Department of Electronics and Telecommunications (DET)

Master's Degree in Mechatronic Engineering

Visual Place Recognition as a solution for the Kidnapped Robot Problem

Supervisors:	Candidate:
Prof. Marcello CHIABERGE	Simone RICCIARDELLI
Dott. Chiara BORETTI	
Dott. Marco AMBROSIO	
Dott. Andrea EIRALE	
Dott. Andrea OSTUNI	
Dott. Alessandro NAVONE	

Academic Year 2023/2024

Alla mia famiglia

Abstract

The kidnapped robot is a well-known problem in mobile robotics; it is a special case of the global localization problem, which arises when the robot is physically moved to another location without its knowledge, or gets lost in the environment. Kidnapped Robot Problem is usually studied and tested to evaluate the performance of localization algorithms since none of them can guarantee not to fail. In this thesis, we propose a solution in order to solve the kidnapped robot problem in dynamic and similar environment based on Visual Place Recognition (VPR), Density-Based Spatial Clustering of Applications with Noise (DBSCAN) to initialise the position of the robot and a particle filter to correct the pose. The VPR algorithm is based on using a Convolutional Neural Network (CNN) to extract global features from images and recognise the place where the robot is located, selecting a set of best poses using a similarity measurement algorithm. The DBSCAN is used to filter out isolated and small clusters of selected poses and initialise the position of the robot with the centroid of the largest cluster. A particle filter is then applied to correct the robot's pose. The proposed solution is fully implemented in Robot Operating System (ROS) using Python as the main programming language. It is first tested in a simulated environment in the Gazebo platform and then in a real scenario at PIC4SeR (PoliTo Interdepartmental Centre for Service Robotics). The results show that the proposed solution is able to correctly relocalise the robot after the kidnapping event.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the realm of mobile robotics, the problem of localization is a fundamental one. Localization is the process of calculating the pose of robot in a given map. In this field, we cannot avoid the problem of failure, since we are in what is called "probabilistic robotics": the robot can only estimate his actual pose; this means it is also possible that the robot can think to know where it is located in the environment, while it is not. This is called *Kidnapped Robot Problem* and it concerns the case in which the robot is moved to a different location in the environment, while it is operating. In this case, the robot has to recognize that it has been "kidnapped" and correct his global pose. Based on the sensors implemented, there are many ways to solve this problem. It is not so easy to develop a robust and reliable solution in all possible scenario. For outdoor scenario, GPS is a good choice to solve the problem; for indoor scenario, systems based on 2D lidar (Light Detection and Ranging) have gained popularity thanks to accuracy of the sensor: however, they have some limitations due to similar environment and the presence of dynamic objects. Since every type of sensors has its own limitations, it is possible to combine them to obtain a more robust and reliable solution; this is the realm of sensor fusion, which has been growing strongly in recent years. The aim of this thesis is to develop a robust and reliable solution for the kidnapped robot problem. The solution is based on a lidar-camera system. The robot detects the kidnapping using a method based on 2D lidar data and relocalize the robot in the map using Visual Place Recognition with which robust visual features from RGB images, acquired by a camera, are extracted by a Convolutional Neural Network (CNN) to recognize the place and the particle filter of the Monte Carlo localization to correct the pose through 2D lidar and odometry sensors. The system has been tested in a real environment at PIC4Ser. The thesis is organized as follows:

- *Chapter 1*: a brief introduction about the kidnapped robot problem, explaining the aim of the thesis and the structure of the document.
- *Chapter 2*: overview on robot localization theory and on the state-of-the-art about the kidnapped robot problem.
- *Chapter 3*: in this chapter a detailed explanation about the methodology used and the system developped is given.
- *Chapter 4*: description about the software used to design the system and the hardware used to test it. A brief explanation about how ROS2 framework works and Gazebo simulation environment is given.
- *Chapter 5*: results obtained from the experimental work are shown and discussed
- *Chapter 6*: conclusions and future works are presented.

Chapter 2

Robot Localization

1 Introduction to Probabilistic Robotics

Probabilistic robotics [32] is an approach to incorporate uncertainty in perception and action in the field of autonomous robot navigation, based on the idea of using probability theory. So, instead of relying just on a "best guess", we consider information based on probability distribution that let us consider all possible sources of uncertainty: environment, robots, sensors, models and computation. An example where probabilistic robotics is applied is *mobile robot localization*. Mobile robot localization concerns the estimation of the robot's pose relative to a reference frame that is linked to a map of the environment. We use the robot's belief using a probability density function and then, through sensor data and probabilistic paradigm, the robot updates its belief. The localization will be explored in more detail. Additionally, probabilistic robotics allows to use flexible models because robots can adapt to changing conditions without having to completely redesign their navigation algorithms.

2 Gaussian Filters

Gaussian filters are fundamental tools used in probabilistic robotics to estimate the state of a dynamic system in the presence of noise and uncertainty; in fact, they are widely used in the localization and navigation of mobile robots. Gaussian filters are based on the idea that beliefs are represented by a multivariate normal distribution:

$$p(\mathbf{x}|\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right) \tag{2.1}$$

where:

- $\mathbf{x}$ is the state vector with dimension d.
- μ is the mean vector, that has the same dimension of the state vector.
- $\mathbf{\Sigma}$ is the covariance matrix; it is symmetric and positive-semidefinite. Its dimension is the state vector dimension squared.

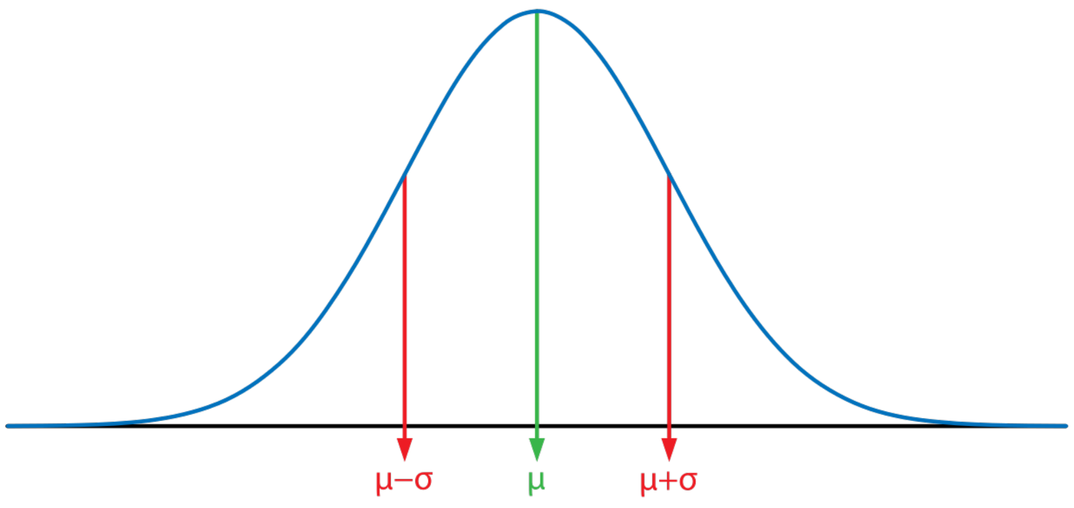


Figure 2.1: Univariate gaussian distribution [36]

2.1 Kalman Filter

Kalman filter , also known as Linear Quadratic Estimator, was invented by Rudolf E. Kalman in 1960. It evaluates the state of a *linear dynamical* *system* using a series of measurements acquired over time, instead of using just one measurement [33]. The mathematical representation of a linear dynamical system is given by the following formulae:

$$\begin{cases} x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \\ z_t = C_t x_t + \delta_t \end{cases} \tag{2.2}$$

where x is the state vector, u is the control vector, z is the output vector, A,B and C are matrices, while ϵ is the process noise and δ is the measurement noise. Both noises are white and indipendent

$$p(\epsilon) \sim \mathcal{N}(0, R_t) \tag{2.3}$$

$$p(\delta) \sim \mathcal{N}(0, Q_t) \tag{2.4}$$

It is a widely used and extremely powerful tool, useful in different applications. It is also particularly effective in addressing the following problems:

- **State estimation**: we can use it to estimate an unobservable variable of the dynamical system, like estimating the position and velocity of a robot using sensor data.
- **Future prediction**: based on the dynamical model of the system and his past estimates, we can estimate the current state.
- **Sensor fusion**: it can integrate data coming from different sensors to increase the accuracy of the state estimate. This capability is particularly useful in applications where multiple sources of information are available, such as autonomous vehicle localization using GPS, inertial sensors, and cameras.
- **Filtering noise**: use a combination of weighted state estimates and observations, reducing uncertainty over time. So, the system has more reliable and accurate estimates.

It is composed of two phases: *prediction* and *measurement update*: during the prediction phase the system estimates the future state using the past belief taking into account also the process noise, while in the measurement update, the filter updates the estimation using available information and sensor noise.

Algorithm 1 Kalman filter algorithm for linear dynamic system [33]

```
1: procedure KALMAN FILTER($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2:     ▷ $\mu_{t-1}$ is the initial state
3:     ▷ $\Sigma_{t-1}$ is the initial uncertainty covariance matrix
4:     ▷ $u_t$ is the control input
5:     ▷ $z_t$ is the actual measurement
6:     $\bar{\mu}_t = A_t\mu_{t-1} + B_t u_t$                      ▷ Predicted state estimate
7:     $\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + R_t$                ▷ Predicted estimate covariance
8:     $K_t = \bar{\Sigma}_t C_t^T(C_t\bar{\Sigma}_t C_t^T + Q_t)^{-1}$    ▷ Kalman gain
9:     $\mu_t = \bar{\mu}_t + K_t(z_t - C_t\bar{\mu}_t)$              ▷ Updated state estimate
10:    $\Sigma_t = (I - K_t C_t)\bar{\Sigma}_t$                    ▷ Updated estimate covariance
11:    return $\mu_t, \Sigma_t$        ▷ Return the updated state and covariance
12: end procedure
```

- *Prediction* (line 6-7): based on old state estimate μ_{t-1}, state-transition matrix A and actual control input u_t, we can calculate the posterior belief of the state and of its covariance matrix.

- *Kalman Gain* (line 8): it is a matrix that determines how much the state estimate should be corrected based on the measurement. It is calculated using the predicted state covariance, the measurement matrix C, the measurement noise covariance Q and the predicted state covariance.

- *Measurement update* (line 9-10): the state estimate is updated using the Kalman gain and the difference between the actual measurement and the predicted measurement, called *innovation.*

The image 2.2, illustrate a cycle step of the Kalman filter: in (a) we see the initial belief of the state with a gaussian distribution, in (b) we see in bold the gaussian distribution of the measurement, while in (c) we see the posterior belief calculated by the algorithm; in (d) we see the belief of the state after motion to the right with a wider distribution caused by uncertainty in the motion model, and in (e) we have the new measurement and in (f) the new posterior belief. We can see that the posterior belief is a compromise between the motion model and the measurement model, and the uncertainty is reduced over time.

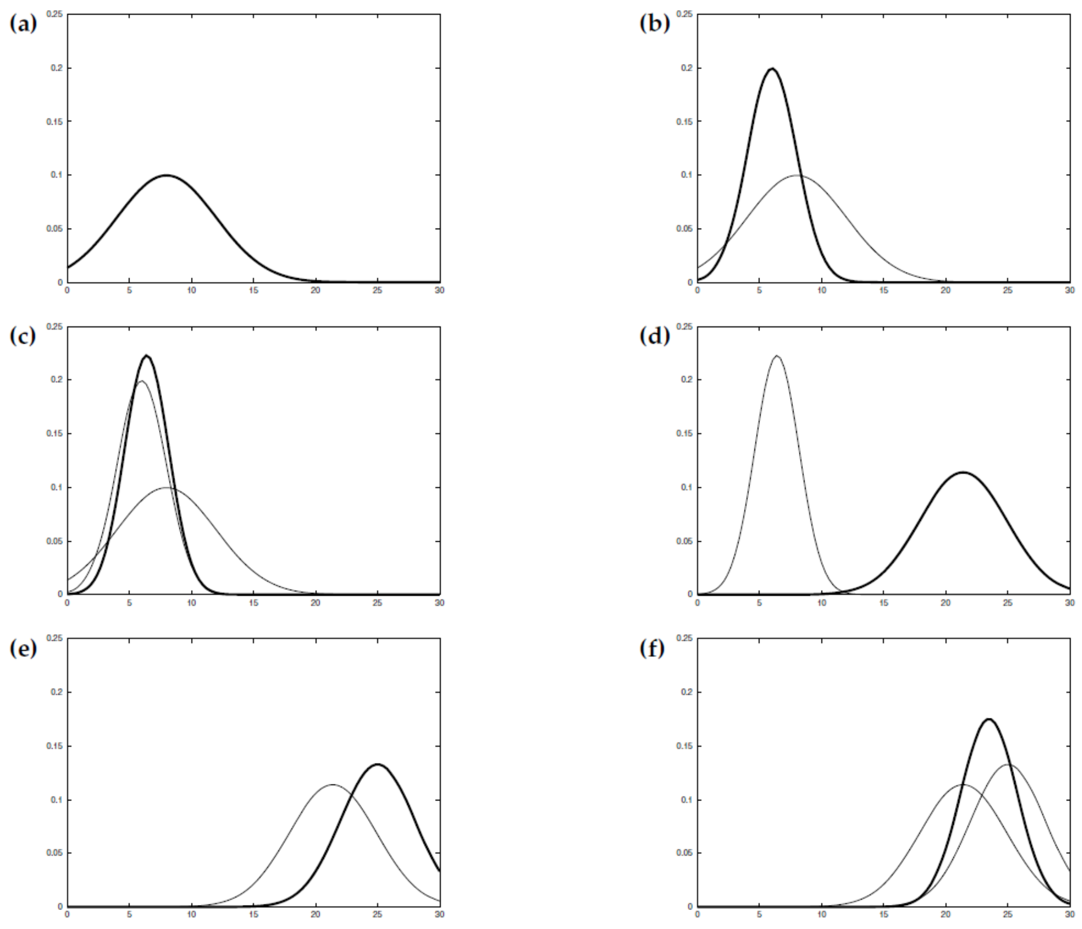


Figure 2.2: Illustration of the Kalman filter procedure

2.2 Extended Kalman Filter

The Kalman filter is a powerful tool, but his main limitation is that it can only be applied to linear systems. In most of the robotic applications, the system is non-linear, so the Kalman filter cannot be used; a simple example is when the robot moves in a circular trajectory. The Extended Kalman Filter (EKF) [33] is a solution to this problem; it is a non-linear version of the Kalman filter, and it is based on the idea of linearizing the non-linear system using a first-order Taylor expansion at each time step, before applying the Kalman filter. So, the first assumption is that the dynamical system is non-linear, and the second assumption is that the non-linear functions are differentiable:

$$\begin{cases} x_t = g(u_t, x_{t-1}) + \epsilon_t \\ z_t = h(x_t) + \delta_t \end{cases} \tag{2.5}$$

Essentially, we replace the matrices A, B and C with Jacobian matrices G_t and H_t:

$$G_t = \frac{\partial g(u_t, x_{t-1})}{\partial x_{t-1}} \tag{2.6}$$

$$H_t = \frac{\partial h(x_t)}{\partial x_t} \tag{2.7}$$

The EKF is a powerful tool, but the main limitation is that it is not always guaranteed to converge to the true state of the system, especially when the non-linearities are strong, risking to lead to very unreliable estimates.

