction of this non-gaussian, non-linear likelihood functions, the filter avoids the use of barrier to compute sets $\chi(x_t, s_i)$ and $\overline{\chi(x_t, s_i)}$, which was a time consuming operation. It also uses all the information from sensors, the measures hitting the vehicle and hitting the walls, this is an enormous advantage of PF, that might enhance the estimation quality.

The reason for the expected enhancement with PF is that, now the system gathers information from rays that hit the vehicle, like EKF, but also from the ones hitting the walls, this last is relevant, because if a ray is passing and hitting a wall, it means the vehicle is not blocking that laser ray, so we now have information on where the vehicle is and where it is not, rather than just where it is.

Resample

The resample step is a very important part for the PF, it manages to keep a particle set around the most likely states, avoiding the degeneracy problem, i.e. all particle weights but one are closer to zero. It basically samples a set of particles from the current posteriori distribution, giving them the same weight. After this step the belief is no longer represented by a set of weighted particles, but just with a set of particles with a certain distribution over the state space. Resample step maps the weights into space distribution.

The resample process operates over the set Φ_t^- turning it into Φ_t^+, both sets represent the posterior belief at time t. The resample is done by cloning particles from Ψ_t^-, so after the process, Ψ_t^+ can have multiple particles with the same state. This seems illogic, but in fact is done to save processing work, it relies on movement model to add random noise during

the prediction step, so the particles with the same state in Ψ_t^+ will have unique states in $\bar{\Psi}_t$. Some methods to resample from a discrete distribution are described in [5], [34], [27], the implemented method is the stochastic sampling with replacement (roulette wheel selection) where the particles are selected to be cloned with probability $w_t^{[n]}$. This is a simple method with easy comprehension and implementation. The basic idea is to make a cumulative sum of the weights Cum_k (3.52) and then generate N_p random values, uniformly distributed from 0 to 1, uni_n, criteria to clone a particle from Φ_t^- to Φ_t^+ is shown in 3.53.

$$Cum_k = \sum_{n=1}^{k} w_t^{[n]} \qquad Cum_0 = 0 \tag{3.52}$$

$$\Psi_t^- \ni x_t^{[k]} = x_t^{[n]} \in \Psi_t^+ \;\; if \;\; Cum_{k-1} < uni_n \leq Cum_k \tag{3.53}$$

To enhance the performance of PF, there are some methods to avoid resampling in every iteration, the idea, apart from saving computational effort, is that a little spreading of particle states helps to have a reliable estimation, resampling in every iteration does not allow the particle states to go away from each other, and in some situation if the estimation is not very exactly, it can even loose track of the true state. Intuitively, if the method keeps all the particles in nearly the same state, there is no point in having various particles.
After some iterations passed from last resample, most of the particles have drifted away, their weights became very small, irrelevant to the distribution. When a certain percentage of particles are irrelevant to the distribution the method should resample again. To decide when to to do it, two values are proposed in [20], the coefficient of variation cv_t^2 (see 3.54) and effective sample size, ESS_t, (see 3.55), the approach is to resample whenever ESS_t drops below a threshold, usually a percentage of N_p.

$$cv_t^2 = \frac{var\left(w_t^{[n]}\right)}{E^2\left(w_t^{[n]}\right)} = \frac{1}{N_p}\sum_{n=1}^{N_p}\left(N_p w_t^{[n]} - 1\right)^2 \tag{3.54}$$

$$ESS_t = \frac{1}{1 + cv_t^2} \tag{3.55}$$

Estimation

The filter belief is represented by a set of particles, but, from this set, what is the best way to decide where the vehicle really is?
There are several ways to compute an estimation, $\hat{x}_t$, about the vehicle true pose x_t, from the set of weighted particles:

- **Maximum weight** – The particle with higher likelihood is considered the correct pose, 3.56. This approach is easy to compute bu has a lot of oscillation, as the particles move side by side with the real pose, not always the same particle has the maximum weight, the estimation can jump around the real pose, making the estimation accurate but not precise.

$$\hat{x}_t = x_t^{[n]}, \qquad \max_n w_t^{[1]} \;\cdots\; w_t^{[n]} \tag{3.56}$$

- **Weighted mean** – These estimation takes in consideration all particle states and weights, it computes a weighted mean from all particles (3.57). This is good for the

oscillation, it works like a low pass filter, but should be used only when the certainty is high and all particles are together. It is dangerous if the distribution can turn multi-modal, if half the particles are in one position and the other half somewhere else, the mean between them can give an estimation in the middle which do not make sense.

$$\hat{x}_t = \sum_{1}^{N_p} w_t^{[n]} x_t^{[n]} \tag{3.57}$$

- **Weighted mean around the maximum** – This approach avoids the problems from the previous two, it computes a weighted mean but only with particles in the vicinity of the maximum likelihood particle, $x_t^{[n]max}$ (3.58). It smooths the oscillations and does not have the risk of making mean with particles far away.

$$\hat{x}_t = \frac{\sum w_t^{[n]} x_t^{[n]}}{\sum w_t^{[n]}} \qquad if, \;\; ||x_t^{[n]} - x_t^{[n]max}|| < vicinity\, radius \tag{3.58}$$

3.3.3 Fault Detection, Global Localization and Pose Ambiguity

Similar to EKF, the PF can only rely on measurements to perceive if the estimated pose $\hat{x}_t$ is near the true one, x_t.

Unlike the EKF, here, a likelihood value is computed for each particle state, in order to attribute weights to particles, $w_t^{[n]}$. These likelihood values are greater if the particle state, $x_t^{[n]}$ is near x_t. The approach to know if the pose is correct is to compute a mean value of all likelihoods,$w_t^{*[n]}$, before the normalization procedure (3.59). In fact given the huge quantity of measurements on each iteration, the likelihood is a very good evaluator, it immediately drops when the prediction drives away from true pose.

These indicator, $Like_t$ is computed in logarithmic scale, to keep the numerical limits of floats, and even in this scale the drop in likelihood is abrupt when the estimation is wrong. This indicator can be used to reinitialize the belief when it loses track of the vehicle. In our context it is used in the kidnap situation explained in section . When the likelihood drops, a probability p_{cm}, proportional to the likelihood drop is added to the resample step. This means that, every time the likelihood drops there is a probability to generate particles in cm position. p_{cm} is defined in 3.60, it is computed based on a ratio between the current likelihood value, $like_t$ and a threshold, $like_{thresh}$ predefined by the typical values presented by the likelihood in normal tracking conditions. cm is the mass center of measures hitting the vehicle, here the PF must use the barrier to decide which measurements enter to the calculus of cm, (Eq. 3.32). The behavior of this method is illustrated in Figure 3.12, in red, the magnitude of the error (3.61) and in blue, the likelihood. It is visible where the kidnapping occurred, the likelihood immediately drops and particles star to migrate to cm, after a few iterations the estimation is on the true pose again.

$$Like_t = \frac{1}{N_p} \sum_{n=1}^{N_p} \log w_t^{*[n]} \tag{3.59}$$

$$p_{cm} = \max\left\{0, 1 - \frac{like_{thresh}}{like_t}\right\} \tag{3.60}$$

$$||l_{err}||(t) = \sqrt{(\hat{x}_r^t - x_r^t)^2 + (\hat{y}_r^t - x_r^t)^2} \tag{3.61}$$

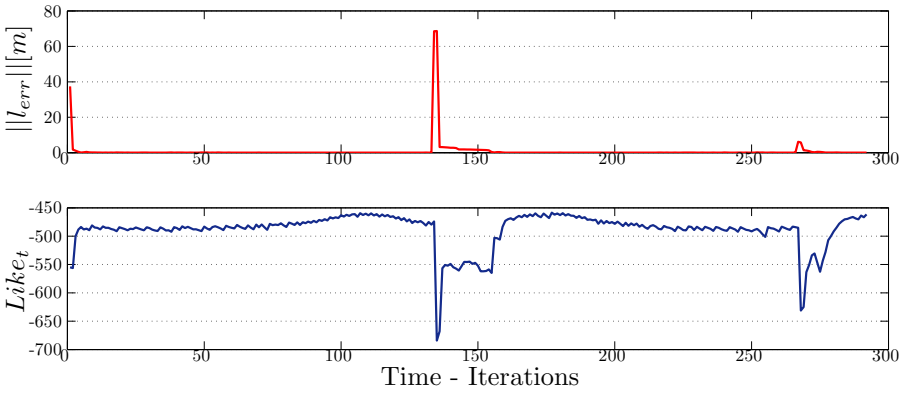


Figure 3.12: Error magnitude $||l_{err}||(t)$ and likelihood $like_t$

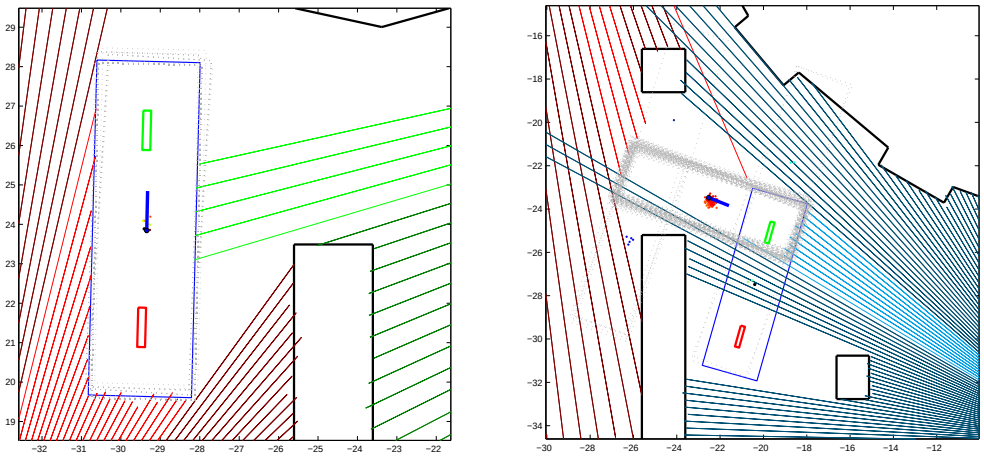
The geometric ambiguities are a problem due to vehicle symmetry, the laser measurements can be very similar in several poses of the vehicle, the algorithm can converge to local maximums in likelihood function instead of converging to the global maximum. With PF, the geometric ambiguities are solved with a set of heuristics specifically created for the TCS vehicle. The heuristics idea is to use the layout geometric information on the filter. There are two distinct situations:

- **180° ambiguity** – Figure 3.13(a) – The correct pose in this situation can only be achieved if the vehicle is moving, by velocity integration. If it is stopped the measurements are exactly the same for poses with 180° orientation offset.
  So if the vehicle is moving there is a probability that the estimated pose is mistaken in 180°, p_{180}. Using the multi-modal distribution capability of PF, on the resample step, there is p_{180} probability of generating particles with 180° orientation offset from the current estimation, $\hat{x}_t$. If those particles have a superior likelihood, then, particles start to migrate more for that mode, and with some iterations all particles are in the correct direction. p_{180} is a fixed value, and in every resample step, if the vehicle is moving, there is this probability of generating particles with inverse direction, even is the correct pose is correct.