Algorithm 2 Extended Kalman Filter algorithm for non-linear dynamic system [33]

```
    procedure EXTENDED KALMAN FILTER($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$)
2:      ▷ $\mu_{t-1}$ is the initial state
        ▷ $\Sigma_{t-1}$ is the initial uncertainty covariance matrix
4:      ▷ $u_t$ is the control input
        ▷ $z_t$ is the actual measurement
6:      $\bar{\mu}_t = g(u_t, \mu_{t-1})$                         ▷ Predicted state estimate
        $\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$             ▷ Predicted estimate covariance
8:      $K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$    ▷ Kalman gain
        $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$          ▷ Updated state estimate
10:     $\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$                 ▷ Updated estimate covariance
        return $\mu_t, \Sigma_t$              ▷ Return the updated state and covariance
12: end procedure
```

3 Particle Filter

The main problem with gaussian filters is that they are based on the assumption that the belief is a unimodal gaussian distribution, but in many cases, the belief is multimodal or non-gaussian. An alternative to Gaussian technique are *nonparametric filters* [33], that don't rely on fixed form of the posterior belief; they can approximate posteriors with a finite set of parameters: as the number of parameters tends to infinity, the nonparametric filter converges to the true posterior. The particle filter is the most used nonparametric filter in the field of robotics, thanks to his nature, especially in the field of mobile robot localization. It is based on the idea of representing the posterior with a set of samples, called *particles*, drawn from the probability distribution, denoted by:

$$X_t := \{x_{1_t}, x_{2_t}, \dots, x_{M_t}\}$$

where M is the number of samples in the particle set X_M. So, the denser a region of the state space is populated by particles, the more likely it is that the true state falls into this region. The particle filter algorithm is based on the following steps:

Algorithm 3 Particle Filter Algorithm [33]

```
1: procedure PARTICLE FILTER($X_{t-1}, u_t, z_t$)
2:     $\bar{X}_t = X_t = \emptyset$
3:     for $m = 1$ to $M$ do
4:         sample $x_t^{[m]} \sim p(x_t|u_t, x_{t-1}^{[m]})$
5:         $w_t^{[m]} = p(z_t|x_t^{[m]})$
6:         $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
7:     end for
8:     for $m = 1$ to $M$ do
9:         draw $i$ with probability $\propto w_t^{[i]}$
10:        add $x_t^{[i]}$ to $X_t$
11:    end for
12:    return $X_t$
13: end procedure
```

In detail:

- *Prediction* (line 4): we predict a new state $x_t^{[m]}$, that actually is a new particle, generated from the previous state $x_{t-1}^{[m]}$ and the control input u_t.
- *Update* (line 5-6): we include in the particle set the measurement calculating for each particle the *importance factor* (or weight), that represent the probability of the measurement given the state. So the set of weight particles approximate the posterior distribution.

- *Resampling* (line 8-10): starting from the weighted particle set (posterior belief), we draw new particles with replacement; the probability of drawing a particle is proportional to its importance factor. So, particles with lower weights (those that are less consistent with the measurement) are less likely to be drawn, than particles with higher weights.

So, relatively to Gaussian filters, advantages and disadvantages of particle filters are:

- **Advantages**: we can represent multimodal and non-gaussian distributions, and it is more robust than Kalman Filter and Extended Kalman Filter in non-linear dynamical systems.
- **Disadvantages**: in order to represent the belief in a proper way, we need a large amount of particles, that increase computational complexity. However, researchers developped techniques to adapt the number of particles online, based on the uncertainty of the belief.

4 Mobile Robot Localization

One of the field of autonomous navigation where probabilitic robotics is applied is the *localization*. In the localization, the robot has to estimate its current pose based on information that come from external sensors. It is considered the most important problem in mobile robotics, because it is a fundamental capability for a robot to be able to navigate in an environment. The pose information can be inferred from measurement data and control data given a map of the environment. We can classify the localization problem into three different sub-problems [25] [34]:

- **Pose Tracking**: it concerns the continuous tracking of the robot pose while it moves in the environment. It is fundamental for the robot to be aware of its initial position at the beginning of the trajectory; in this phase, the system continuously updates the robot's pose combining actual sensors informations with previous one. Most of the time odometry sensors, such as encoders and IMU, are used, but they accumulate errors over time; pose tracking algorithms needs to handle the uncertainty to ensure accurate pose estimation. The pose uncertainty is usually approximated with a unimodal distribution (e.g. Gaussian), so Kalman Filter approaches are widely used in this phase.

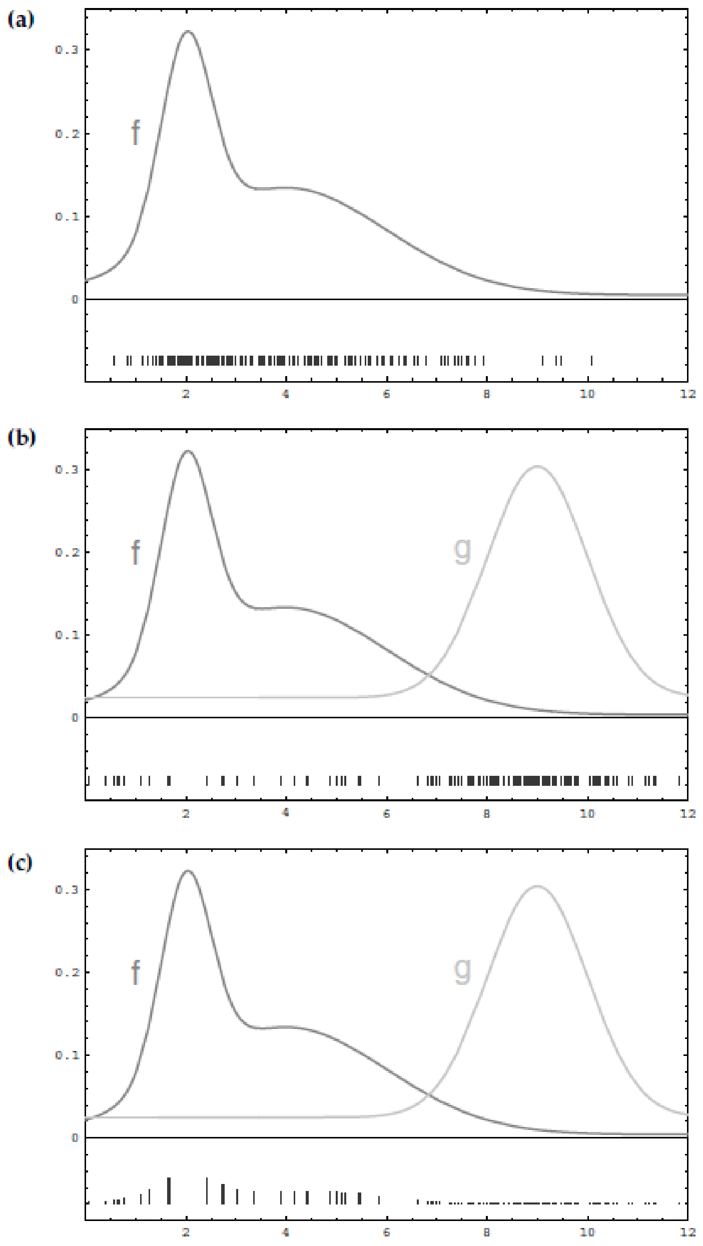


Figure 2.3: Illustration of the Particle filter procedure

- **Global Localization**: in this localization problem, the initial position is unknown. Robots usually use sensor data, for instance, coming from lidar, cameras or inertial sensors, and they are compared with a map of the environment, previously built. It is a fundamental problem to be solved, especially in dynamical environments. The uncertainty of the robot's pose is usually approximated with a multimodal distribution, so Kalman Filter approaches are not suitable to solve this localization problem; in this case, particle filters are widely used or even more advanced techniques, like SLAM (Simultaneous Localization and Mapping) [13] that will be discussed more in detail later.

- **Kidnapped Robot Problem**: it is a sub-problem of the global localization, but it is even more difficult because it arises when, during operation, the robot gets teleported to another location; so, essentially thinks to know where it is in the map, while it does not. This problem is studied because most of the localization algorithm cannot be guaranteed never to fail, so it is essential for the robot to be able to recover from global localization failures.

Another element that has impact on the localization problem is the environment. Environments can be *static* or *dynamic*:

- **Static environments**: they are environments where all surrounding elements remain relatively unchanged over time. Static nature of an environment simplifies the autonomous navigation because the robot can rely on stable references (obstacles, features, landmarks), making the behaviour of the robot more predictable.
- **Dynamic environments**: they are environments where surrounding elements can change position over time in an unpredictable way. Robots has to adapt their plans and actions, like avoiding dynamic obstacles. Localization could become an hard problem, especially kidnapped robot, since the robot cannot rely on stable references anymore; so in recent years, researchers are developping more advance sensor fusion systems in order to achieve robust localization handling unforeseen situations.

4.1 Monte Carlo Localization

Monte Carlo Localization (MCL) is a probabilistic algorithm used to solve the global localization problem, and it is based on the idea of using a particle filter to represent the posterior belief of the robot's pose. It was introduced by Dieter Fox in 1999 [9]. The algorithm is based on the idea of representing

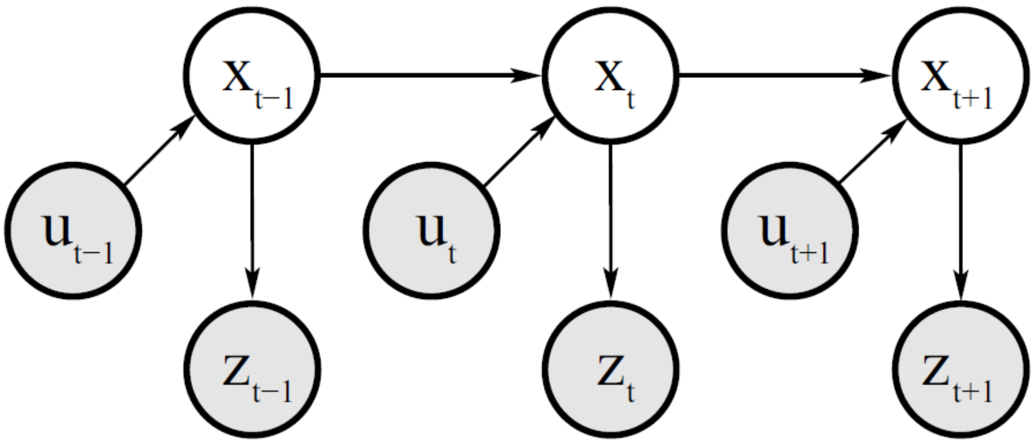


Figure 2.4: Graphical representation of mobile robot localization steps [35]

the posterior belief with a set of particles, that are drawn from the probability distribution and was born to solve the limitations of the Kalman Filter and Extended Kalman Filter in non-linear and multimodal environments. It can bu used for local and global localization.

Algorithm 4 Monte Carlo Localization Algorithm [9]

```
 1: procedure ALGORITHM MCL($\chi_{t-1}$, $u_t$, $z_t$, $m$)
 2:     static $w_{\text{slow}}, w_{\text{fast}}$
 3:     $\bar{\chi}_t = \chi_t = \emptyset$
 4:     for $m = 1$ to $M$ do
 5:         $x[m]_t =$ sample_motion_model$(u_t, x[m]_{t-1})$
 6:         $w[m]_t =$ measurement_model$(z_t, x[m]_t, m)$
 7:         $\bar{\chi}_t = \bar{\chi}_t + \langle x[m]_t, w[m]_t \rangle$
 8:         $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M} w[m]_t$
 9:     end for
10:     for $m = 1$ to $M$ do
11:         draw $i$ with probability $\propto w[i]_i$
12:         add $x[i]_i$ to $\chi_t$
13:     end for
14:     return $\chi_t$
15: end procedure
```

The motion model describes the robot's motion in the environment; it is used to predict the new state of the robot given the previous state and the control input u_t. It is generally based on odometry sensors, like encoders and IMU. The measurement model describes the relationship between the robot's pose and the sensor measurements and is used to calculate the importance factor (or weight) of each particle. In order to globally localize, the MCL initializes all over the map a set of particles, and then the robot moves in the environment; the algorithm updates the belief of the robot's pose using the motion model and the measurement model. The image 2.5 shows the procedure used to globally localize on a map; as we can see in (a) we have the uniform initial belief with particles spread all over the map and thanks to the motion and measurement model, the belief is updated in (b) and (c) and the robot is able to localize itself. So, the MCL is good at solving global localization, but cannot recover from robot kidnapping and global localization failures, especially in large environments; this is because after some time, the particles will converge to the true pose, so it is difficult to recover to the correct pose if it far away from the true one. This problem is solved by the *Adaptive Monte Carlo Localization* (AMCL).

Adaptive Monte Carlo Localization

Adaptive Monte Carlo Localization (AMCL) is a variant of the classic Monte Carlo Localization that let to solve the kidnapped robot problem by adding random poses in the map: this behaviour it is regulated by two new parameters, w_{slow} and w_{fast}: they represent decay rates for the exponential filters that estimate the long-term, and short-term, averages, respectively.

4.2 Simultaneous Localization and Mapping

In the field of robotics, during autonomous navigation in an environment, the robot has to know the map of it and it should be accurate enough depending on the kind od operation required. However, in many cases, the map is not available, so the robot has to build it while navigating. So, it is important to have a system able to map the environment and at the same time to localize the robot in it. This problem is called *Simultaneous Localization and Mapping* (SLAM) [13]. Actually, there are three different types of maps that can be built:

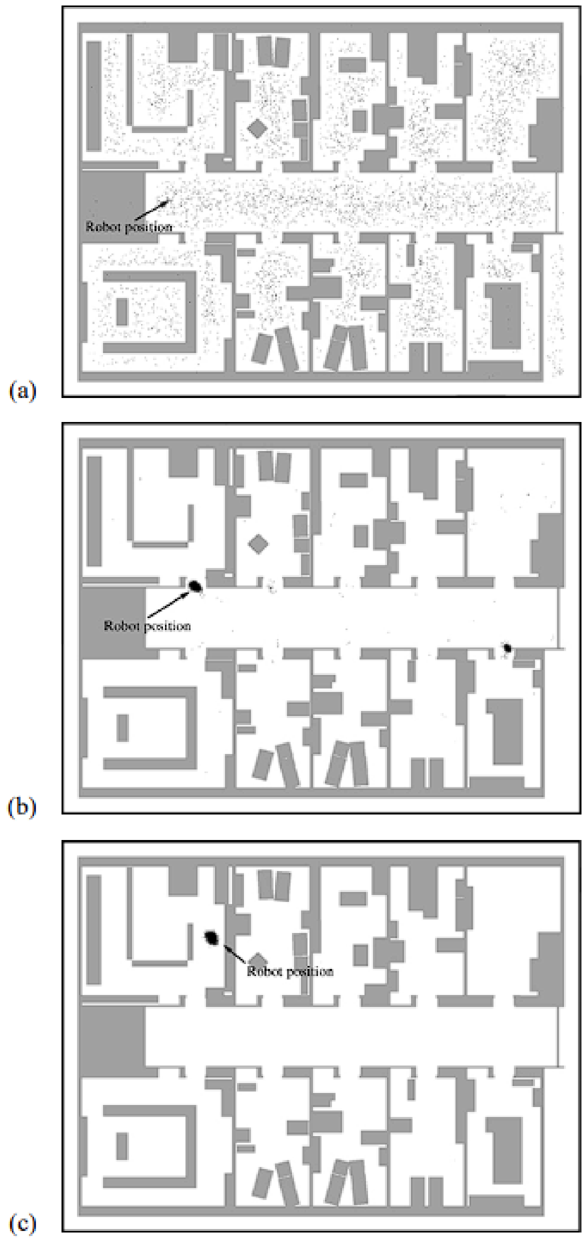


Figure 2.5: Global localization using Monte Carlo Localization algorithm [31]

- **Metric Map**: it is a discretized map that represents the environment in a metric way. The environment is divided in grid cells, and each cell is associated with a probability of being occupied. It is usually built using

Algorithm 5 Adaptive Monte Carlo Localization Algorithm [31]

```
1: procedure ALGORITHM AMCL($\chi_{t-1}$, $u_t$, $z_t$, $m$)
2:     static $w_{\text{slow}}, w_{\text{fast}}$
3:     $\bar{\chi}_t = \chi_t = \emptyset$
4:     for $m = 1$ to $M$ do
5:         $x[m]_t = \text{sample\_motion\_model}(u_t, x[m]_{t-1})$
6:         $w[m]_t = \text{measurement\_model}(z_t, x[m]_t, m)$
7:         $\bar{\chi}_t = \bar{\chi}_t + \langle x[m]_t, w[m]_t \rangle$
8:         $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M} w[m]_t$
9:     end for
10:    $w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$
11:    $w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$
12:    for $m = 1$ to $M$ do
13:        with probability $\max\{0.0, 1.0 - \frac{w_{\text{fast}}}{w_{\text{slow}}}\}$ do
14:            add random pose to $\chi_t$
15:        else
16:            draw $i \in \{1, ..., N\}$ with probability $\propto w[i]_t$
17:            add $x[i]_t$ to $\chi_t$
18:        end with
19:    end for
20:    return $\chi_t$
21: end procedure
```

distance sensors, like lidar and sonar. It is generally used for navigation and avoid obstacles because the robot can plan a safe path through free cells, avoiding occupied ones.

- **Topological Map**: it is a map that represents the environment as a collection of significant places (nodes) linked between them. This kind of map is useful to represent the environment in a more abstract way, so it is useful for high-level navigation tasks.

- **Feature Map**: it is a map that represents the environment highlighting salient features, relevant for navigation. Features can be something recognizable in the environment, like corners, edges, lines, and can be extracted with computer vision algorithms.

Each of this maps has its own advantages and disadvantages, and the choice

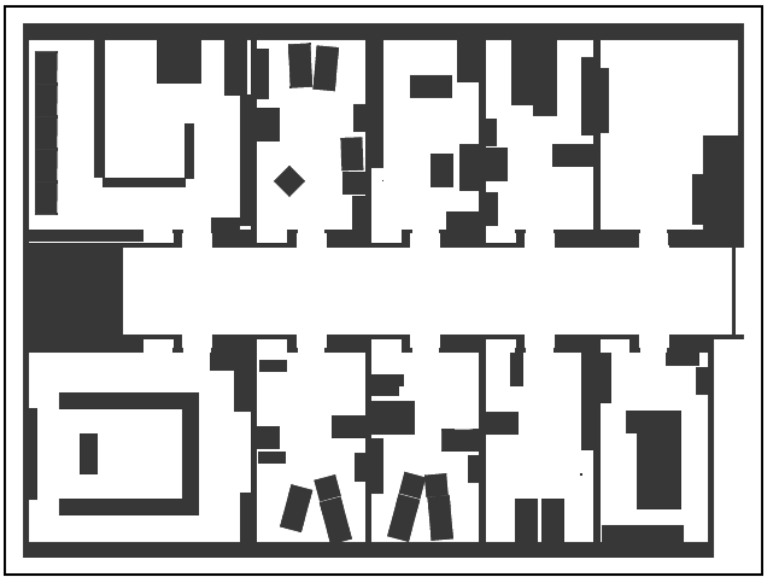


Figure 2.6: Example of metric map[32]

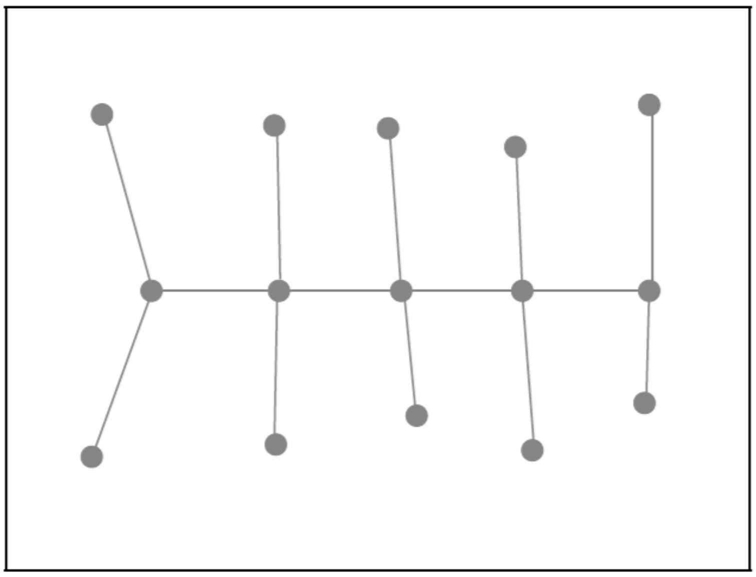


Figure 2.7: Example of graph-like topological map of an environment [32]

of the map depends on the kind of operation required. They can also be used in combination, for instance, a metric map can be used for low-level navigation tasks, while a feature map can be used for recognize places. In

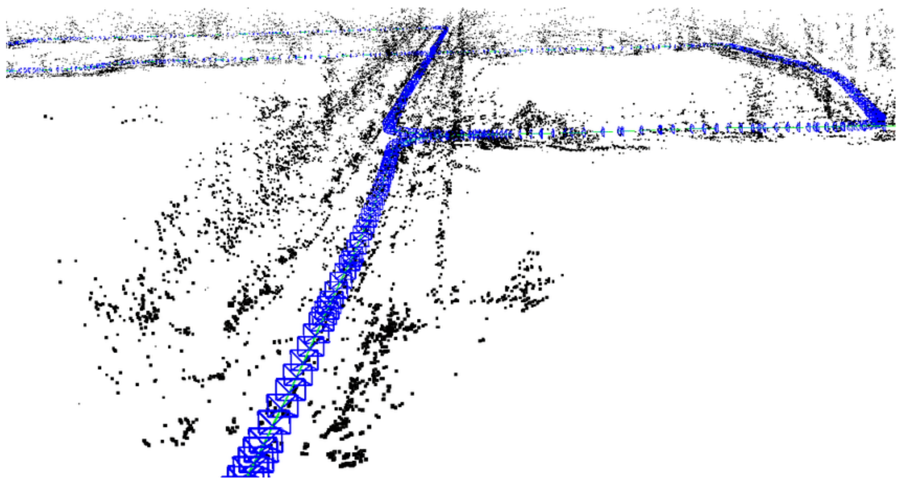


Figure 2.8: Example of feature map built with ORB-SLAM. The black dots are ORB features [23]

SLAM problem, the robot has to estimate its pose and the map of the environment at the same time, using exteroceptive sensors, like cameras, lidar, sonar, and proprioceptive sensors, like encoders and IMU. From a probabilistic point of view, the SLAM can be classified in two different form:

- **Online SLAM**: in this type of SLAM, the robot estimates the posterior belief over the actual pose along with the map

$$p(x_t, m|z_{1:t}, u_{1:t}) \tag{2.8}$$

- **Full SLAM**: in this type of SLAM, the robot estimates the posterior belief over the entire trajectory along with the map

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{2.9}$$

The SLAM problem is a complex problem, and it is difficult to solve because of the following reasons:

- **Data Association**: it is the problem of associating the sensor measurements with known information; in SLAM, it concerns the correlation between the actual measurement and the estimated pose in map. It is a basic prerequisite to solve for successful implementation of SLAM.
- **Loop Closure**: it is a sub algorithm of SLAM useful for recognizing a place already visited by the robot [16]. It is a powerful tools because

it allows to correct the drift of the robot's pose over time and map misalignment errors. It is fundamental for large environments because sensors can accumulate errors over time and the map could not be accurate enough. It can be achieved using different techniques, including geometric constraints and appearance-based methods.

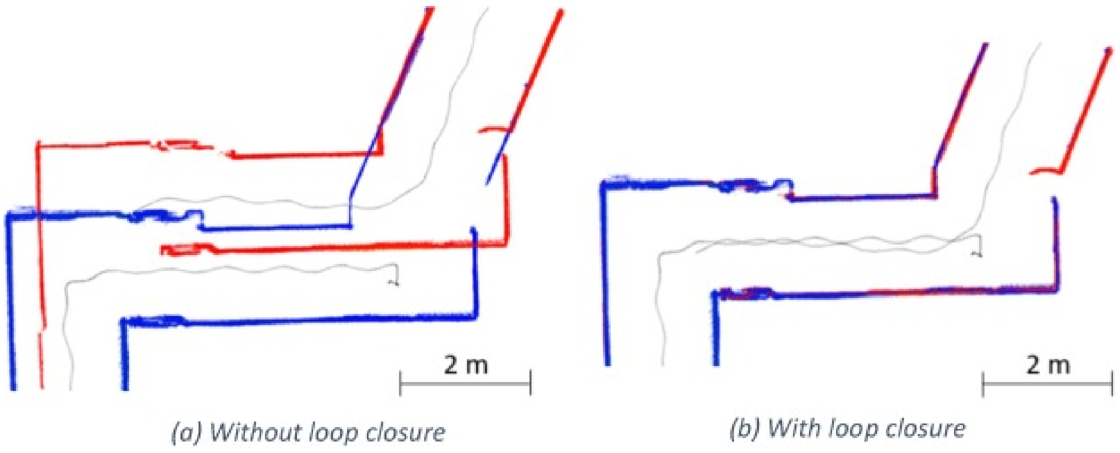


Figure 2.9: Comparison between map without loop closure (a) and map with loop closure (b) [16]

5 Sensor Fusion

5.1 Motivation for Sensor Fusion

In the field of robotics and artificial intelligence, the ability to perceive the environment is a fundamental capability for a robot to be able to navigate and localize itself in the environment. From autonomous vehicles to industrial robots to drones, the accuracy and completeness of informations gathered from sensors are crucial for the success of the operation. However, processing data from indivisual sensors can be challenging, especially when the environment is dynamic and unpredictable. In order to achieve robust and reliable perception and reduce, it is necessary to use multiple sensors, and to combine their data in a coherent way. This process is called *sensor fusion.* W. Elmenreich in his paper [10] defines *"Sensor Fusion is the* *combining of sensory data or data derived from sensory data such that the resulting information is in some sense better than would be possible when these sources were used individually"*. System that employ sensor fusion are called *multisensor systems* and they expect different benefits over a single sensor systems.