- **corner ambiguity** – Figure 3.13(b) – Corner ambiguity only happens when the vehicle is stopped, but it is also a very important situation, because if there is a system failure, the vehicle stops, and the localization algorithm must know his correct pose before it starts moving again.
  To solve this ambiguity, when the vehicle is stopped a probability is inserted on the resample step that the prediction is wrong, so this is the probability of generating particles on the ambiguity poses. The working principle is equal to 180 ambiguity but happens when the vehicle is stopped, and instead of an 180° offset the particles are generated in different poses (3.13(b)).

With multi-modal distribution, PF enables the use of various heuristics like these that are easy to implement and solve the problem with robustness. This is an advantage over the

Kalman that has only unimodal gaussian representation.



(a) 180° ambiguity

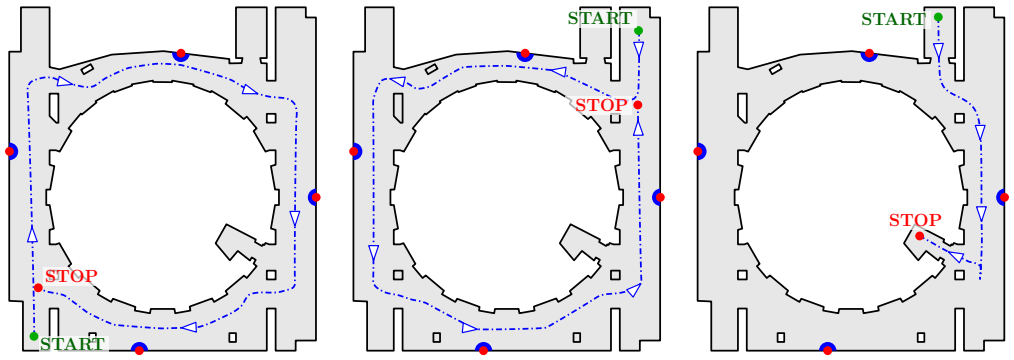
(b) corner ambiguity

Figure 3.13: Geometric symmetry ambiguities

3.4 Experimental Results

The methods were tested in a Tokamak building floor, like the one used on Chapter 2, but for these experiments one VV door is open, like shown in map layouts on Figures 3.14. Tests were performed for three different vehicle trajectories, one traveling the corridors in CW direction 3.14(a), other in CCW, 3.14(b) and one real trajectory used to go from elevator to VV 3.14(c). The designation for these trajectories, from now on, is *CWtraj*, *CCWtraj* and *VVtraj*, respectively. Notice that these are trajectories for the vehicle wheels and not for his center, and during the trials the localization methods estimate the pose of the vehicle's center, giving his position and orientation.

The vehicle used is the TCS, it has a rectangular layout as shown in Figure 3.15 that



(a) CWtraj (b) CCWtraj (c) VVtraj

Figure 3.14: Map layout and trajectories used to test localization methods

corresponds to V_0 layout (3.62), where L_{TCS} is the length and W_{TCS} the width of the vehicle.

The principal issue with this layout is the symmetry, from LRF sensor point of view it is difficult to distinguish between corners. This leads to an ambiguity in pose estimation, the dynamics integration with sensor measurements tend to help with this problem, but when the vehicle stops there are no velocities to integrate and it is impossible to distinguish a 180° orientation error. For this ambiguities we must rely on specific heuristics explained in section 3.3.3.

$$V_0 = \begin{bmatrix} -L_{TCS}/2 & L_{TCS}/2 & L_{TCS}/2 & -L_{TCS}/2 \\ -W_{TCS}/2 & -W_{TCS}/2 & W_{TCS}/2 & W_{TCS}/2 \end{bmatrix}, \tag{3.62}$$

The sensory network layout is shown on Figure 3.14, these sensor placement results from the optimization, in Chapter 2, for a network with four sensors, $L = 4$. The reason to choose $L = 4$ is the trade of between coverage, around 97 %, and computational effort. Both algorithms perform almost in real time, simplifying the experiments. Also it has almost no redundancy, this way, the robustness of each method is tested, it must hold with measurements from only one sensor most of the time.

Both methods run in a developed MATLAB simulation environment (Figure 3.16), where all system's parts are combined, vehicle dynamics, LRF readings and localization algorithms.

To execute these experiments some parameters were carefully chosen, concerning each part

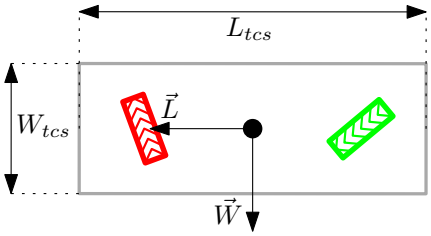


Figure 3.15: TCS vehicle layout and coordinate system

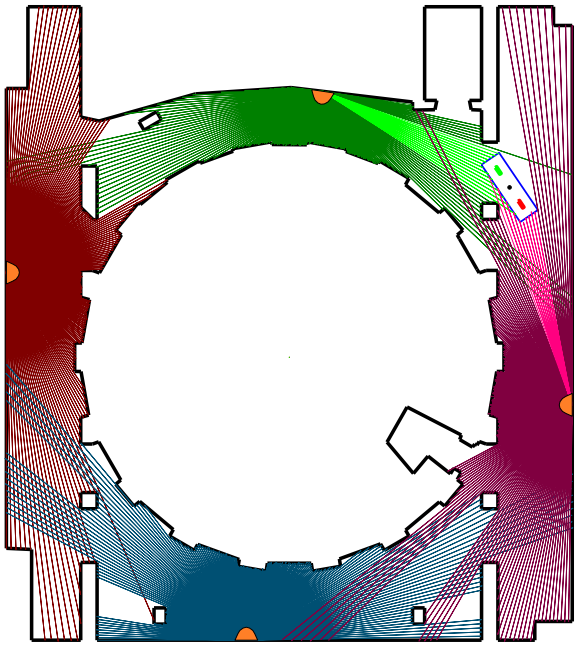


Figure 3.16: MATLAB simulation environment

of the simulation model.

- Vehicle dynamics – The control of the vehicle is outside the scope of these thesis, and is assumed to be perfect. The commands of velocity given to the vehicle are respected in simulation. The noise is only considered in the vehicle position and not on velocities. The time iteration Δt is set to 0.3 s, from now on the iteration number is also defined by t for simplicity.

- LRF – The range measurements from sensors are corrupted with zero mean gaussian noise around the true range. This gaussian is modeled by the standard deviation, σ_s^i, assumed to be 0.1 m, for all sensors. These is much superior than typical values for LRF sensors [40]. It is overestimated, since the model does not include other laser measurement problems, like outliers, or erroneous measurements due to angles of incidence or hitting surface proprieties. The angular resolution, δ_s^i, is set to 1° for all sensors.

- Localization system – The localization system is the crucial part, it accounts for all possible error in the system, the assumed standard deviation for sensor measurements,

σ_{range}, is set to 0.5 m, to overcome every possible error.
For the position error it is assumed to have an zero mean gaussian error in every pose component, the covariance matrix, $Q_t = diag(0.01m \quad 0.01m \quad 1°)$. For PF implementation,the number of particles, N_p, is fixed to 100.

This section compares methods through this framework, with this sensory network, the impact of sensor placement in localization performance is studied only on Chapter 4.

Comparison is based on a set of defined criteria that are required to have a suitable localization algorithm. The motivation for this work is the operation on remote handling operations in ITER, so these criteria are very important given the restricted tolerances inside those buildings.

- **Accuracy** and **Precision** – Predicted pose $\hat{x}_t$ must be a good estimation of the real pose x_t, tolerance is minimal. This criteria is evaluated by the magnitude of the error $e_t = \hat{x}_t - x_t$.
  Precision is also important, major variations on estimation are not allowed, since, in a real implementation, it would be used to control the vehicle. It is evaluated by the error e_t, but not only from his magnitude, also from his variation. If the error e_t has high frequency components along filter iterations, it means that the prediction is always jumping around the correct pose. The covariance of filter beliefs can be used to conclude about the precision as well.

- **Reliability** – This criteria is shaded by the previous two, we rely on a method if it is both accurate and precise, but here, the reliability concept implies more, it tests if the methods perform well in different types of situations, not just in normal operation. Here the tests are about the time that the method takes localizing the vehicle correctly. Time to globally localize the vehicle after a complete lost of vehicle track, for example if the system has a complete break-down and needs to localize the vehicle immediately after, typical kidnapping situation. With vehicle running and with vehicle stopped. We insist on global localization with stopped vehicle because in ITER buildings if something goes wrong, the most likely approach is to stop the vehicle until his pose is known again.

- **Robustness** – For different parameters and different possible failures tests if the algorithm still works and still performs well, it is robust. This includes robustness to noise, to sensor failure or even to sensor miss placement. Obviously this criteria depends both on localization method but also on sensory network displacement. Redundancy should be verified in the network to account for possible sensor failures. Robustness is studied over in Chapter 4 with different network displacements.

3.4.1 Accuracy and Precision

This is the main criteria evaluated in this Chapter, since we only do experiments for a single network, the objective her is to compare performance for the localization methods and not for the overall system.

The most important value to evaluate this criteria is the error, we will consider the error in position, for each iteration t, by the distance from real pose to estimated pose, $||l_{err}||(t)$ (3.63), and the error in orientation by the difference between estimated angle and real angle, $\theta_{err}(t)$ (3.64). The error is assumed to be gaussian, so it is computed the mean and standard deviation for each run of each method to establish the gaussian moments that describe the error. Mean values are $\mu_{||l_{err}||}$ and $\mu_{\theta_{err}}$, and standard deviations are $\sigma_{||l_{err}||}$ and $\sigma_{\theta_{err}}$ for distance and orientation errors, respectively (3.65). There are errors in map coordinate system, but there is an interest in knowing the error in vehicle coordinates. The new coordinate system (Figure 3.15) takes in consideration the vehicle pose, being the reference origin in vehicle center and the axis rotated with the vehicle. $\vec{L}$ is the axis along vehicle length and $\vec{W}$ along vehicle width.

Errors in those direction are computed by a geometric rotation in each iteration, depending on θ_r^t. $L_{err}(t)$ and $W_{err}(t)$ are the projections of $||l_{err}||$ on $\vec{L}$ and $\vec{W}$ directions.$\mu_{L_{err}}$, $\sigma_{L_{err}}$, $\mu_{W_{err}}$ and $\sigma_{W_{err}}$ are the gaussian moments of those projections.