A physical sensor measurement generally suffers from:

- **Uncertainty**: it is the lack of complete knowledge that makes it hard to predict the outcome of a particular event. It can be caused by different factors, like noise, bias, drift, missing data or feature (e.g. occlusions).
- **Imprecision**: individual sensors can be limited in their ability to measure the environment
- **Sensor Failure**: breakdown of a sensor leads to a loss of information.
- **Temporal Constraints**: some sensors can be limited in their ability to measure the environment in real-time because they could need some time to process and transmit data.
- **Spatial Constraints**: some sensors can be limited in their ability to measure the environment in a wide area because of their limited range.

So, sensor fusion system has to be able to handle these problems in order to provide a reliable and accurate perception of the environment. The main advantages of sensor fusion are:

- **Redundancy**: it is the ability to use multiple sensors to measure the same quantity, so if one sensor fails, the system can rely on the other sensors.
- **Extended spatial and temporal coverage**: sensors can conver wider area and one sensor can eprform measurement when other sensors cannot.
- **Robustness against interference**: the system can be more robust against noise and interference.
- **Increased confidence**: having different sensors measuring the same quantity can increase the confidence of the system.

5.2 Categories of Sensor Fusion

Sensor fusion can be mainly cathegorized in two different ways [6]: three-level categorization and Dasarathy's categorization[8].

Three-level categorization

In this fusion process, we can distinguish three different levels:

1. *Low-level fusion*: it is often called *data fusion* and concerns the fusion on several sources of raw data to produce new ones. It is the most common form of sensor fusion and it is used to reduce noise and uncertainty in the data. A common example of low-level fusion is the fusion of odometry data from encoders and IMU through Kalman Filter.

2. *Feature-level fusion*: also called *information-level* fusion, it concerns the fusion of features extracted from raw data. It is used to combine different features extracted from different sensors after processing raw data.

3. *Decision-level fusion*: it is the highest level of fusion and it concerns the fusion of decisions made by different sensors. Generally this level include methods like voting, fuzzy logic and statistical methods.

Dasarathy's categorization

Dasarathy's categorization [8] starts from the fusion categorization described above, but consider also type of input and output. We distinguish five different categories:

1. *Data-in-Data-out*: in this category, fusion happens immediatly after the data acquisition and the output is a new set of data, more robust and reliable.

2. *Data-in-Feature-out*: at this level we still fuse raw data after acquisition, but the output is a set of features extracted to describe the environment in a more abstract and compact way.

3. *Feature-in-Feature-out*: in this category, we fuse already processed and extracted information.

4. *Feature-in-Decision-out*: we fuse a set of features of the environment to provide a decision.
5. *Decision-in-Decision-out*: in this category, we fuse decisions made by different sensors to provide a final one that is more robust.

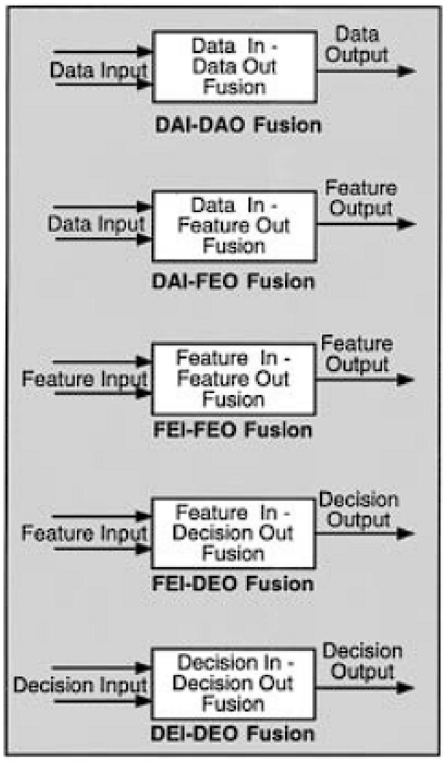


Figure 2.10: Dasarathy's categorization of sensor fusion [8]

6 State-of-the-Art of Kidnapped Robot Problem

To introduce the state-of-art of the kidnapped robot problem, we need a deep understanding of challenges and the main proposed solutions. The kidnapped robot problem is a sub-problem of the global localization, so it has been studied since the beginning of the autonomous navigation field. Over the years, researchers have developed several different techniques to solve this problem, depending on the type of sensors used in the system and the type of environment where the robot operates. Each sensors has its own advantages and disadvantages, and the choice of the sensor depends on the kind of operation required. Various sensors play fundamental roles in providing useful data for the robot to perceive its surroundings and determine its location accurately. Some commonly used sensors in the field of robotics are:

- **Laser Range Finders** (Lidar): this sensors emit many laser beams at high frequency and measure the time it takes to reflect off objects and return to the sensor to determine the distance to the obstacles. Thei high precision makes them suitable for mapping and localization. They are also versitile and can be used in many different environments, including indoor and outdoor and during daytimes and nighttime operations.

- **Cameras**: they are used to capture images of the environment and process them to extract useful information, like features, landmarks, and obstacles. There are also more advanced cameras that implement depth sensing, like stereo cameras and RGB-D cameras, that can be used to exploit visual information and depth information at the same time. They are widely used in the field of computer vision and they are suitable for localization in service robotics. Generally they are used in combination with other sensors, like lidar and IMU, to provide a more robust and reliable perception of the environment.

- **Inertial Measurement Units** (IMU): they consist of accelerometers and gyroscopes and are used to measure the robot's acceleration and angular velocity. So, they provide informations about the robot's motion in the environment. Generally, they are used in combination with wheel encoders to provide odometry data

- **Wheel Encoders**: they count the number of rotations of the wheels and with information of the wheel diameter, they can be used to estimate the distance traveled by the robot. They are used to provide odometry data, usually in combination with IMU.

- **Global Positioning System** (GPS): it is a satellite-based navigation system that provides location and time information. It is widely used in outdoor environments, but it has some limitations, like the inability to work in indoor scenario.

- **Wireless Sensors**: in the field of robotics localization, wireless sensors are used to estimate the robot's position through the acquisition of signal strength (RSSI) from different wireless access points. They are mainly used in indoor environments.

In the following sections, we will discuss the main techniques used to solve the kidnapped robot problem, depending on the type of sensors used in the system. Since global relocalization has been studied since the beginning of the autonomous navigation field, we will just discuss the main techniques.

6.1 Lidar-based Relocalization

Lidars are by far the most used sensors in autonomous navigation and localization. Many localization and relocalization algorithms are based on lidar data, because of their high precision and accuracy. The most common techniques used to solve the kidnapped robot problem is based on Monte Carlo Localization (MCL) and its variant, Adaptive Monte Carlo Localization (AMCL). The Monte Carlo Localization algorithm, with the help of a relocalization trigger mechanism, can recover from kidnapping by reinitializing the particle set, spreading particles all over the map. The Adaptive Monte Carlo Localization algorithm, instead, is a variant born to solve the limitations of the classic MCL in relocalization task since it adds random poses in the map based on the decay rates for the exponential filters that estimate the long-term and short-term averages. Basically, they rely on lidar and odometry data to estimate the robot's pose taking multiple measurements $Z_t \triangleq \{z_{k=1}, \ldots, z_t\}$ enhancing robot's pose with robot moving. So they can have some limitations especially in dynamic, geometrically similar and large-scale environments. In figure 2.11, we can see the reinitialization of the particle cloud in MCL relocalization systems. As we can notice, the particle cloud is spread all over the map, so we need a large amount of particles to cover it completely, leading to a high computational complexity and large amount of memory and time; if the scenario is even more complex, like a large-scale environment and with similar geometric features, it's easy for the system to fail. In recent years, researchers have developed more

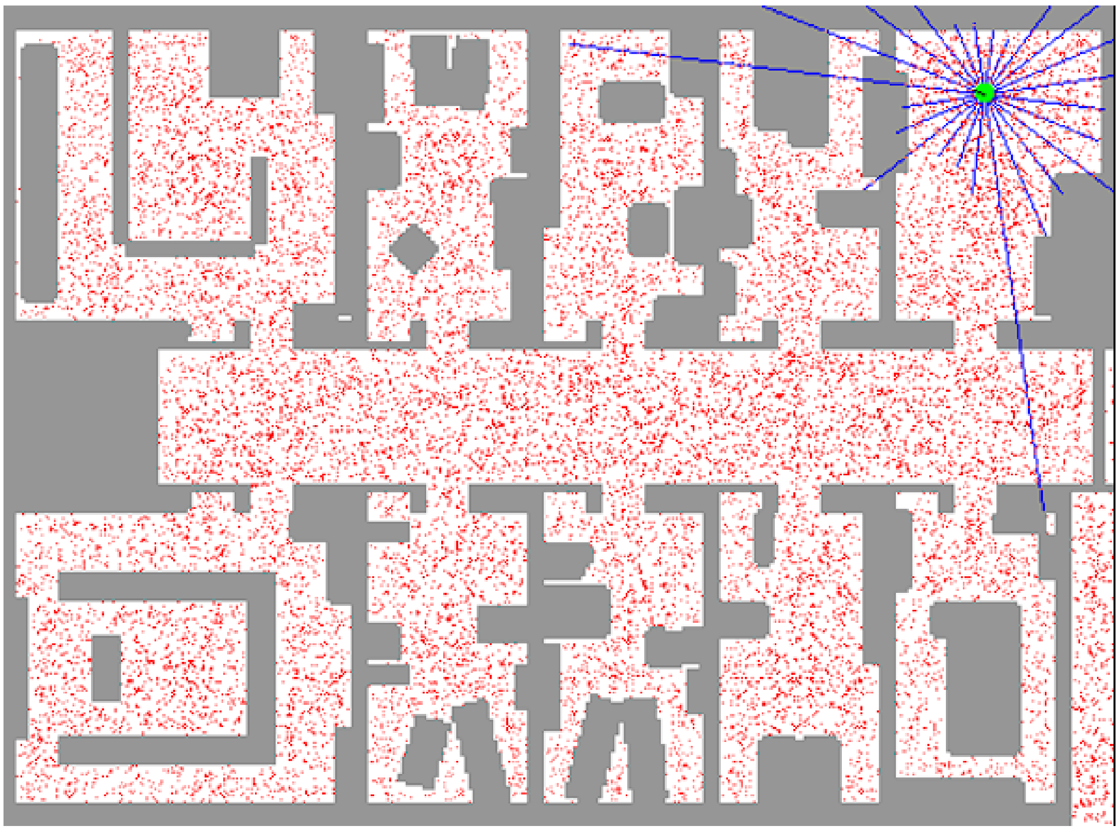


Figure 2.11: Reinitialization of particle cloud in MCL relocalization systems [31]

advanced techniques to solve the kidnapped robot problem, like the use of visual and wireless sensors to assist the lidar in the relocalization task.

6.2 Visual-based Relocalization

Visual sensors, like cameras, are widely used in the field of robotics, especially in the field of computer vision. They are used to extract richer features and landmarks from the environment and to provide a more robust and reliable perception of it. In the field of relocalization, visual sensors are used to recognize places to recover from kidnapping. The main techniques used to solve the kidnapped robot problem are based on visual SLAM and appereance-based method. Visual SLAM is a very powerful tool based on the idea of exploiting visual features to estimate the robot's pose and the map of the environment at the same time using exteroceptive sensors, like cameras, and proprioceptive sensors, like encoders and IMU. When robot is "kidnapped", visual data get collected from camera sensors and the system extract distinctive features of the environment and compare them with the map to recognize the place. One of the most used VSLAM algorithm is ORB-SLAM: it is a real-time SLAM library for Monocular, Stereo and RGB-D cameras that implement a Bag-of-Words place recognition module based on DBoW2 [12] to relocalize the robot; when the robot lost tracking, the system extract ORB features from the camera images, convert the frame into a bag of words and compare it with the database to select keyframe candidates and then perform a PnP [18] pose optimization algorithm to correct the camera pose. The appereance-based relocalization approach focus on the idea of recognizing a place already visited directly comparing real-time images with a database of the environment to identify similar positions and deduce the robot's current pose. Since images are generally high-dimensional data, the system has to reduce the dimensinality extracting global and/or local features from the images and compare them with the database. The main advantage of this approach is that it is more robust against changes in the environment, like illumination changes of dynamic objects. Visual Place Recognition (VPR) is one of the most used appereance-based strategy used for relocalization.

6.3 Wireless-based Relocalization

Lidar-camera systems are the most used in the field of robotics and autonomous navigation because of their high precision, accuracy and compactness. However, they have some limitations; lidar has problem in localization in symmetrical environment, while cameras in featureless ones. In recent years, researchers have developped systems that use wireless sensors to assist the lidar and the camera in the relocalization task. The methodologies that implement wireless sensors are based on the idea of estimating the robot's position through the acquisition of signal strength (RSSI) from different wireless access points, using techniques like fingerprinting. The basic principle of WiFi fingerprinting involve involves the analysis of the power levels of the received signal from each access point, and the comparison of these levels with a database of signal strength maps. In their paper [37], Song Xu and Wusheng Cho proposed a system that is able to globally relocalize the robot using WiFi fingerprinting and Adaptive Monte Carlo Localization; after the robot is kidnapped, with a classical relocalization trigger mechanism the robot detect the kidnap and start to collect WiFi signal strength data from the environment and compare them with the database using KNN to recognize the place and recover from kidnapping. Instead, in [24], researchers proposed a similar method analyzing the RSSI data with Recurrent Neural Networks (RNN), achieving better results in terms of accuracy and robustness than with KNN. So, WiFi techniques can only be used in indoor environments, but they are more robust against changes in the environment, like illumination changes and dynamic objects; the main limitation is that they require many access point to properly cover the environment, so the cost of the system can be high, especially in large environments.

Chapter 3

Methodology

In this chapter a detailed description of the implemented methodology will be provided. The system's goal is to robust relocalize a robot in a dynamical and similar scenario without any external infrastructure (like WiFi or UWB), using only a 2D lidar and a camera. The system is based on the use of visual place recognition as a coarse relocalization method and then a particle filter is exploited to correct the pose of the robot. The main algorithm is divided in four main parts: section 1 describes the mapping of the environment with a slam algorithm and the creation of an image database for visual place recognition; section 2 describes the implementation of the visual place recognition system to roughly estimate the pose of the robot; section 3 describes the use of a DBSCAN [11] in order to filter out outliers in the pose estimation and section 4 describes the use of a particle filter to a fine relocalization correcting the pose.