On this coordinate system it is introduced a new value, it is a standard deviation of the on-line filter belief, this is computed for every iteration t and is a good evaluator about the certainty of estimation that filter has at time t. This is calculated based on eigenvalues of current belief covariance matrix, Σ_t. EKF, provides this matrix directly from his belief. In PF these matrix must be extracted from the particle set, Ψ_t, obviously this is an approximation of the particle distribution with a gaussian, accomplished with (3.66) and (3.67) for mean and covariance matrix, respectively.

The system's belief standard deviation is defined, at time t, by $L_{std}(t)$ and $W_{std}(t)$ in $\vec{L}$ and $\vec{W}$ directions, respectively, as represented in Figure 3.17. From values in this coordinate system it is possible to observe if the system is prone to have error along vehicle's length or width directions. This is a considerable knowledge because in most trajectories the vehicle moves mainly along the $\vec{L}$ direction, it is interesting to know if there are some correlation between the movement direction, or the observation direction, and the errors.

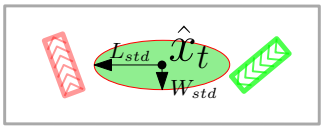


Figure 3.17: Estimation Standard deviation ellipse, L_{std}, W_{std}

$$||l_{err}||(t) = \sqrt{(\hat{x}_r^t - x_r^t)^2 + (\hat{y}_r^t - x_r^t)^2} \tag{3.63}$$

$$\theta_{err}(t) = \theta_r^t - \theta_r^t \tag{3.64}$$

$$\begin{aligned} \sigma_{||l_{err}||} &= \sqrt{\textstyle\sum_{t=1}^{IT} \left[||l_{err}||(t) - \mu_{||l_{err}||}\right]^2}, \qquad \mu_{||l_{err}||} = \tfrac{1}{IT} \textstyle\sum_{t=1}^{IT} ||l_{err}||(t) \\ \sigma_{\theta_{err}} &= \sqrt{\textstyle\sum_{t=1}^{IT} \left[\theta_{err}(t) - \mu_{\theta_{err}}\right]^2}, \qquad \mu_{\theta_{err}} = \tfrac{1}{IT} \textstyle\sum_{t=1}^{IT} \theta_{err}(t) \end{aligned} \tag{3.65}$$

$$\bar{x} = \sum_{n=1}^{N_p} w_t^{[n]} x_r^{[n]} \qquad \bar{y} = \sum_{n=1}^{N_p} w_t^{[n]} y_r^{[n]} \qquad \sum_{n=1}^{N_p} w_t^{[n]} = 1 \tag{3.66}$$

$$\begin{aligned} \Sigma_t &= \begin{bmatrix} \sigma_{xx}^2 & \sigma_{xy}^2 \\ \sigma_{yx}^2 & \sigma_{yy}^2 \end{bmatrix} \\ \sigma_{xy}^2 &= \frac{\sum_{n=1}^{N_p} w_t^{[n]} (x-\bar{x})(y-\bar{y})}{1-\sum_{n=1}^{N_p} \left(w_t^{[n]}\right)^2} \end{aligned} \tag{3.67}$$

The experiments are shown from Figure 3.19(a) to 3.27, according to Table 3.4.

Table 3.4: Experimental Results Figures

	CWtraj		*CCWtraj*		*VVtraj*	
	EKF	PF	EKF	PF	EKF	PF
Trajectory	3.19(a)	3.19(b)	3.22(a)	3.22(b)	3.25(a)	3.25(b)
$\|\|l_{err}\|\|$ and θ_{err}	3.20(a)	3.20(b)	3.23(a)	3.23(b)	3.26(a)	3.26(b)
L_{err}, W_{err} and L_{std}, W_{std}	3.21(a)	3.21(b)	3.24(a)	3.24(b)	3.27(a)	3.27(b)

Figures 3.19, 3.22 and 3.25, show the real path of vehicle center in dashed red, and path of estimated position in blue when the vehicle travels in *CWtraj*, *CCWtraj* and *VVtraj*, respectively. The initial jumps from the center to the path corresponds to method initialization, the initial pose given is in the center, so each method has to localize the vehicle globally, and just then follow his real trajectory.
To notice in these Figures that global localization is achieved almost in the beginning, which means it does not need much iterations to do it, it performs fast and accurately. To repair also that in general PF has a better performance, EKF tends to jump around the real pose.

Figures 3.20, 3.23 and 3.26, show the error $||l_{err}||$ and θ_{err} in both methods along the trajectories, the error is initially very large, during global localization procedure but rapidly decrease to very acceptable values. In both methods the position error is in the order of centimeters most of the time, EKF has big jumps in prediction, this problem is yet to solve, but is due to non-smooth behavior of measurement model, (section 3.2.2). The problem with this jumps on the prediction is a peculiar situation where some measurements hitting the vehicle in prediction and in reality, i.e. $\varphi_j^i \in [\chi(x_t, s_i) \cap \chi(\bar{\mu}_t, s_i)]$. But some of them hit different sides of the vehicle. The computation of jacobian matrix will have no correct meaning, since it depends on the side of the vehicle where each laser beam hits. The behavior will be as explain in Figure 3.18, these measurements will imply a wrong kalman gain and, consequently, a bad update step. One possible solution is to ignore measurements that are far away from the predicted ones, $(d_j^i - \bar{d}_j^i > threshold)$ but is impossible to know which side of the vehicle the sensor measurements are coming from, as we don't know his pose. Ignoring more measurements besides the ones already ignored, imply neglecting many useful

information. This extra feature is not implemented during these experiments.
It is visible in Figure 3.23 the constant error of 180° this is a problem of EKF, as well, the symmetry of the vehicle leads to very indistinguishable measures, and not even by velocities integration the method can converge to correct orientation, as explained in section 3.3.3.

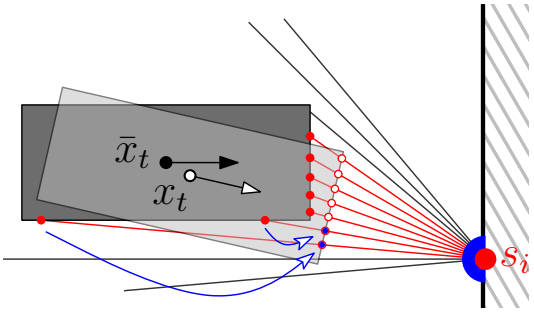
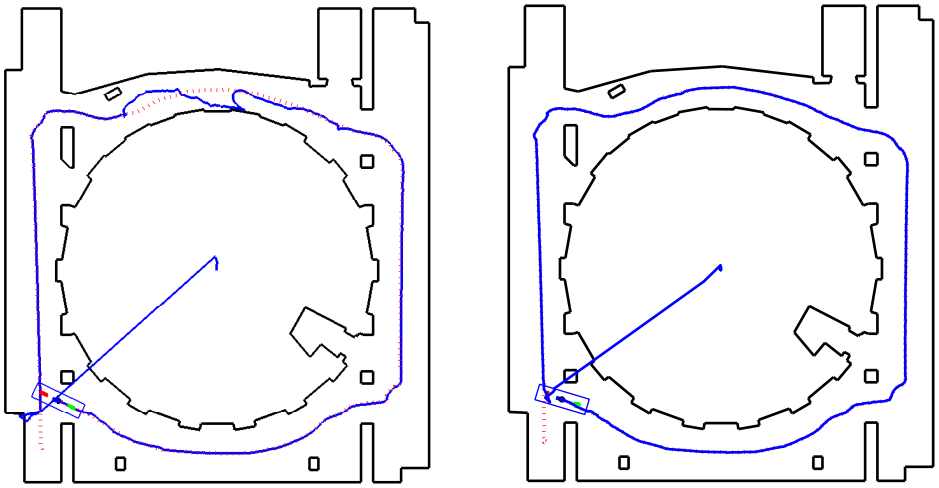


Figure 3.18: Non-smooth observation model issue

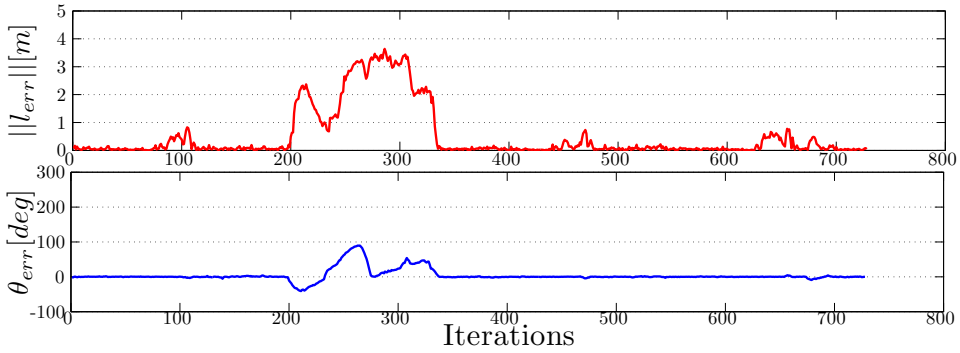
Figures 3.21, 3.24 and 3.27, show the errors and standard deviations of filters' beliefs along the different paths, for $\vec{L}$ and $\vec{W}$ directions. It is visible that, in normal operation, (excluding the jumps in EKF and the initial global localization), the errors and standard deviations are larger in $\vec{L}$ direction. This has a pure relation with the map geometry and the distribution of the sensor network. TB is essentially formed by long corridors, in these experiments the sensors are placed in the outer walls to maximize the coverage, but, as the vehicle passes by, the laser rays tend to hit the side of the vehicle, measurements are collinear with $\vec{L}$, increasing the uncertainty in that direction.
PF tends to have less deviation, the integration of measurements that do not hit the vehicle has an important role in this behavior. If particles start migrating in $\vec{L}$ direction, measurements that do not hit the vehicle will rapidly decrease their weight.