1 Mapping for Environment Representation

In order to relocalize a mobile platform, the system needs to have a representation of the environment whereby matching the current view of the robot with the stored views. The system implemented involves a 2D lidar and a camera. The 2D lidar is used to obtain an occupancy grid map where the robot localize itself and the camera is used to create an image database for visual place recognition.

1.1 Occupancy Grid Mapping

The occupancy grid mapping is a technique used to create a map of the environment using a 2D lidar. The map is represented as a grid where each cell represents the probability of being occupied; so, basically in any occupancy grid mapping algorithm, we calculate the posterior

$$p(m_i|z_{1:t}, x_{1:t}) \tag{3.1}$$

where, m is the map, *z1:t* the set of all measurements up to time t, and *x1:t* is the path of the robot defined through the sequence of all poses. We can mathematically represent the map as:

$$m = \sum_i m_i \tag{3.2}$$

where m_i is the occupancy of the cell i. The probability for each cell to be occupied is:

$$p(m_i|z_{1:t}, x_{1:t}) \tag{3.3}$$

and based on it, we can assign "0" if the cell is free, "1" if the cell is occupied. In my case, I used the SLAM Toolbox [21] to create the occupancy grid map. SLAM Toolbox is a graph-based SLAM library for 2D and 3D environments developped in 2021 and it is already implemented in a ROS2 package.

1.2 Image Database for Visual Place Recognition

Once the map is created, we can implement the visual part of the system starting from the creation of an image database. Visual Place Recognition will be detailed explained in the next section, but essentially what we need is to associate an image with a given pose. In order to perform this task, a ROS2 node has been implemented to run in parallel with the SLAM Toolbox algorithm. In this node, the OpenCV library is used to capture and save images from the camera at a frequency of 5 Hz; at the same time, the node saves the poses of the robot in a numpy array in order to associate each

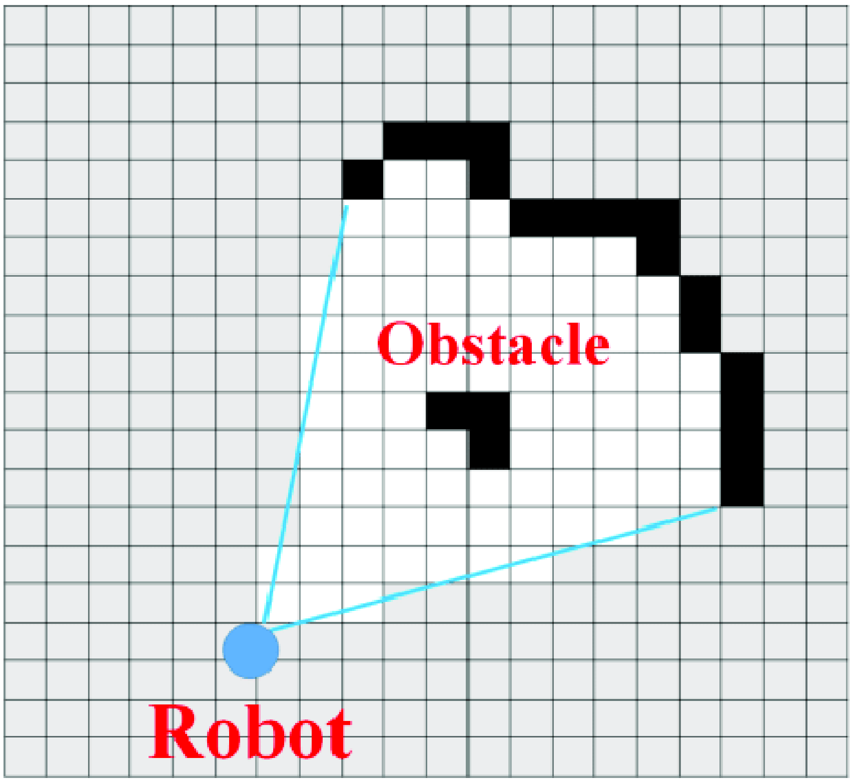


Figure 3.1: Example of an occupancy grid map built with a 2D lidar

image with a pose. As we can notice in figure 3.2, the database is really dense. In the relocalization phase we have to consider that the system has to load in memory the database, so we need to filter out some images in order to reduce the computational cost. To do so, there are basically two possible ways: the first one is to filter out images based on the euclidean distance and yaw angle difference between adjacent poses; the second one is to evaluate the similarity between adjacent images, through calculation of global features and similarity measurement. The problem with the second method is that it can lead to big holes between poses. In this work, the first method is implemented because it represents a good trade-off between computational cost and accuracy. In details, the images were filtered out when the distance of difference of yaw angle of relative pose between adjacent images are less than a given threshold.

- **Distance threshold**: this threshold is used to filter out images based on the euclidean distance between adjacent poses. The threshold has

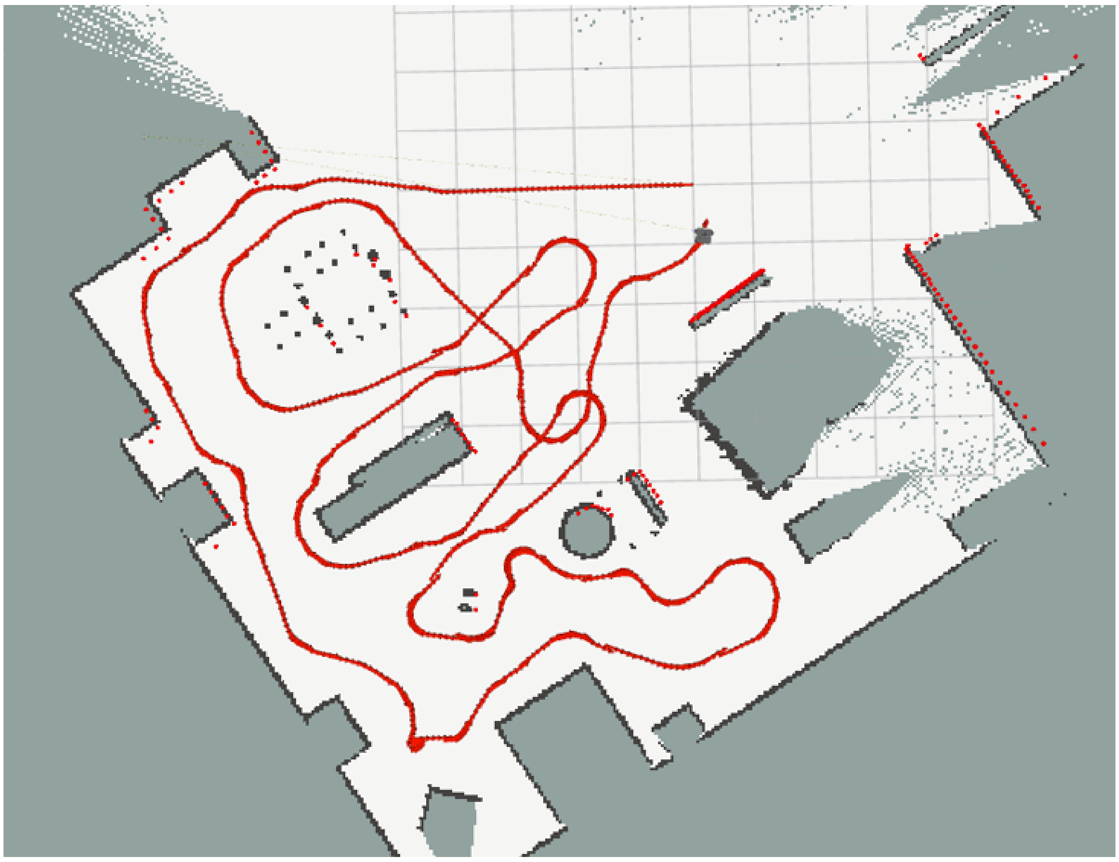


Figure 3.2: Dense image database created with the ROS2 node. The red path is composed of poses to which images are associated.

been set to 15 cm.

- **Yaw angle threshold**: this threshold is used to filter out images based on the yaw angle difference between adjacent poses. The threshold has been set to 10 degrees.

In figure 3.3 we can see the sparse database created with the ROS2 node. This filtering process is performed offline in order to evaluate the performance of the system with different thresholds. However, this task can be also done online, during the mapping phase.

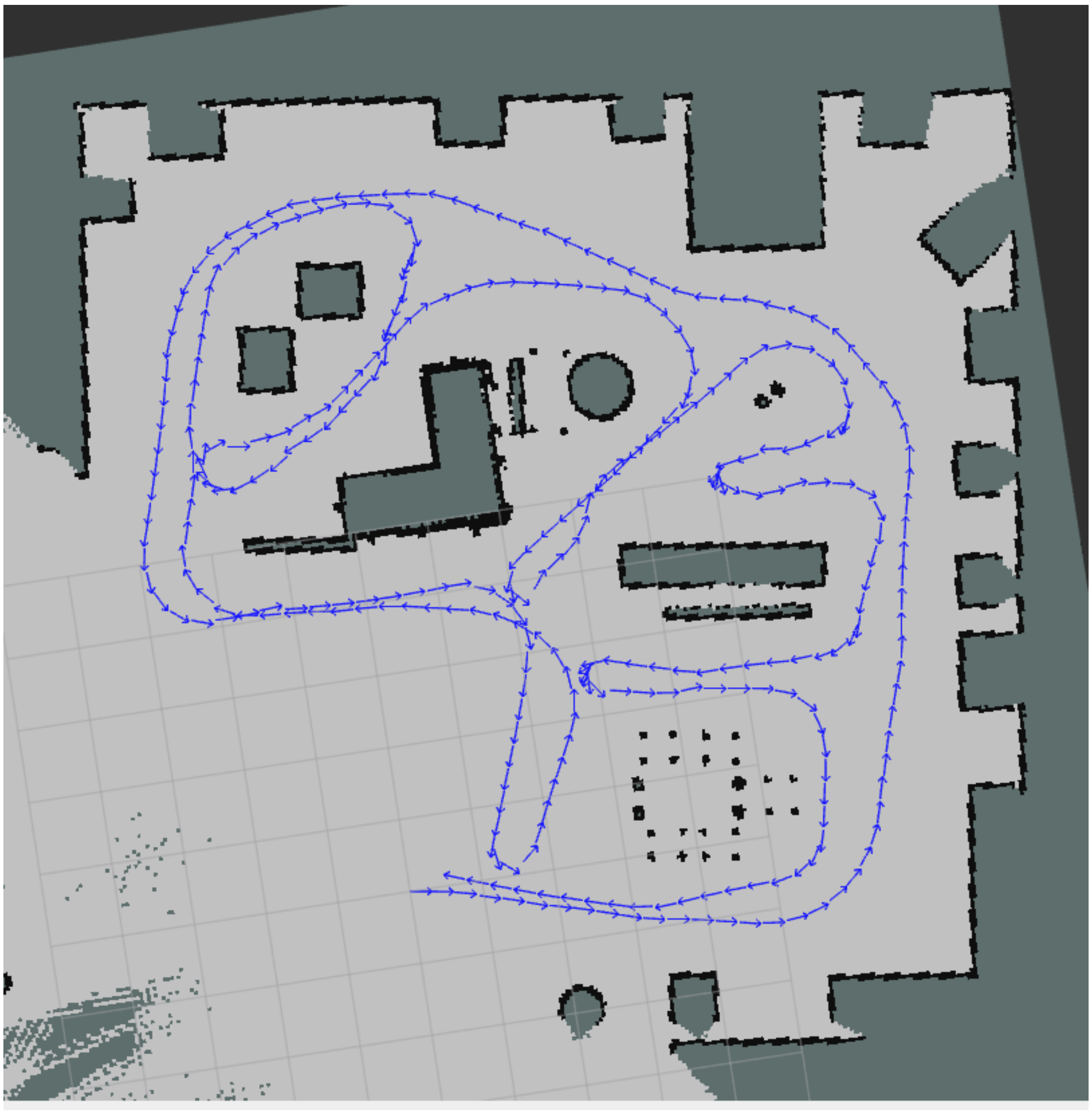


Figure 3.3: Sparse image database created with the ROS2 node. The blue arrows represent poses at which the images were taken.

2 Relocalization Trigger Mechanism

In the context of the kidnapped robot problem, where the robot is relocated in an unknow location without any prior knowledge, the systems, before the relocalization phase, needs a trigger mechanism to understand when the robot is kidnapped. So, it is crucial to select a proper trigger mechanism to detect such events and initiate actions to perform the relocalization. As discussed in the previous chapter, the kidnapped robot problem generally occurs when the localization algorithm fails, so when the actual information coming from sensors are not consistent with the map. In thesis, a Monte Carlo localization algorithm is implemented. It is based on a particle filter where each of the particles in the set is associated with a weight that represents the discrete probability distribution of the robot's pose. So, the distribution of values of the particles' weights can be used to detect the kidnapped robot scenario. There are basically three main strategy to detect the kidnapped robot problem through the analysis of the particles' weights:

- **Metric-based detector**
- **Measurement Entropy**
- **Max Current Weight**

2.1 Metric-based Detector

In 2013, Campbell [7] proposed a set of four metrics to detect the kidnapping event. It is based on the a deterministic scan matching algorithm called Normal Distributions Transform (NDT) [5] introduced by Biber and Straßer in 2003. Given two sequential point clouds P and Q, the matrix T_Q^P calculated by the NDT algorithm, represents the transformation of coordinates from frame Q to frame P and the matrix U_P^Q represents the same transformation but calculated by odometry. Considering these quantities, the four metrics are defined as follows:

1. Mean squared error (MSE) between the two point clouds defined as:

$$Q_e\left(P, Q, T_Q^P\right) = \frac{1}{n}\sum_{i=1}^{n} ||p_i - q_i||^2 \tag{3.4}$$

   where p_i and q_i are the points of the two point clouds P and Q and n is the number of overlapping points.

2. The likelihood that the NDT-transform point cloud Q is a match to the point cloud P defined as:

$$Q_s\left(P, Q, T_Q^P\right) = \frac{1}{n}\sum_{i=1}^{n} \tilde{p}\left(T_Q^P b_i\right) \tag{3.5}$$

where n is the number of point of b_i in Q.

3. The standard deviation of the least known pose degree of freedom defined as:
$$Q_h\left(P, Q, T_Q^P\right) = \sqrt{max_{i=1}^{v}\lambda_i} \tag{3.6}$$
where λ_i are the eigenvalues of NDT optimization function

4. The mean squared error of pointclouds Q transformed by T_Q^P and U_P^Q defined as:
$$Q_h\left(Q, T_Q^P, U_P^Q\right) = \frac{1}{n}\sum_{i=1}^{n} ||p_i - r_i||^2 \tag{3.7}$$

2.2 Measurement Entropy

Under the Monte Carlo localization there are two type of measurement entropy:

- The first one was defined by Choi in [38] in a topological map where we can recognize landmarks at each node. He defined the measurement entropy as:
$$H_t(p) = -\sum_{x_t^{[y]}} p\left(s_t, o_t, z_t | x_t^{[y],m}\right) \log p\left(s_t, o_t, z_t | x_t^{[y],m}\right) \tag{3.8}$$
where at each time step t, we have s_t,o_t,z_t that are the distance context to objects observed, object observed and features extracted from from them; $x_t^{[y]}$ is the state of particle y and m is the map.

- The second one is similar to the previous one, but redefined in a more simple way, calculating the sum of a particle's weights entropy:
$$H_t(p) = -\sum_{x_t^{[y]}} \omega_t^{[y]} \log \omega_t^{[y]} \tag{3.9}$$

Kidnap detection based on measurement entropy can be defined as:
$$Kidnapped_t = \begin{cases} 1 & H_t(p) \geq \pi \\ 0 & \text{Otherwise} \end{cases} \tag{3.10}$$

where π is a proper threshold value.

2.3 Max Current Weight

Like the measurement entropy, the *max current weight* is a detection strategy used in Monte Carlo localization based systems. The samples are weighted based on the likelihood of the robot's pose given the sensor measurements and the map. They reflects how closely the environmental reading provided by each sample align with the actual reading of the robot; the higher is the weight of a sample, the more likely is the robot to be in that pose. The max current weight takes just the biggest value of the importance factors and compared them with a proper threshold.

$$Kidnapped_t = \begin{cases} 1 & \max \omega_t^{[y]} < \gamma \\ 0 & \text{Otherwise} \end{cases} \tag{3.11}$$

As we can notice from figure 3.4, the max current weight is a good strategy to detect the kidnapped robot problem. When the robot is well localized the normalized max current weight is generally close to 1 and when the robot is kidnapped, since the particles are spread in the environment because the localization algorithm is not able to find a good match between the actual sensor readings and the map, the normalized max current weight is generally close to 0. This strategy is really simple, effective and light in terms of computational cost; therefore, it is the method used in this work to detect the kidnapping event and also to understand when the robot converged to a good pose after the relocalization phase.

3 Visual Place Recognition for Pose Estimation

3.1 Overview of Visual Place Recognition

Computer vision is one of the field of artificial intelligence (AI) that is growing faster in the last years. It is the field of study that deals with how computers can gain high-level understanding from digital images or videos and has various applications in robotics, autonomous vehicles, augmented reality, and many others. Visual Place Recognition (VPR) is a subfield of computer

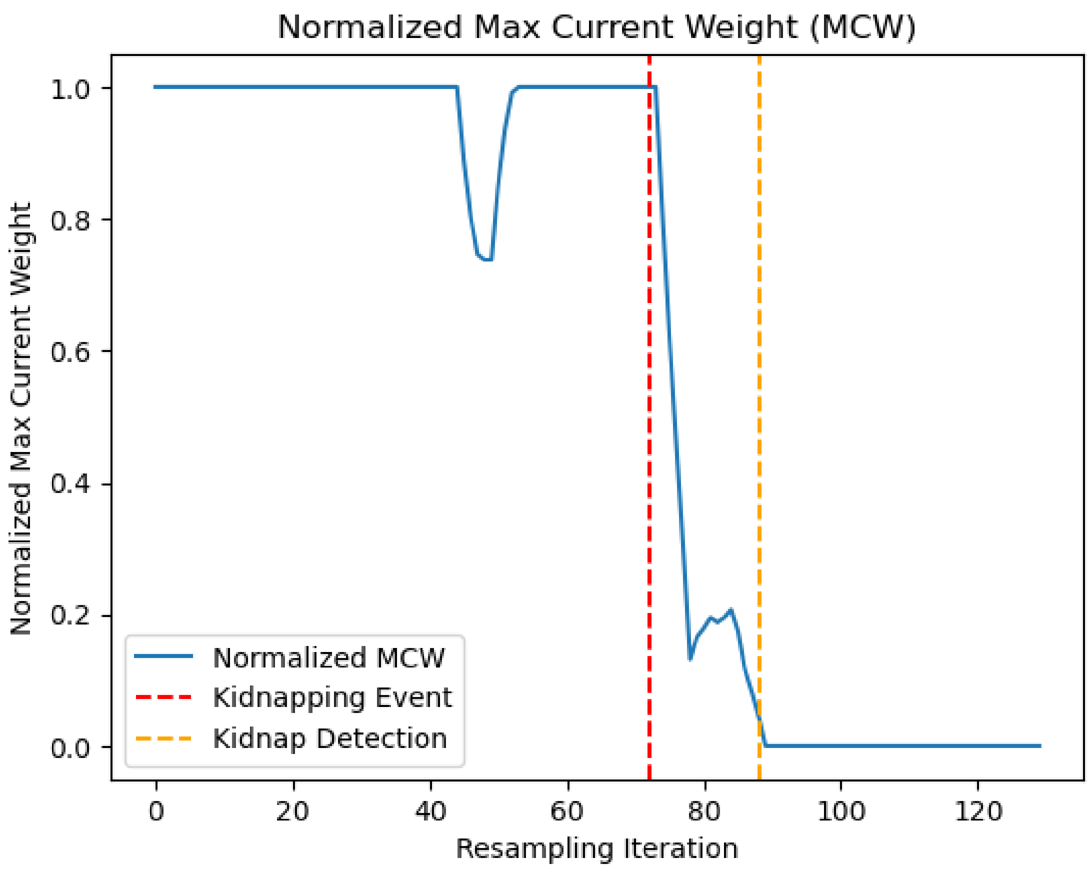


Figure 3.4: Max Current Weight with normalized values

vision that deals the recognition of places based on visual cues extracted from images. in the recent years, especially with the advent of machine learning, VPR is becoming more and more popular, playing crucial role in the field of robotics and autonomous vehicles. The main goal of VPR involve matching visual features of one or of a set of multiple images through a similarity metrics between them and a database of reference images. By leveraging advancement in deep learning and convolutional neural networks (CNNs), VPR has been improved in terms of accuracy and robustness, identifying places across different conditions, such as weather, lighting and dynamic changes. Therefore, it has great potential in robot relocalization in dynamic environments. To find matches between images, we need to extract features and calculate descriptors from them. A descriptor is typically represented as a numerical vector that describes in a more compact way the visual information of an image. Descriptors are abstractions of images calculated from raw pixel in order to be more robust against changes in viewpoint and appereance. There are two types of descriptors:

- **Local Descriptors**: they are calculated from local patches of the image and they are used to describe the local appearance of it. They encode the patches with a set of feature vectores $\{d_k | k = 1, ..., K\}$. They lead to efficient comparison between query image and database images and are generally invariant to scale and rotation. The main disadvantage consists in the fact that they may lack contextual information about the overall scene, leading to problem in place recognition in environments with similar local features. Moreover, local descriptors can be sensitive to noise, affecting reliability with varying lighting conditions and they can be computationally expensive, especially in large-scale databases. The most popular local descriptors are SIFT [20], SURF [4] and ORB [28]. In most cases they are used in combination with local feature aggregation methods like Bag of Words and Vector of Locally Aggregated Descriptors (VLAD) [15].
- **Global Descriptors**: also called *holistic descriptors.* They represent an image with just a single feature vector $d_i \in \mathbb{R}^k$, where k is the dimension of the feature vector. They are robust to dynamic changes to the scene and the retrieval process is generally faster than local descriptors. The main disadvantage is that they are sensitive to viewpoint changes. We can calculate robust global descriptors through convolutional neural networks (CNNs), like AlexNet [17], VGG [30], ResNet [14], etc...

3.2 Feature Extraction with Convolutional Neural Network for Image Representation

In the context of computer vision, "features" are specific characteristics used to identify patterns, objects or meaningful information. They can have a physical meaning, like edges and corners, but also be characteristics on pixel, intensities, statistical or characteristic based on machine learning. For instance, the HOG algorithm [22] is a popular algorithm that extract intensities information and it's used for description of edges, corners and shapes of object in an image. Another popular feature extraction algoritm is

ORB [28], that has a more abstract representation of the image. With the advent of deep learning, convolutional neural networks (CNNs) have become increasingly important and relevant in the context of feature extraction due to their complexity and robustness. CNNs are a type of neural networks based on convolutional layer that extract robust and abstract information from raw pixel of the image and dense layers of neurons used to classify the image. In the context of visual place recognition, it's extremely important to used feature extraction algorithm that are robust to changes in viewpoint and appearance, and CNNs are the best choice. In this work, a pre-trained CNN model of ResNet-50 [14] is used, that is one of the most popular convolutional neural network adopted for image representation. It has been

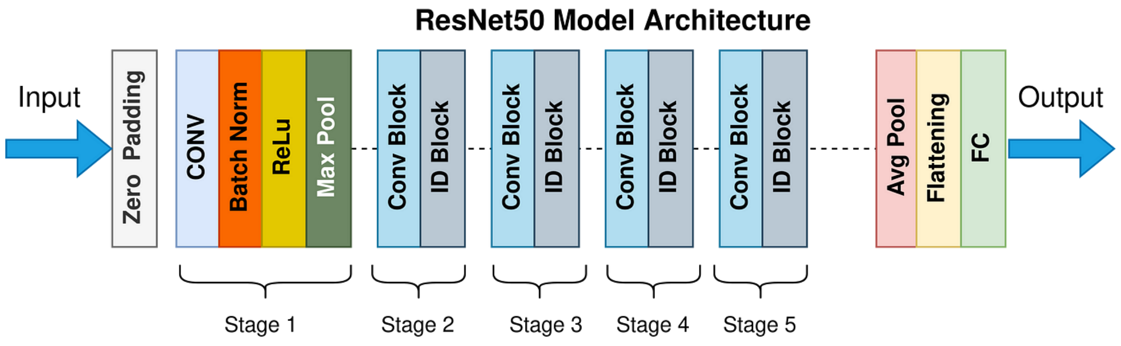


Figure 3.5: ResNet-50 architecture

tested in numerous computer vision context with great results because it is able to extract discriminative and complex features. In the system, the usage of global descriptors is preferred because they are more robust to dynamic changes in the environment, occupies less memory and perform a faster image retrieval. The main drawback is that, generally, global descriptors are less robust in viewpoint changes; in order to deal with this problem, the robot acquires images at a certain frequency while is rotating around his z-axis of about 360°. This lead to a more robust representation and place recognition.

- **Frequency of acquisition**: the robot acquires query images at 2.5 Hz leading to a total of 18 images for each rotation.
- **Rotation velocity**: the robot rotates at a velocity of 1.0 rad/s leading to a total of 6 seconds for each rotation.

In order to obtain a global descriptor from the ResNet-50 model, it was needed to crop the last layer of the model, also called *classification layer*, and pool with average the second to last layer which leads to a feature vector with dimension of 2048. The same process is done for the database images.

3.3 Similarity Measurement for Pose Estimation

Once we have calculated global descriptors for query images, we can recognize the place by evaluating similarity between queries and descriptors in the database. There are basically two main similarity measurement:

- **Euclidean Distance**: given two feature vectors d_i and d_j we can calculate euclidean distance as:

  $$d_{ij} = ||d_i - d_j||_2 \tag{3.12}$$

  The similarity is inversely proportional to the distance, so the smaller is the distance, the more similar are the images.

  $$s_{ij} = \frac{1}{d_{ij}} \tag{3.13}$$

- **Cosine Similarity**: given two feature vectors d_i and d_j we can define cosine similarity as:

  $$s_{ij} = \frac{d_i^T \cdot d_j}{||d_i||_2 ||d_j||_2} \tag{3.14}$$

The euclidean distance was actually used in the implemented system. For each of the 18 query images, we calculate the similarity with each of the database images and take the first N poses with the highest similarity. From figure 3.6 we can notice a denser cluster of red dots around the actual pose of the robot and some isolated points and small clusters as outliers that has to be taken into account. The filtering phase will be discussed in the next section.