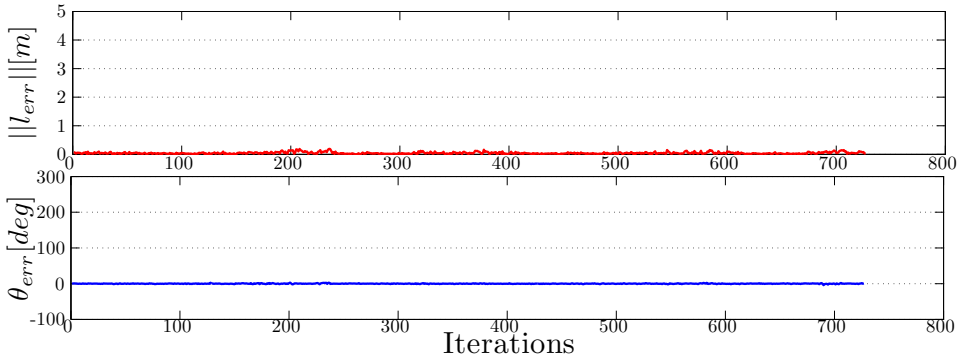
(a) EKF estimation

(b) PF estimation

Figure 3.19: Estimated and real path for vehicle center, CWtraj

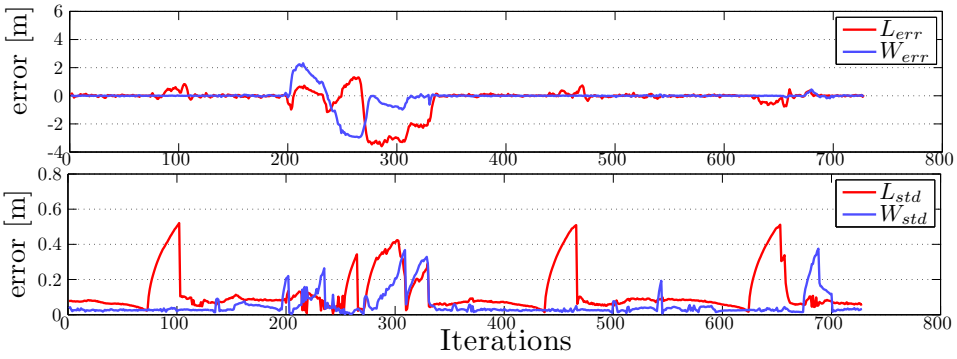


(a) EKF estimation

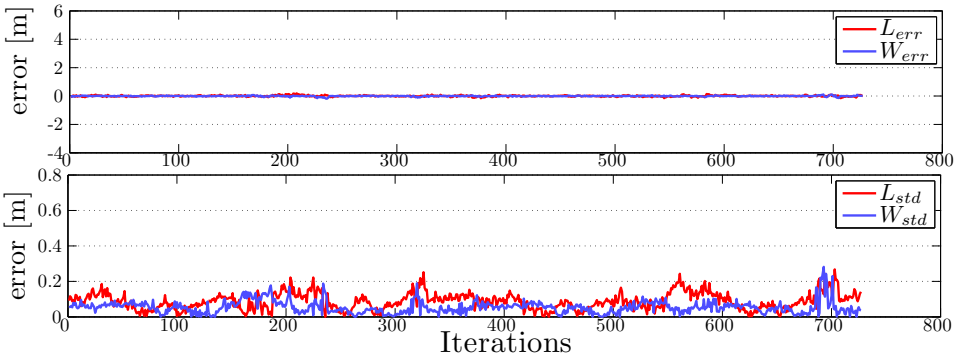


(b) PF estimation

Figure 3.20: Estimation error, CWtraj

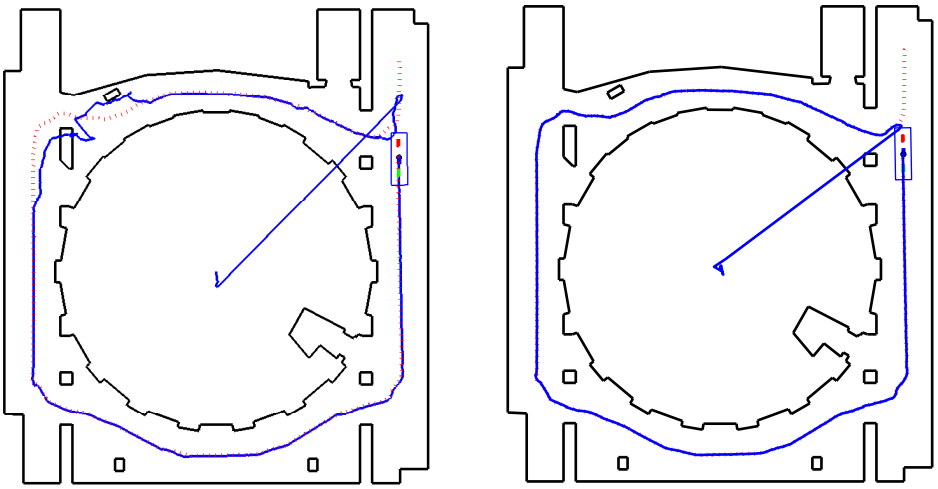


(a) EKF estimation



(b) PF estimation

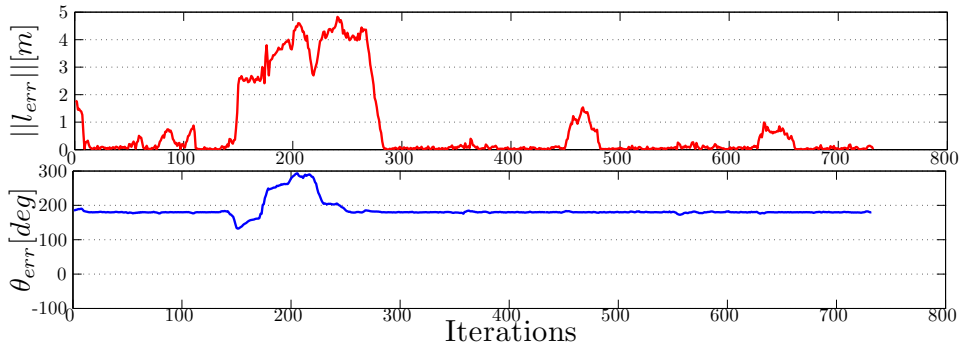
Figure 3.21: Estimation error and variance along vehicle coordinates, CWtraj



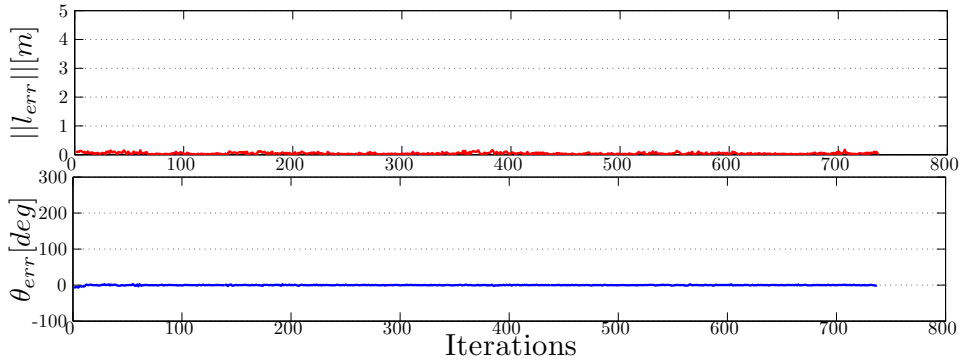
(a) EKF estimation

(b) PF estimation

Figure 3.22: Estimated and real path for vehicle center, CCWtraj

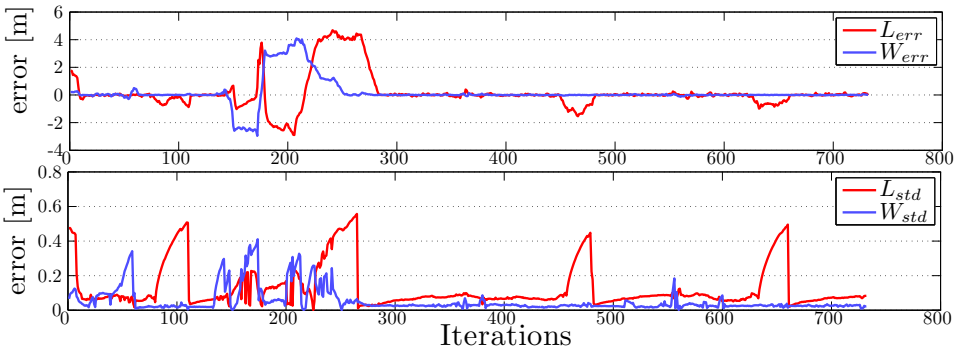


(a) EKF estimation

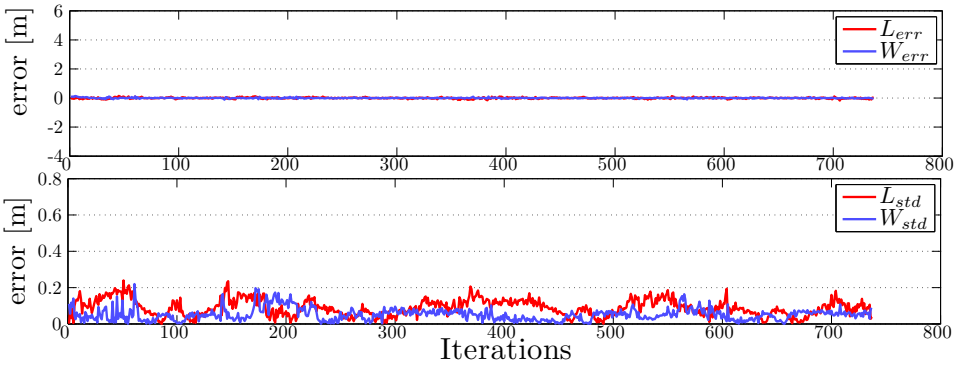


(b) PF estimation

Figure 3.23: Estimation error, CCWtraj

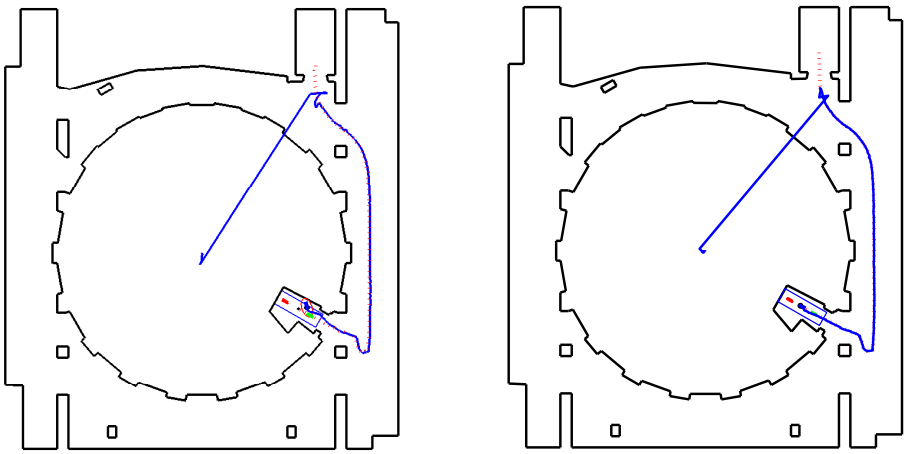


(a) EKF estimation



(b) PF estimation

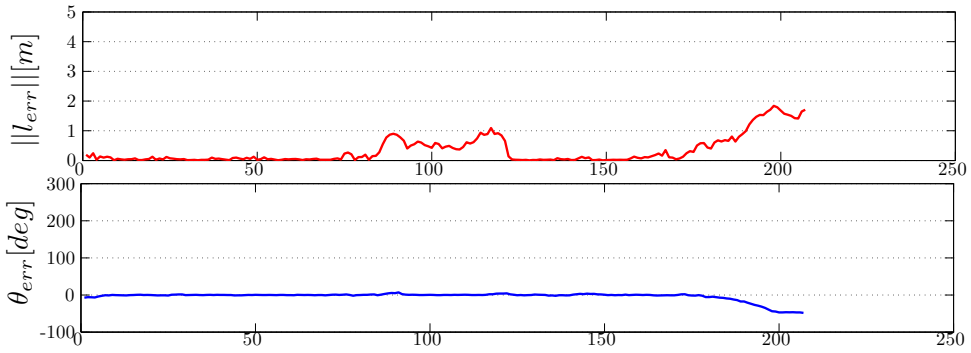
Figure 3.24: Estimation error and variance along vehicle coordinates, CCWtraj