4 Pose Filtering using DBSCAN

Since it is possible that the visual place recognition system can return some outliers in the pose selection, it is extremely important to handle them in

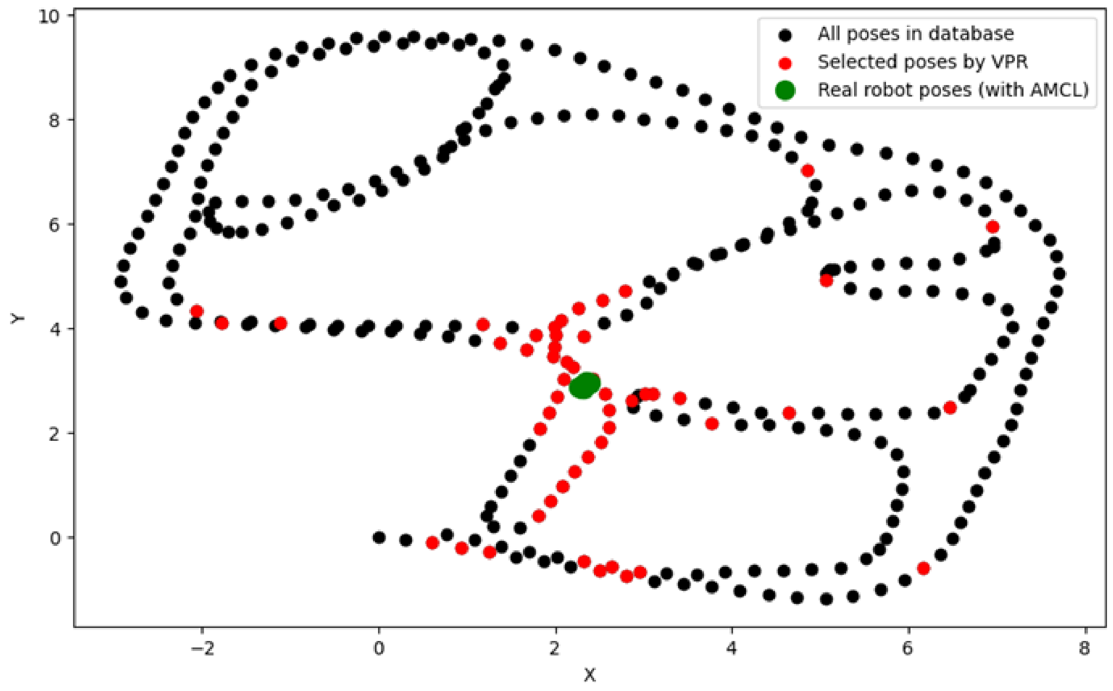


Figure 3.6: Dots represent the poses saved in the database; the red ones are the selected based on similarity measurement

order to avoid wrong relocalization. In a large number of test, we could notice that there is generally a larger group of poses around the actual one and smaller one spread all over the map. Clustering techniques were taken into account to face this problem. The most popular clustering algorithms are K-means [19] and DBSCAN [11]. The main differences between the two are:

- **K-means**: we have to specify a priori the number of clusters. Observations are assinged to the closest centroid and the centroids are recalculated at each iteration, so it generates spherical clusters. The drawback is that it cannot handle outliers since it assigns all the points to a cluster.
- **DBSCAN**: it doesn't need a priori number of clusters, so they will be determined automatically by the algorithm. The shape of clusters are not predefined and it can handle outliers.

For the type of problem to be addressed in the system, DBSCAN is considered the best choice. The DBSCAN model is based on a simple minimum density level estimation, introducing two parameters: *minPts* and *eps*. The *eps* represent the maximum distance between two point in the

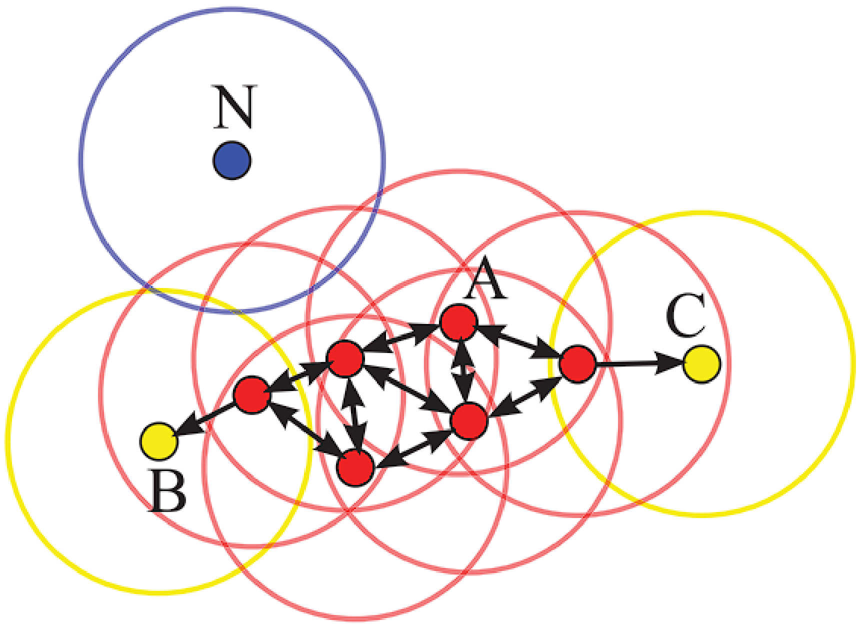


Figure 3.7: Illustration of DBSCAN model [29]: red point are core points, yellow ones are border points and blue ones are noise points

database to be considered neighbours; *minPts* is the minimum number of points within the distance *eps* in order for a point to be considered a core point. So it is important also to define the concept of core point, border point and noise point:

- **Core point**: it is a point that within the distance *eps* has at least *minPts* neighbours.

- **Border point**: it is a point that has less than *minPts* neighbours within the distance *eps*, but it is reachable from a core point.

- **Noise point**: it is a point that is not reachable from any core point and has less than *minPts* neighbours within the distance *eps*.

Algorithm 6 DBSCAN algorithm [11][29]

```
Input: Database of poses $D$, distance threshold $eps$, minimum number of points $minPts$
Output: Clusters of poses
$C \leftarrow 0$
for each point $p$ in $D$ do
    if $p$ is not visited then
        mark $p$ as visited
        $neighbours \leftarrow$ getNeighbours($p$, $D$, $eps$)
        if size($neighbours$) $< minPts$ then
            mark $p$ as noise
        else
            $C \leftarrow C + 1$
            expandCluster($p$, $neighbours$, $C$, $D$, $eps$, $minPts$)
        end if
    end if
end for
```

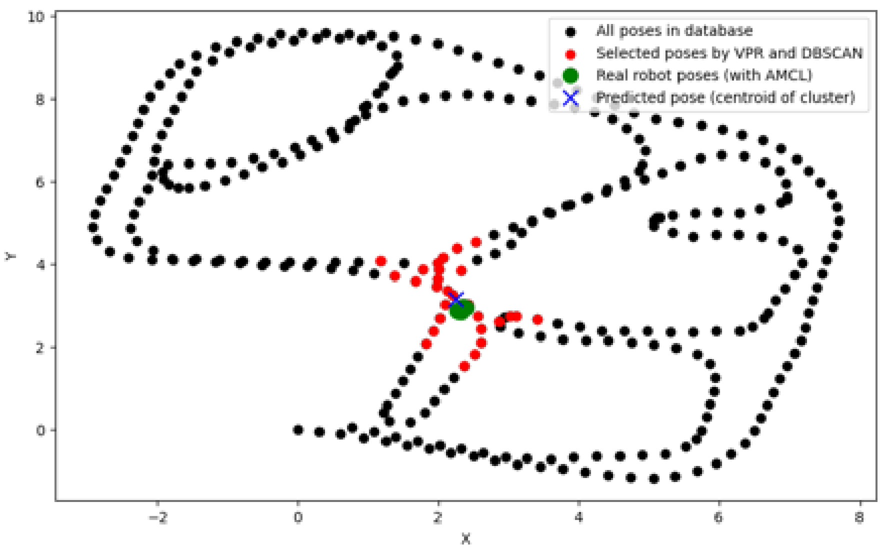


Figure 3.8: Pose distribution analysis with DBSCAN

In figure 3.8 we can notice the difference with figure 3.6. All the isolated points and small clusters has been filtered out. So, we can initialize the position of the particles in the particle filter with the centroid of the cluster.

5 Fusion in Particle Filter for Pose Correction

There are two types of relocalizations: coarse and fine relocalization. In the coarse relocalization, a system estimates approximately the pose of the robot as first step of the relocalization process. There are several type of coarse relocalization algorithm: feature matching, visual place recognition, trilateration (in context of wireless relocalization), etc... In the fine relocalization, the system has to refine the rough estimate calculated by the coarse relocalization algorithm. Some of the most popular fine relocalization algorithms are: particle filter, SLAM, bundle adjustment, etc... In the implemented system, the coarse relocalization is performed by the visual place recognition module, slightly refined by the DBSCAN clustering algorithm. Anyway, it could not still estimate a precise pose of the robot. So a fine relocalization method is required. Since the robot localize itself on a 2D map through Monte Carlo localization algorithm, the particle filter method has been chosen. Once the estimated pose is computed as the centroid of the cluster, a set of particles is initialized around it with equal covariances. As we can see in figure 3.9, the particle are spread around the estimated pose in order to cover a small part of the map where the robot could be. This method is used especially to correct the orientation of the robot, since the VPR is able to estimate the position along X-Y coordinates, but not the yaw angle. As already mentioned in previous chapter, the particle filter needs to update the particles based on the sensor readings to refine a pose. In figure 3.10 the set of particles has been updates based on the sensor readings and the estimated pose has been corrected: as we can notice, the radius of the particles has been reduced significantly and the 2D lidar point cloud matches with the occupancy grid map. Since the values of weights of particles represent the discrete probability distribution of the robot's pose, the Max Current Weight strategy can still be used to evaluate when the robot is well relocalized.

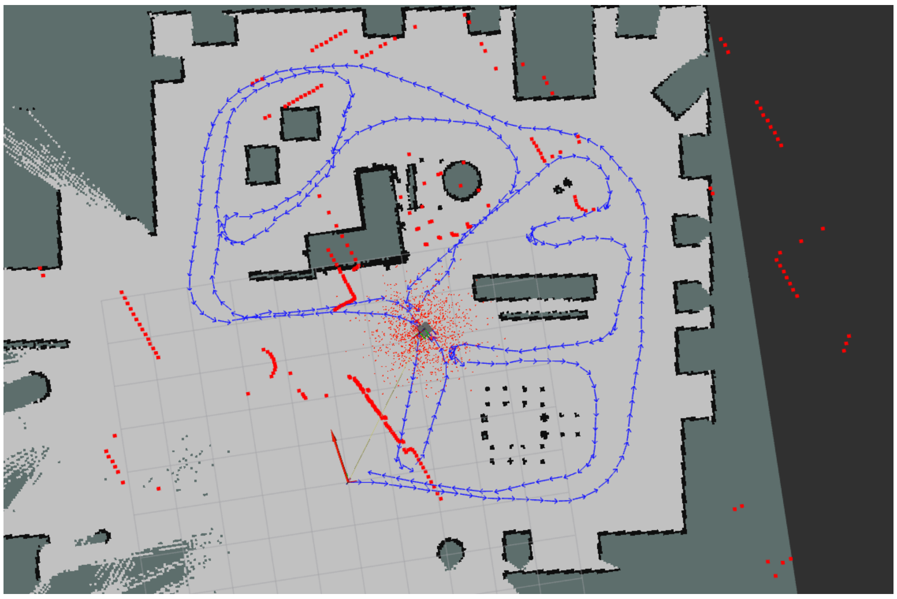


Figure 3.9: Rough estimated pose has been initialized as the centroid of the cluster and a set of particles has been generated around it

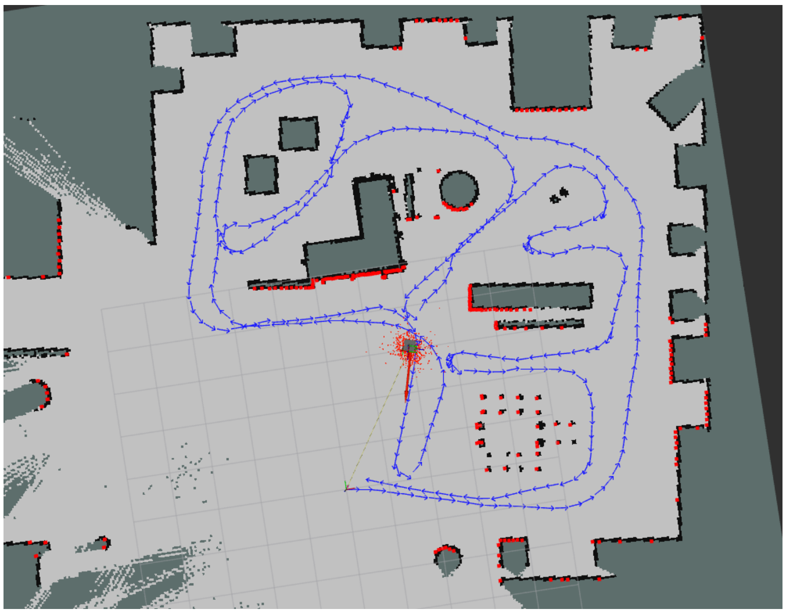


Figure 3.10: Pose of the robot has been corrected through motion model and sensor readings

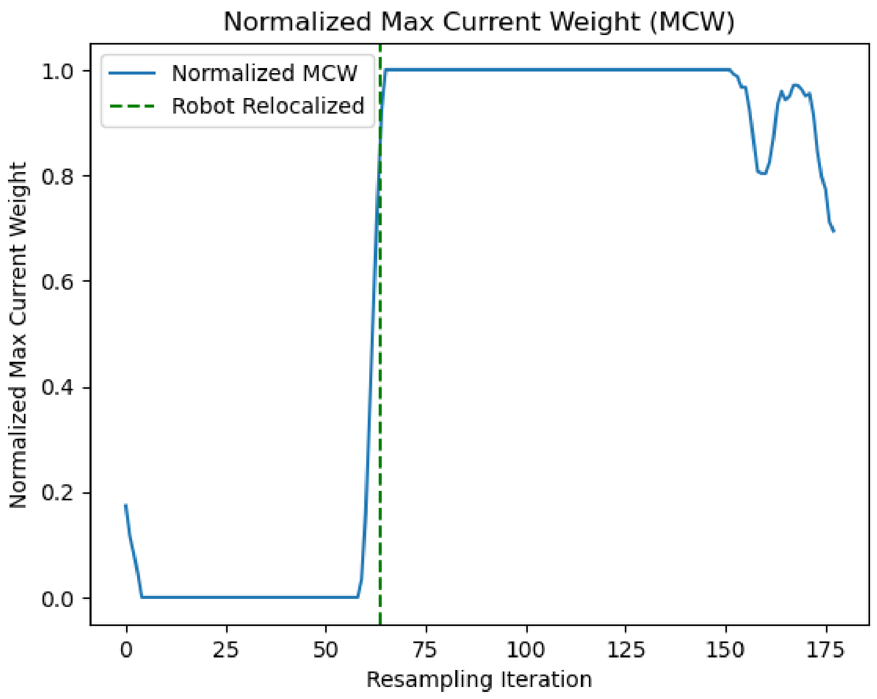


Figure 3.11: The robot has been considered relocalized when the max current weight is close to 1

Chapter 4

Software and Hardware Tools

In this chapter the sotfware and hardware tools used to test and implement this project will be discussed. The main software tools used are the Robot Operating System 2 (ROS2) and Gazebo. All the software runs on a Linux Ubuntu 22.04 LTS operating system. The system has been implemented on a Turtlebot2 Kobuki mobile robot, equipped with a Rplidar and a Realsense camera.