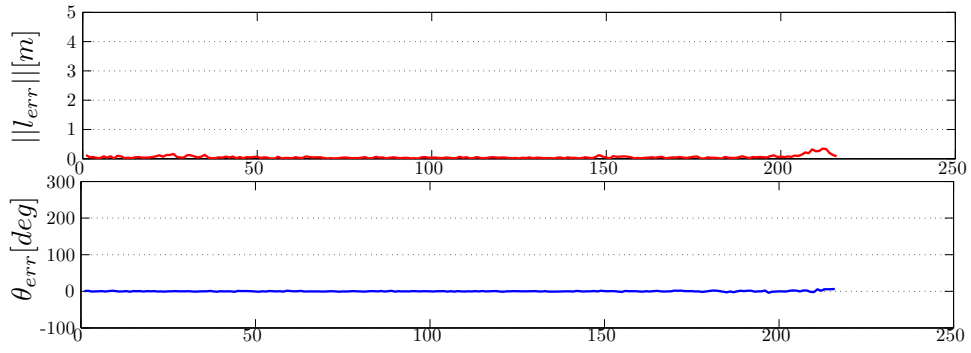
(a) EKF estimation

(b) PF estimation

Figure 3.25: Estimated and real path for vehicle center, VVtraj

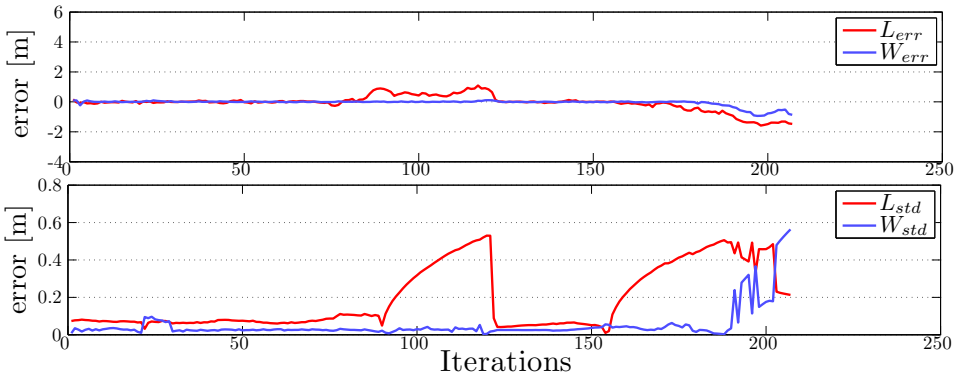


(a) EKF estimation

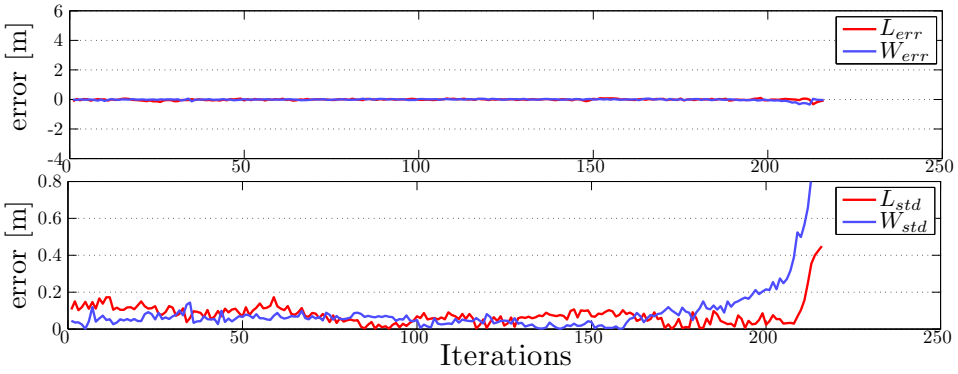


(b) PF estimation

Figure 3.26: Estimation error, VVtraj



(a) EKF estimation



(b) PF estimation

Figure 3.27: Estimation error and variance along vehicle coordinates, VVtraj

Analyzing all the trajectories and tests done, Tables 3.5 and 3.6 compile the most significant data that states the accuracy and precision of both methods. PF is the better filter for these criteria, this is due to the possibility of having non-linear and non-gaussian models associated, which fits better the current application. PF need some adaptations and heuristics to solve non-smoothness problems, on the observation models, which inevitably decreases the filter performance.
EKF performs well for cases where the prediction and real pose are very near, but with a little deviation it can lose track of the vehicle very quickly, the filter is not stable is some situations. So in terms of accuracy, precision and stability, the PF is the better choice.

Table 3.5: Estimation Errors, position [mm] and orientation [deg]

		EKF		PF	
		$\|\|l_{err}\|\|$	θ_{err}	$\|\|l_{err}\|\|$	θ_{err}
CWtraj	σ	944.18	17.63	33.38	0.69
	μ	515.88	4.01	45.50	0.02
CCWtraj	σ	1365.3	23.96	30.40	0.91
	μ	786.25	185.95	43.68	-0.07
VVtraj	σ	457.34	11.42	51.71	1.22
	μ	336.91	-3.72	53.98	0.13
All traj.	σ	1122.20	93.39	35.21	0.87

Table 3.6: Estimation Errors, $\vec{L}$ and $\vec{W}$ directions [mm]

		EKF		PF	
		L_{err}	W_{err}	L_{err}	W_{err}
CWtraj	σ	838.06	636.28	47.41	30.25
	μ	-213.79	-70.71	4.00	-2.98
CCWtraj	σ	1249.90	933.78	45.42	27.69
	μ	145.63	165.12	-2.31	-0.29
VVtraj	σ	522.33	207.19	50.32	50.68
	μ	-61.94	-59.86	-11.27	-12.83
All traj.	σ	1026.7	760.3	47.18	33.21

3.4.2 Reliability

From results shown on Figures above we can make a judgment about the reliability of a method, EKF often lose track of real pose and is not very realiable for this task. Moreover there are some values, concerning global localization, that can also be analyzed to decide on the reliability of a method for the ITER TB required tasks.
In this environment, there should be always a redundant way of resolving a problem, if something goes wrong. One of the possible failures is the system losing track of the vehicle, for some reason, power failure or hardware malfunction. The system must be capable when the system is operable again, to localize the vehicle globally, if it is stopped or moving.

Table 3.7 shows the number of iterations spent from loss of vehicle until localization with a steady estimation around correct pose. Multiple experiments were made by moving the vehicle from several different poses to others while the filter is paused, the values are a mean from all experiments for the respective conditions.
We can see both methods converge very fast due to the heuristic adopted, computing the measurements mass center to reinitialize the filter belief distribution. Sometimes a method gets stuck in a geometric ambiguity, like described in section 3.3.3, to solve the problem, PF generates with a certain probability new particles around the ambiguity states, when stopped, in every corners and when moving, with 180 degrees of orientation offset. Kalman does not allow this kind of distribution, with multiple modes, so there is no way to avoid ambiguities with EKF, making the PF a more reliable method for this task as well. It allows the insertion of known information about ambiguities, with high simplicity, by generating new distribution modes around those poses, while EKF is limited to the gaussian representation.

Table 3.7: Global Localization, number of iterations

EKF Stopped	PF Stopped	EKF Moving	PF Moving
73	38	48	30

3.4.3 PF computational effort

A particular test to evaluate the effect of number of particles, N_p in the PF performance, leads to Table 3.8, where we can see a small improvement, the standard deviations of the errors tend to decrease as the number of particles grows, but the improvement is, at this level, very small compared to the increasing computational effort, so, in simulation framework we kept the particle number to $N_p = 100$.
Large numbers of particles are typically necessary, in robotic localization, in large spaces with multiple symmetric areas, where the particle distribution can become multi-modal, and sometimes, for kidnapping situations there is the need to spread particles around all possible poses. On our problem such situations are very limited, since the sensory network is already global, the particles normally are very localized. A large number of particles is not necessary here.

Table 3.8: Error Gaussian moments vs number of particles

N_p	$\mu_{\|\|l_{err}\|\|}$ [m]	$\sigma_{\|\|l_{err}\|\|}$ [m]	$\mu_{\theta_{err}}$ [deg]	$\sigma_{\theta_{err}}$ [deg]
100	0.0429	0.0275	0	0.5490
150	0.0375	0.0281	0.0718	0.5539
200	0.0337	0.0199	0.0196	0.4498
300	0.0324	0.0193	0.0378	0.3376
400	0.0352	0.0236	0.0361	0.3254
500	0.0316	0.0211	0.0191	0.2930

From these experiments, and the results obtained in this framework, the filter that suites

the problem is the PF, since his capability of maintain a non-gaussian belief is helpful to solve the geometric ambiguity problems. The non-smooth, non-linear observation model is also a weak link of EKF approach, being PF more robust for these issue.The first impact from the simulations shown, and from the values obtained is that PF is the correct choice for these kind of localization system.

3.5 Discussion

The Vehicle Localization chapter makes a extensive description of both EKF and PF filter implementation for a distributed LRF sensor network, both methods have powerful capabilities, but by the results shown the PF is a better choice for this kind of problem.
The accuracy, robustness and stability is better for PF estimation then for EKF, tests performed during this chapter are a good proof of this. The choice for the three trajectories shown CWtraj, CCWtraj and VVtraj, is directly connected to the functions of the TCS and the future application, the two CW and CCW trajectories on the corridors are a good way to evaluate the performance, since the corridors are the main area visited by the vehicle. The last, VVtraj, is an example real trajectory.
The performance in special situations, like loss of track, stopped vehicle is also very important, inside the rad-hard scenario there is no room for failures, and the system must be capable of self recovering from eventual malfunctions.
Table 3.9 has a summary evaluation of the two methods, where the classification has 3 levels, - for bad performance, + for acceptable performance, with limitations or failures, ++ for good performance.

Table 3.9: Summary localization method evaluation

	EKF	**PF**
Accuracy	+	++
Stability	+	++
Robustness	+	++
Recovery from failure	+	+
Computation Effort	++	+
180° ambiguity	-	++
Corner Ambiguity	-	+

From the tests performed and results obtained, the logical choice to integrate the system localization system is the PF, although it becomes hard to compute for a very high number of measurements. The computational effort pays back in performance.

Chapter 4

Localization dependence on sensor placement

The entire system proposed in this thesis concerns mainly the localization system, but also an algorithm to optimize the sensor's placements (Figure 4.1). We have saw the optimization procedures in Chapter 2, where the network displacement is optimized taking in consideration the coverage. In Chapter 3, two localization method are conceived and compared, making no considerations about the sensor network that supplies the measurements.
This Chapter studies the effects of sensors parameters on localization performances, analyzing the main error characteristics facing different network configurations. It is basically a chapter of experimental results with different network configurations. The objective is to conclude whether optimal sensor placement is crucial to have a good performance, or the algorithms are robust to multiple sensor placements and still perform at the same levels. Another consideration is the computational effort necessary as the network grows, being that an important topic because these methods should perform at real-time. Concerning the sensors characteristics itself, it is also studied how different angular resolution affects the localization performance.
The results shown in this chapter should help to refine optimization criteria, if necessary, and contribute to the understanding of method robustness to failures on sensors.

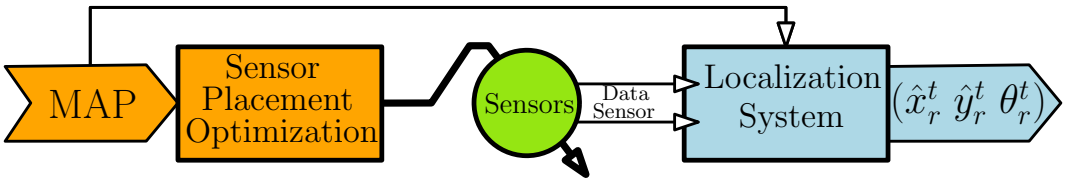


Figure 4.1: Entire System's block diagram

4.1 Sensor characteristics — Angular Resolution

To study the behavior of the system with the acquisition of more or less detailed measurements, we made an experiment where the angular resolution, δ_s^i, assumes different values, corresponding to the ones found on typical commercial sensors, [40]. The experiment is described in Figure 4.2, vehicle travels along a circular path, and is observed from two sensor, without occlusions. This tens all the possible angles of observation. The vehicle executes 20 laps around the path, and in the end, mean and standard deviation of the errors is computed and shown on Table 4.1.

It is visible that, increasing the resolution the PF has better results, but EKF has not a clear improvement . The behavior of PF is expected, as we have more measures, more information, better knowledge about vehicle position. EKF has a non expected behavior, but, this is due to non-smoothness of observation model, as cited in section 3.4, Figure 3.18. With laser rays closer to each other, there is a higher probability of occurrence of these situation, and so, the estimations tends to be less stable with more angular resolution.

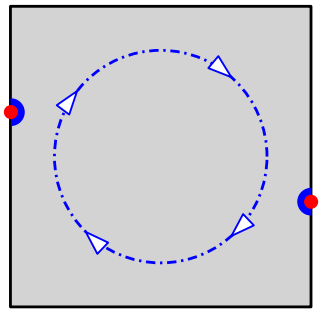


Figure 4.2: Angular resolution experiment

Table 4.1: Error for different angular steps [mm]

		$\delta_s^i = 1°$		$\delta_s^i = 0.5°$		$\delta_s^i = 0.25°$		$\delta_s^i = 0.125°$	
		μ	σ	μ	σ	μ	σ	μ	σ
EKF	$\|\|l_{err}\|\|$	95.7	83.9	77.1	85.4	58.5	61.7	70.2	115.3
	θ_{err}	0.01	1.58	-0.04	1.38	-0.01	1.21	-0.07	1.13
PF	$\|\|l_{err}\|\|$	46.0	26.1	32.8	19.8	25.3	15.7	24.8	15.5
	θ_{err}	-0.03	0.71	0.01	0.52	0.01	0.45	-0.01	0.44

4.2 Network characteristics

4.2.1 Number of sensors

The number of sensors affects directly the localization performance, with more sensors there is more accuracy and more time spent per iteration. This section tests the effect of number of sensors, L, in the system performance, the error of localization is tested using the optimized networks from Chapter 2, the computational effort is measured by the mean time spent on simulation for each configuration.

Localization errors

The results obtained show that the errors in position and orientation are, in mean, inferior as L grows. This was expected, intuitively with more information acquired by the system, accuracy should be better. With more sensors, there is a larger area covered and an higher probability to hit different sides of the vehicle, which minimizes uncertainty.
EKF results, show that this method's performance increase a lot with the number of sensors, it has much more reliable results with high laser ray density, than it has with low, but still, has the worst performance, comparing to PF.
PF achieves better results, but his performance grows slowly with the number of sensors. The methods behavior is a direct consequence of the observation models applied, being non-smooth there are measurements pulling the EKF prediction away from the correct pose, but, with more sensors, EKF almost neglects measurements that generate a bad behavior on the update step. The measurements pulling the estimation to the correct pose are much more, and overcome the "bad-behaved" measurements.
PF is more robust to L changes, it performs much better for networks with few sensors, so the increment of L has low impact in this filter.
Results for a network of just one sensor, are out of acceptable values, and do not account for the conclusions, this is because the network does not cover wide areas of the map, when the vehicle passes those areas, there is no way to do the update step, however, when the system gets measurements of the vehicle the global localization is done well, as the results shown are means along the all trajectory, for 1 sensor, the results have no meaning.
The values obtained in simulation for error mean are visible in Figure 4.3 and for error standard deviation, in Figure 4.4 where we see that PF estimation is always more accurate than the one from EKF. For the experiment with $L = 3$ EKF estimation has a big error in orientation, this is caused by the 180° ambiguity that is not yet solved for this method.
Numerical results are in Table 4.2, proving that, except for $L = 1$, the accuracy is better for PF than in EKF, and that PF is much more stable with L variation.

Table 4.2: Error for optimized network configurations [mm]

		$L=1$		$L=2$		$L=3$		$L=4$		$L=5$	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
EKF	$\|\|l_{err}\|\|$	9942.0	16453.1	811.1	1011.1	576.1	1199.1	323.7	614.3	365.7	873.8
	θ_{err}	103.58	79.81	0.66	14.30	-139.76	68.96	3.59	15.14	4.29	16.46
PF	$\|\|l_{err}\|\|$	13647.3	23492.1	125.8	173.7	52.8	37.8	44.6	31.8	42.2	27.5
	θ_{err}	14.67	39.32	-0.30	3.34	0.01	0.92	-0.25	6.87	-0.98	7.66

		$L=6$		$L=7$		$L=8$		$L=9$		$L=10$	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
EKF	$\|\|l_{err}\|\|$	220.1	568.7	123.4	206.9	73.9	58.9	85.3	91.1	72.7	60.5
	θ_{err}	21.22	55.41	0.47	4.16	0.08	1.26	0.25	2.57	-0.05	1.27
PF	$\|\|l_{err}\|\|$	44.1	35.7	41.0	31.5	35.8	24.2	36.4	26.4	38.2	54.8
	θ_{err}	-0.01	0.61	0.03	0.66	0.01	0.51	-0.04	0.58	-0.45	12.18

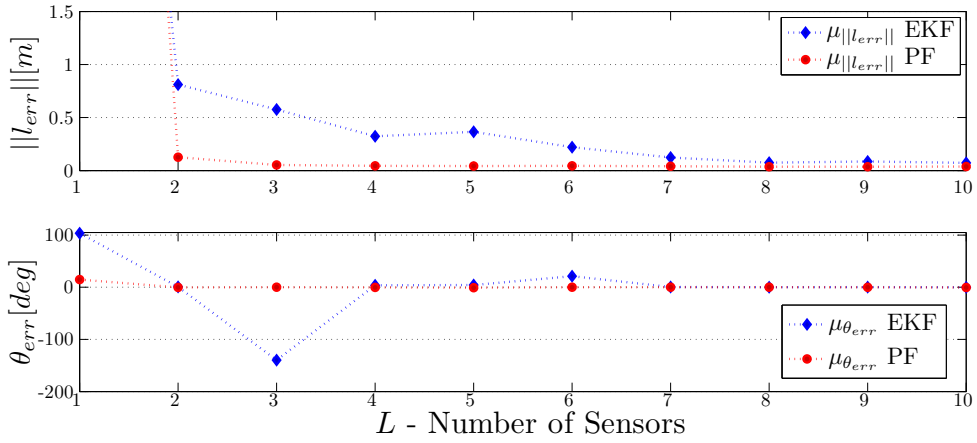


Figure 4.3: Error mean for network with L sensors

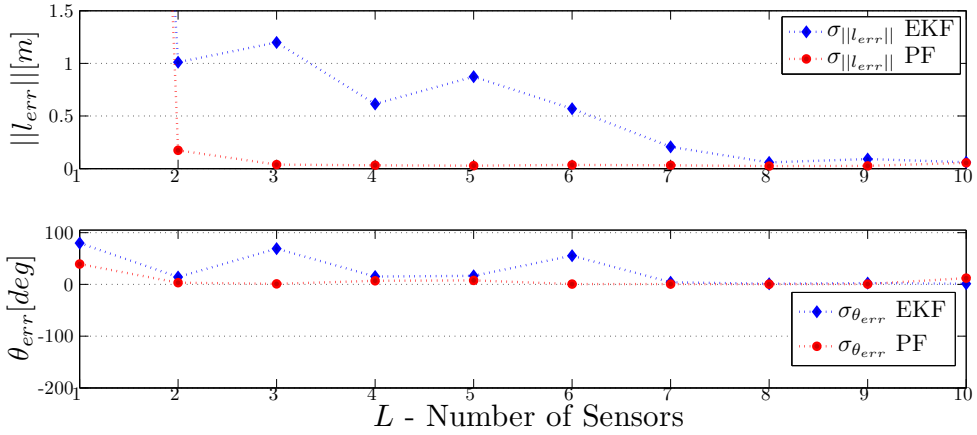


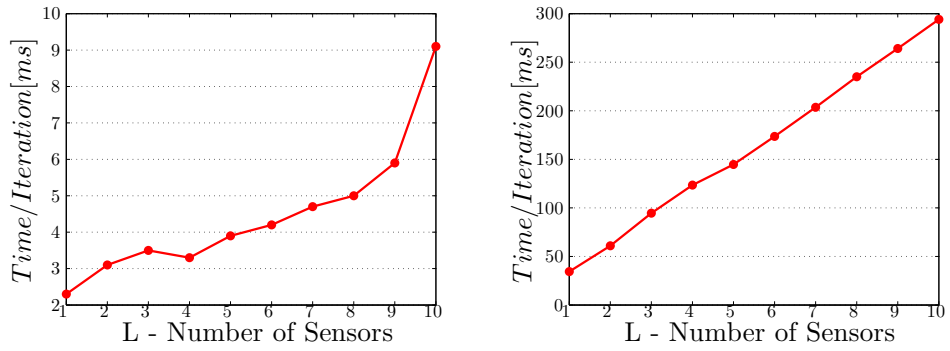
Figure 4.4: Error standard deviation for network with L sensors

Computational Effort

EKF and PF algorithms have a great difference considering these criteria, Table 4.3 shows quantitatively and Figure 4.5 illustrates the curves of mean time per iteration depending on the number of sensors placed in the scenario. Quantitative values are not very important, since they depend on the machine where the algorithm runs, they are only used here for comparison purposes and to analyze the behavior of the algorithm as the network grows.
As expected, the PF algorithm is much slower, and his computation time increases lineally (Figure 4.5(b)) with the number of sensors. This is caused because in every iteration PF analyzes every sensor measurement. EKF does not use all measurement, only the ones hitting the vehicle, so as the network grows, the algorithm is not directly affected. The small increment on computation time (Figure 4.5(a)), is due to the higher density of laser rays. So there are, normally, more rays hitting the vehicle.
This may be a reflection topic, as the EKF good time performance is not just from his type of implementation, with linear models and fast matrix inversion, but also from the fact that it neglects many measures. PF, is slower because it repeats computations for every particle, but also due to the evaluation of every measurement, every iteration.
Intuitively, the PF precision is better, in part, due to the integration of all measurements, this gives information about where is the vehicle and where it is not, while EKF only integrates information about, where it is. During normal operation, the laser rays containing the more important information are the ones hitting the vehicle and close to it, measurements from the other side of the map are irrelevant. It is likely that, with a better selection criteria, PF could have a good performance with less computational effort.

Table 4.3: Time/iteration vs number of sensors $[ms]$

L	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
EKF	2.3	3.1	3.5	3.3	3.9	4.2	4.7	5.0	5.9	9.1
PF	34.4	61.0	94.6	123.5	144.7	173.6	203.6	235.0	264.1	294.1



(a) EKF Algorithm (b) PF Algorithm

Figure 4.5: Mean time per iteration vs Number of sensors

4.2.2 Redundancy

Different network displacements, cause changes in coverage and redundancy, but, for a large number of sensors, the coverage is rarely affected, unless the sensors are putted all in the same position. If sensors are distributed in the scenario, coverage is almost constant. Redundancy is more sensible to the sensor placement, in this section we analyze the effect of coverage $R(S)$ on system's accuracy and precision. The impact of redundancy on estimation quality will dictate if the redundancy optimization is crucial for the overall system performance.
The experiments were performed with several networks with 8 sensors. $L = 8$ was chosen due to the behavior during optimization, Figure **??** from Chapter 2 shows that, for the same coverage, there are networks with wide values on redundancy. The results shown in Table 4.4 are the mean and standard deviation of the estimation error for vehicle traveling along several trajectories in TB. Figure 4.6 represents the mean error on several runs of the vehicle in th TB with a network with the respective redundancy value.
It is noticeable that as the redundancy grows EKF yields less error in estimation while PF keeps around the same values. This is another proof of PF robustness, it is not much affected by changes in redundancy. EKF is very sensible to the measures that hit the vehicle and if it has more measures in each step, it has better accuracy, with redundancy reduction, measures integrated by the EKF per iteration reduce as well. It contrasts with PF that has no direct impact from redundancy, since coverage is not compromised it works with a very acceptable accuracy.
This yields a conclusion that, with PF the optimized placing of the sensors is not very important for a good response, but, taking other factors in consideration, as redundancy in case of sensor failure, or optimization of the number of sensors needed. The optimization process is important.

Table 4.4: Error for different redundancy $R(\hat{S}_L)$, [mm]

	$R(S) =$	$86,7\%$		$77,2\%$		68.0%		63.4%		56.4%		47.5%	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
EKF	$\|\|l_{err}\|\|$	73.9	58.9	85.9	81.4	99.8	117.3	393.9	1096.1	658.0	1375.9	982.3	1780.2
	θ_{err}	0.08	1.26	0.20	1.93	0.10	2.00	71.84	84.26	-65.34	79.36	43.84	67.27
PF	$\|\|l_{err}\|\|$	35.8	24.2	39.5	26.2	36.8	24.2	40.7	31.3	36.5	25.6	42.7	29.3
	θ_{err}	0.01	0.51	-0.21	8.24	-0.01	0.60	0.03	0.65	-0.01	0.55	0.22	0.78

4.2.3 Sensor Failure

This section shows the filters behavior with a single sensor failure, the experiment was performed excluding turning off one of the sensors while the vehicle travels along the trajectories. In the end is extracted the mean and standard deviation of the error. The network, S, used was the optimized from Chapter 2 with 8 sensor and maximum redundancy, this sensor displacement gives a coverage $C(S) = 99.9\%$ and a redundancy of $R(S) = 86.7\%$ which is a good value for evaluate if, in case of failure, the system can still perform.
Table 4.5 show the values of reference, with no failures, for this network. The failure values

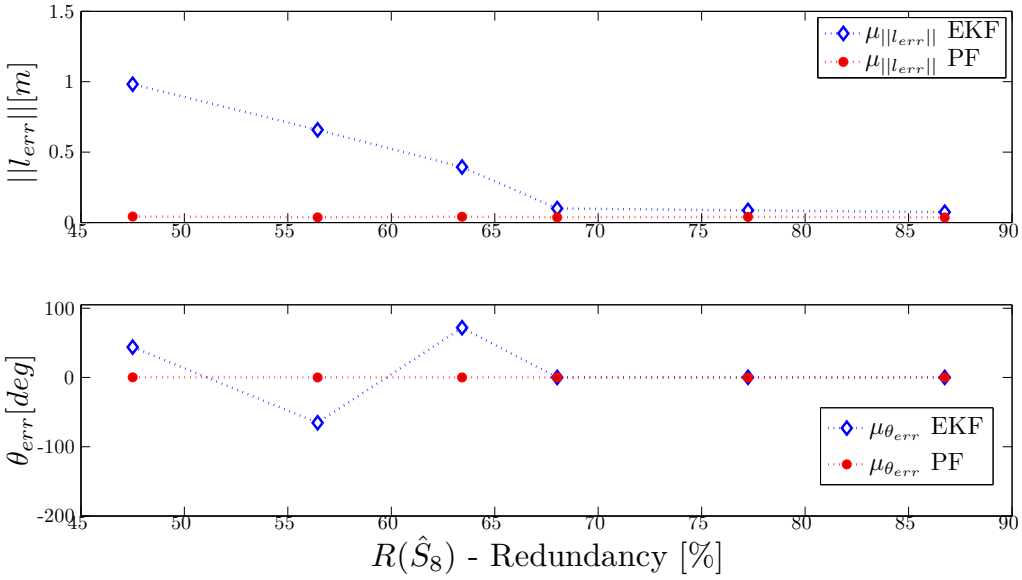


Figure 4.6: Mean errors for networks with different redundancies

were obtained, for the same trajectory, by the mean for several experiments, with different sensors failing.

From both methods, PF is much more robust, having just a small increase of mean error, with failure of one sensor, in the order of 10 mm, EKF fails in this situation, the errors in estimation are not acceptable when a sensor fails.

EKF depends much more on the quality of measurements than PF, putting PF has the most robust method in failure situations.

Table 4.5: Sensor Failure Robustness , position [mm] and orientation [deg]

		EKF		PF	
		$\|\|l_{err}\|\|$	θ_{err}	$\|\|l_{err}\|\|$	θ_{err}
No Failures	σ	58.9	1.26	24.2	0.51
	μ	73.9	0.08	35.8	0.01
Sensor Failure	σ	754.6	28.89	31.2	0.65
	μ	351.4	40.0	40.1	0.03

4.2.4 Sensor Miss Placement

Until this section, the correct placement of the sensor is assumed to be known by the algorithm, but what happens if there is an error on installation and the sensor has a small deviation from the assumed position?

In this section this situation is simulated, giving a position in localization algorithm input, and another position to simulate the sensor readings.

The objective is to test if the methods are robust to deviations from assumed positions, because for all assembly processes there are maximum tolerances and the sensor placement

is no exception.

The experience framework is similar to the one explained in section 4.1, with a circular path, but now one of the sensors have a deviation *dev* along the wall. Figure 4.7 and Table 4.6 present the results from this experiment, which reveals that the methods are not robust to this fact, tending to have an offset in the estimation, with amplitude similar to the deviation. The installation of sensors should be a very accurate process to guarantee the system performance. A "first-use" calibration is also a possible solution, as the map is known, the sensor can locate itself from map readings.

Table 4.6: Error for different sensor deviations [mm]

		$dev = 0$		$dev = 10$		$dev = 30$		$dev = 50$		$dev = 100$	
		μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
EKF	$\|\|l_{err}\|\|$	91.2	63.8	100.2	84.0	93.6	78.6	109.2	114.7	119.3	89.8
	θ_{err}	0.11	1.67	0.16	1.56	0.09	1.30	-0.09	1.68	0.30	1.64
PF	$\|\|l_{err}\|\|$	46.2	26.6	47.9	27.9	50.7	28.8	54.9	31.5	78.5	39.4
	θ_{err}	0.04	0.66	0.02	0.65	-0.01	0.68	0.01	0.69	0.01	0.76

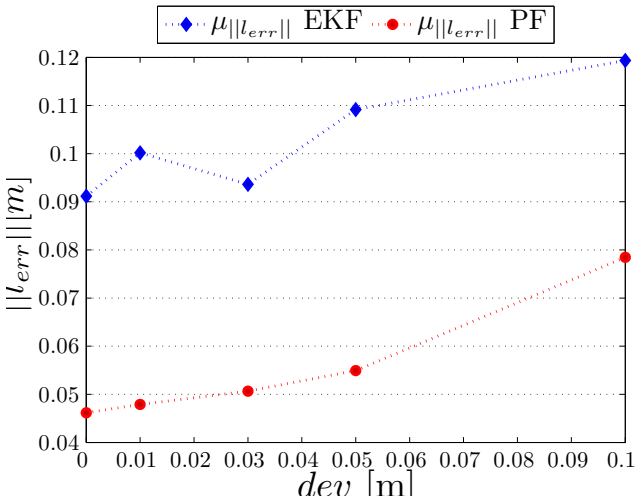


Figure 4.7: Error mean for different sensor placement deviation

This Chapter is composed essentially by experimental results, but evaluating not only the localization methods, but also the sensor network impact on the final system estimation. A previous conclusion for this system is that the PF is the best choice, due to his robustness and accuracy, and that optimization of sensor placement is important but only if it includes coverage and redundancy. Coverage optimization only, like in Chapter 2, is sufficient to a acceptable estimation of the vehicle pose, but in case of failure of one sensor, the estimation degrades a lot.

Localization methods perform well and, with some adjustments, and adaptation to each specific situation, with appropriate heuristics, it is expected that they could perform better.

Chapter 5

Conclusions

The creation of a localization system for a vehicle, operating indoors, with no sensors on board was the main objective of this thesis. It was achieved resorting to a laser range finder sensor network, installed on building walls.
This network observes the surroundings, and, integrating measurements with a Bayes filter, returns the estimated pose of the vehicle (position and orientation).
Being the sensors installed on walls, they are static, and their placement is crucial to the system performance. The areas covered by the sensors, and the redundancy of the system depend critically on sensor placement. An optimization procedure was developed thinking in this critical dependency, the better the network, the better the performance. Simulated Annealing was the selected method to optimize the network placements, it optimizes an evaluation function that, on Chapter 2, contemplates only the covered area. This is the main criteria to optimize, because if there are areas without coverage, observation on vehicle cannot be made, and the estimation is far away from real pose.
From the several experiences with the localization methods, we conclude that redundancy is also a very important factor to maximize. Besides allowing the system to work correctly during a sporadic sensor failure, during normal operation it gives more information to the system. If an area is observed from several sides, when the vehicle passes by, the uncertainty on estimation is much smaller than observing from just one side.
Localization algorithms are based on Bayes filters, two proposes were evaluated, one based on EKF and the other on PF. From Chapter 3 we can conclude that PF is the better suited method for this application. It is more accurate, and robust then EKF, in every situation tested.
The better behavior of PF is due to it's better adaptability to non-linear, non-smooth processes while the EKF is confined to gaussian representation. Filter belief representation is also limited to a gaussian function for EKF, but PF can have multi-modal distributions, which allows the introduction of simple heuristics that solve big problem. The 180° orientation error or the corner ambiguity when the vehicle is stopped.
Having a global network also allows for fast global localization, since we have always measurements from the entire building.
Computation effort is the weakness of PF, it is much slower then EKF, but, it still performs

in real time, and pays back in estimation precision and accuracy.
Performance for the overall system, network plus localization method, is evaluated in Chapter 4, where it is proven that PF is more robust networks with few sensors, to sensor failures or to modification of sensor placements, decreasing redundancy. PF is the best method to apply in this case, but still has some problems to solve, has described in next section (5.1).

Although it has some failures, and is still only working in simulation environment, it yields promising results, not only for application in ITER, but for other application with this type of requirements, or even to implement an AGV system where the paths can change dynamically, there is no need to physically install cables or stripes on the floor. Today's factories are in constant layout change, depending on the product in manufacturing, so an efficient way to have different paths for autonomous vehicles can be of industry interest as well.

5.1 Future work and open issues

This thesis corresponds to the first concept of the proposed system, designed from the requirement of having sensors outside the vehicle, it still has a lot to work on before it reaches a practical stage. With this in mind, some future work and open issues are listed here:

- **Redundancy optimization** — currently the SA algorithm only maximizes the coverage of the network, but as seen in Chapter 3 and 4 redundancy is also very important. So a method for multiple criteria optimization should be addressed in future work, although by including redundancy to the current evaluation function or by using methods of multi objective optimization.

- **Calibration system** — in Chapter 4 we identify a problem on localization system, it is intolerant to sensor deviation from predefined placement, it introduces an offset on estimated poses.
  Has this is a static error, it can be corrected with a calibration system. After assemble stage there must be a system that given the map and the measurements from sensors can calibrate the system and retrieve the correct placements for each sensor.

- **PF measurement pre-selection** — one of the current problems with PF solution is the computational effort, as the sensor network grows the method becomes slower. But from operation, it is known that when the estimation is near the correct pose, the majority of measurement integrated are irrelevant to the system. Just the measurements around the pose contain information. A method to filter the interesting and irrelevant measurements can decrease a lot the effort needed to compute PF iterations.

- **Multiple vehicles** — If there are more than one vehicle traveling along the building, it should be possible to track it also, apart from occlusions between vehicles, the network is always capable to observe both. So a possible interesting development of this technique is the inclusion of multiple objects to track, vehicles with different shapes, etc.

- **Practical Implementation** — It is still far away from this, much more work have to be done, but still, it is a goal for future work, to implement this work on a practical experiment.

Bibliography

[1] A. Aggarwal. The art gallery theorem: its variations, applications and algorithmic aspects. 1984.

[2] K.O. Arras, N. Tomatis, B.T. Jensen, and R. Siegwart. Multisensor on-the-fly localization::: Precision and reliability for applications. *Robotics and Autonomous Systems*, 34(2-3):131–143, 2001.

[3] J. Biswas and M. Veloso. Wifi localization and navigation for autonomous indoor mobile robots. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4379–4384. IEEE, 2010.

[4] M. Bouet and A.L. Dos Santos. RFID tags: Positioning principles and localization techniques. In *Wireless Days, 2008. WD'08. 1st IFIP*, pages 1–5. Ieee, 2009.

[5] J. Carpenter, P. Clifford, and P. Fearnhead. Improved particle filter for nonlinear problems. *Radar, Sonar and Navigation, IEEE Proceedings* -, 146(1):2 –7, feb. 1999.

[6] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[7] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.

[8] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars. *Computational geometry: algorithms and applications*. Springer-Verlag New York Inc, 2008.

[9] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 1322–1328. Citeseer, 1999.

[10] U.M. Erdem and S. Sclaroff. Automated camera layout to satisfy task-specific and floor plan-specific coverage requirements. *Computer Vision and Image Understanding*, 103(3):156–169, 2006.

[11] J. Ferreira, A. Vale, and R. Ventura. Optimizing range finder sensor network coverage in indoor environment. In *IFAC Intelligent Autonomous Vehicles (IAV 2010)*, 2010.

[12] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE Proceedings*, volume 140, pages 107–113, 1993.

[13] V. Granville, M. Krivanek, and J.P. Rasson. Simulated annealing: a proof of convergence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):652–656, 1994.

[14] J.S. Gutmann, W. Burgard, D. Fox, and K. Konolige. An experimental comparison of localization methods. In *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on*, volume 2, pages 736–743. IEEE, 2002.

[15] P. Hiemstra and A. Nederveen. Monte Carlo Localization. 2007.

[16] S. Jain. A survey of Laser Range Finding. *Unpublished paper. Accessed March*, 30:2007, 2003.

[17] S. Kirkpatrick, C.D. Gelatt Jr, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671, 1983.

[18] K.J. Kyriakopoulos and N. Skounakis. Moving obstacle detection for a skid-steered vehicle endowed with a single 2-d laser scanner. volume 1, pages 7 – 12 vol.1, sep. 2003.

[19] J.J. Leonard and H.F. Durrant-Whyte. Mobile robot localization by tracking geometric beacons. *IEEE Transactions on Robotics and Automation*, 7(3):376–382, 1991.

[20] J. Liu, R. Chen, and T. Logvinenko. A theoretical framework for sequential importance sampling and resampling. *Sequential Monte Carlo Methods in Practice*, pages 225–246, 2001.

[21] C. Losada, M. Mazo, S. Palazuelos, D. Pizarro, and M. Marrón. Multi-Camera Sensor System for 3D Segmentation and Localization of Multiple Mobile Robots. *Sensors*, 10(4):3261–3279, 2010.

[22] M. Mitchell. *An introduction to genetic algorithms*. The MIT press, 1998.

[23] A. Mittal and L.S. Davis. Visibility analysis and sensor planning in dynamic environments. *Computer Vision-ECCV 2004*, pages 175–189, 2004.

[24] A. Mittal and L.S. Davis. A general method for sensor planning in multi-sensor systems: Extension to random occlusion. *International Journal of Computer Vision*, 76(1):31–52, 2008.

[25] Nikhil Naikal, Avideh Zakhor, and John Kua. Image augmented laser scan matching for indoor localization. Technical report, EECS Department, University of California, Berkeley, Mar 2009.

[26] F.P. Preparata and M.I. Shamos. *Computational geometry: an introduction*. Springer, 1985.

[27] I.M. Rekleitis. A particle filter tutorial for mobile robot localization. In *International Conference on Robotics and Automation*, volume 3, 2003.

[28] M.I. Ribeiro. Kalman and extended kalman filters: Concept, derivation and properties. *Institute for Systems and Robotics, Lisboa Portugal*, 2004.

[29] A.C. Schultz and W. Adams. Continuous localization using evidence grids. In *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*, volume 4, pages 2833–2839. IEEE, 2002.

[30] S. Se, D. Lowe, and J. Little. Local and global localization for mobile robots using visual landmarks. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 414–420. Citeseer, 2001.

[31] H. Surmann, A. N
"uchter, and J. Hertzberg. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45(3-4):181–198, 2003.

[32] I.E. Sutherland, R.F. Sproull, and R.A. Schumacker. A characterization of ten hidden-surface algorithms. *ACM Computing Surveys (CSUR)*, 6(1):1–55, 1974.

[33] K.A. Tarabanis, P.K. Allen, and R.Y. Tsai. A survey of sensor planning in computer vision. *IEEE transactions on Robotics and Automation*, 11(1):86–104, 1995.

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

[35] R.C. Veltkamp and M. Hagedoorn. 4. State of the Art in Shape Matching. *Principles of visual information retrieval*, page 87, 2001.

[36] R. Wahlde, N. Wiedenman, W.A. Brown, and C. Viqueira. An Open-Path Obstacle Avoidance Algorithm Using Scanning Laser Range Data, 2009.

[37] T. Weise. Global Optimization Algorithms–Theory and Application. *URL: http://www. it-weise. de, Abrufdatum*, 1, 2008.

[38] Stergios I. Roumeliotis Y and George A. Bekey. Segments: A layered, dual-kalman lter algorithm for indoor feature extraction. In *In Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[39] K. Yabuta and H. Kitazawa. Optimum camera placement considering camera specification for security monitoring. pages 2114–2117, 2008.

[40] C. Ye and J. Borenstein. Characterization of a 2D laser scanner for mobile robot obstacle negotiation. In *IEEE International Conference on Robotics and Automation, 2002. Proceedings. ICRA'02*, volume 3, 2002.