1 Robot Operating System 2

Programming a robot from scratch is a very complex assignment since it involves many different aspects such as hardware, software, and communication; therefore, even a simple task could take a while to be developped. Since ROS [27] was started in 2007, a lot has changed in the robotics and ROS community. ROS is a middleware that implements services, tools and libraries used to help software developers to create robotic applications that lies between hardware and real software. Robotic Operating System

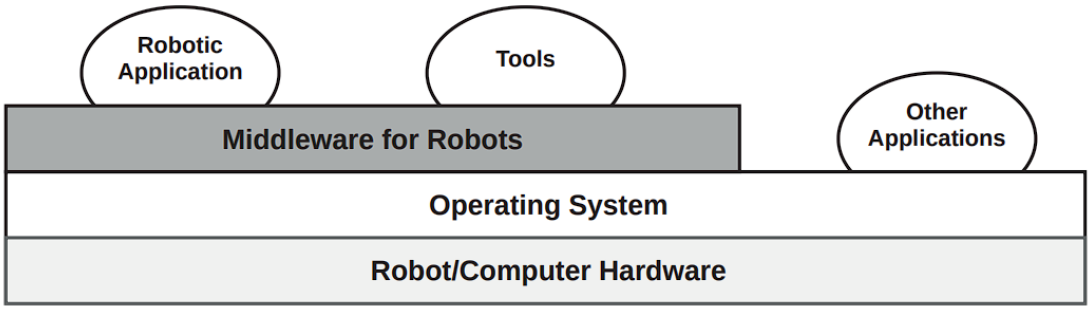


Figure 4.1: Representation of layers from hardware to higher levels of software [26]

2 (ROS2) is the second generation of ROS, it is a distributed framework for robotics that is designed to be more secure, reliable, and more flexible than its predecessor: it incorporates a fast data distribution service, meets real-time constraints and can run on different operating systems, such as Linux and Windows. Its main programming languages are C++ and Python. A brief introduction is given on how ROS2 works.

1.1 ROS2 architecture

The ROS2 architecture can be represented with a graph: the ROS graph is essentially a network of nodes and links through which nodes communicate. The graph is mainly composed by the following blocks:

- **Nodes**
- **Messages**
- **Topics**
- **Services**
- **Actions**

Nodes

Nodes represent the main building blocks of the ROS2 architecture. A node is essentially a process that performs specific operations, such as receiving data through messages and topics, publishing data, or providing services. They can be written mainly in two main programming languages: Python and C++. One or more nodes can be defined in a single executable. Nodes can interact and communicate with each other through topics or through services and actions.

Messages and Topics

Messages are simple data whose structure is defined in a file with the extension .msg. They are exchanged between nodes via topics. A topic is a channel through which nodes interface with each other in a uni-directional way; publishers and subscribers to a topic can be defined in a node. A publisher is a node that sends messages to a specific topic, while a subscriber is a node that receives messages from a particular topic. Many nodes can subscribe to the same topic and, each time a message is published, all subscribers will run the relative callback function.

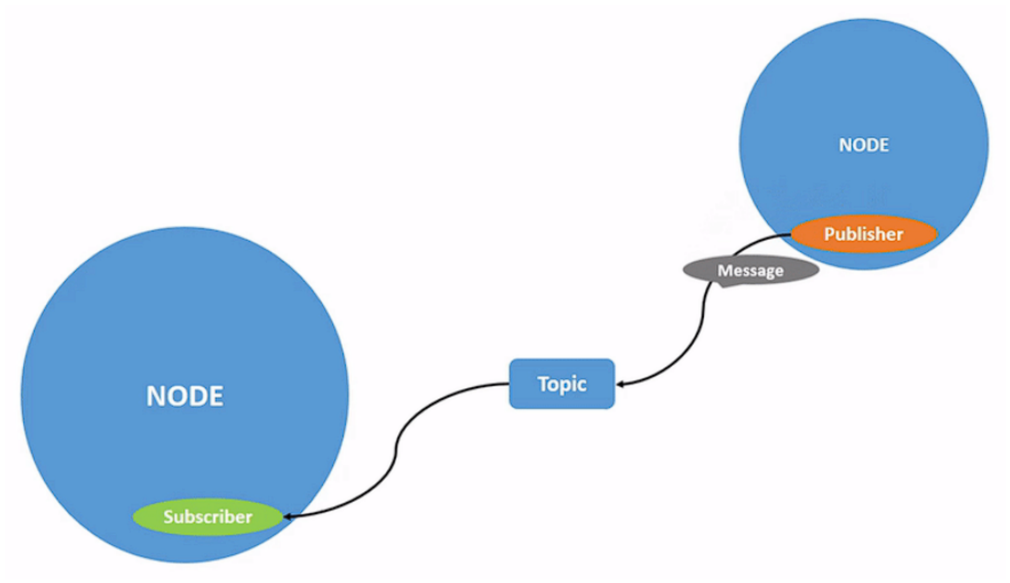


Figure 4.2: Representation of a publisher/subscriber communication itnerface through topic [3]

Services

Services are another type of communication system between nodes. They are different from topics and messages because they are bi-directional; while topics let a continuous flow of data from publisher to subscriber, services are based on a call-response mechanism. Therefore, a node, called *server*, provides data only when another node, called *client*, requests it. This type of communication interfaces are defined in a file with the extension .srv, where the first part is dedicated to the request and the second part to the response.

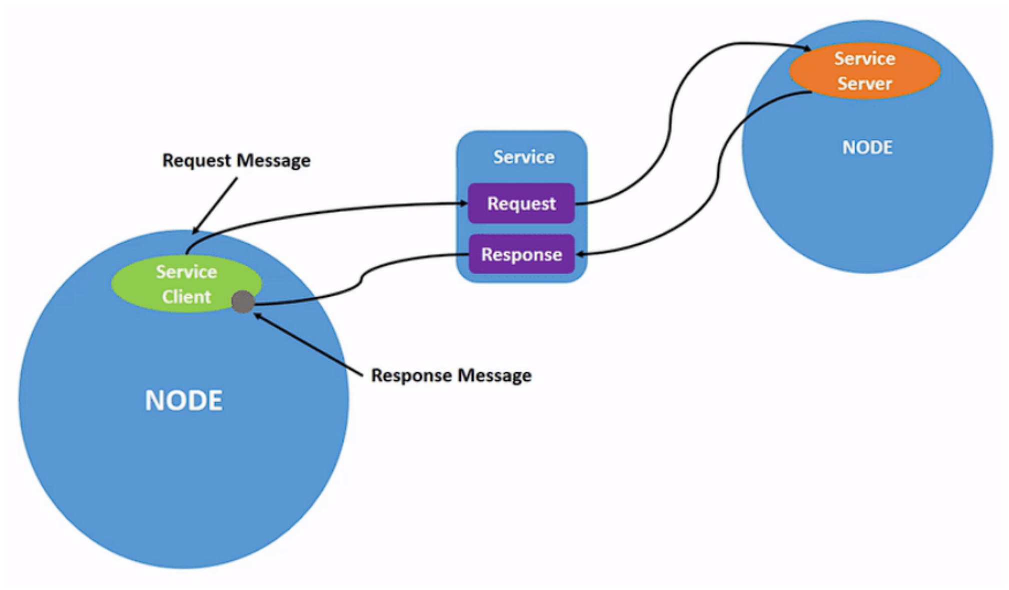


Figure 4.3: Representation of a client/server communication interface through service [2]

Actions

Actions are the third type of communication system implemented in ROS2, but they are intended to be used in long-running tasks. They are based on client-server mechanism, like services, but they include a goal service to which the client sends a request to the action server, which will execute the task, a topic in which the server publishes data stream to the client as feedback on the task, and a result service to which the server sends the result of the task to the client. Actions are defined in a file with the extension .action, where the first part is dedicated to the goal, the second one to the feedback, and the third one to the result.

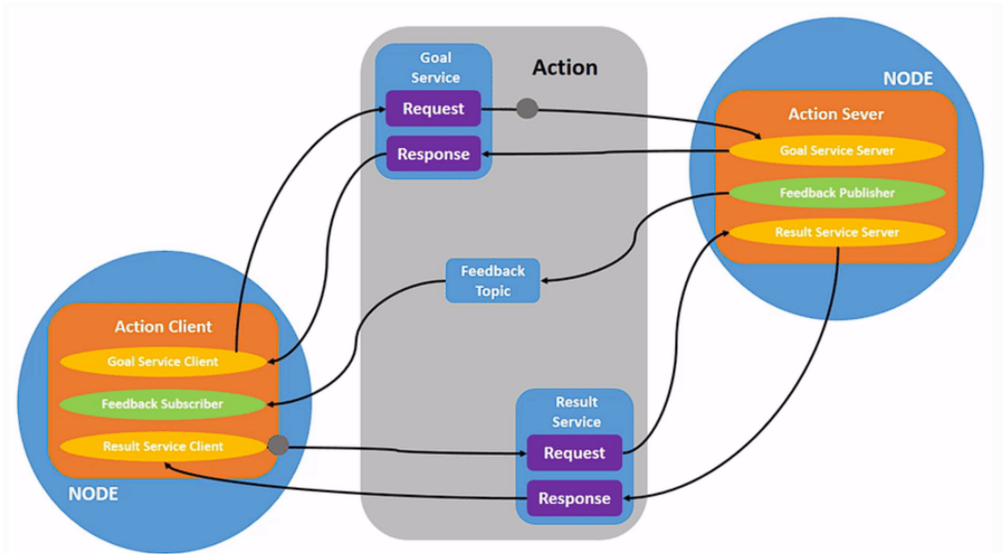


Figure 4.4: Representation of a client/server communication interface through action [1]

2 Gazebo

Since designing and testing a robot system directly on a physical platform can be complex and dangerous, it is extremely important to have simulation tools in order to perform tests in a virtual environment. Gazebo is a popular simulation tool open source that allows you to simulate a 3D environment with robots and sensors. It provides realistic simulations that support advanced physical models and provides powerful API that allows the programmer to extend and customize the simulator by implementing new plug-ins to add custom sensors, robot models, and other features. It is also widely used in the robotics community because it can be integrated with ROS/ROS2 applications. The SDF and URDF languages are used to create the simulation world model and the robot model, respectively. The URDF (Unified Robot Description Format) is an XML format used to describe the robot model properties, like links, joints and sensors, while the SDF

(Simulation Description Format) is an XML format used to describe the simulation world properties, like physical ones, lightings, etc...

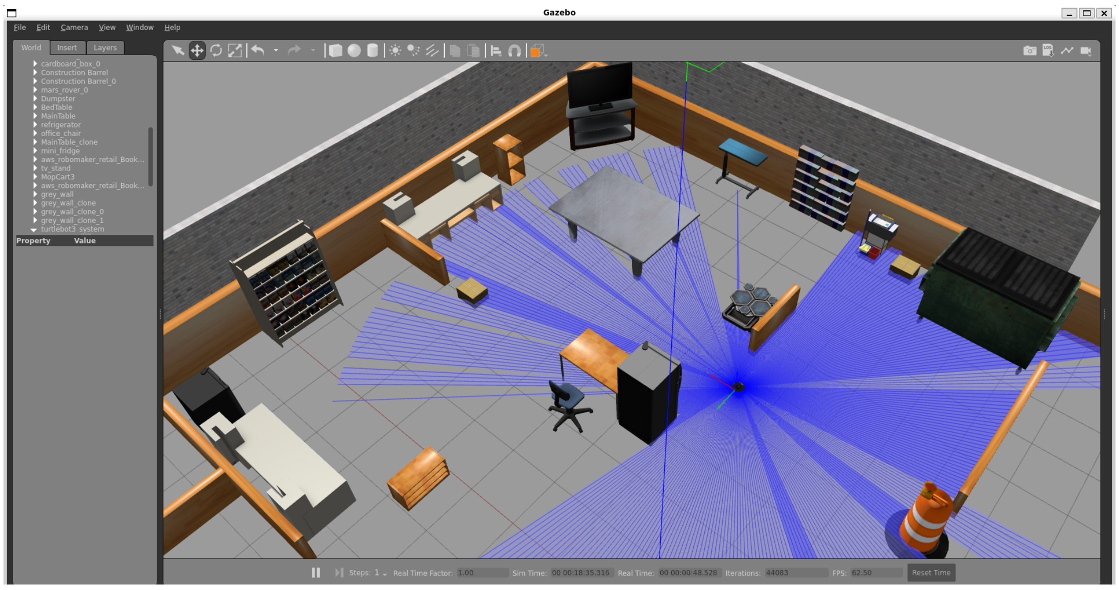


Figure 4.5: Representation of a Gazebo simulation environment

3 Turtlebot2 Kobuki

The Turtlebot2 Kobuki is a mobile robot platform designed by a korean company called Yujin Robotics. It has been developped for education and research, especially for indoor environments, built with two motor-driven wheels and a caster wheel to improve the stability. It has also a modular design that leads to a wide range of sensors and actuators that can be added to the robot, that let the user to customize the robot for various type of applications. Another advantage of the Turtlebot2 Kobuki is that is fully supported by ROS2, so it is possible to use it with the ROS2 ecosystem. For this project, the robot is equipped with a Rplidar A1 used for the mapping and the localization, a Realsense D435i camera used for the visual place recognition

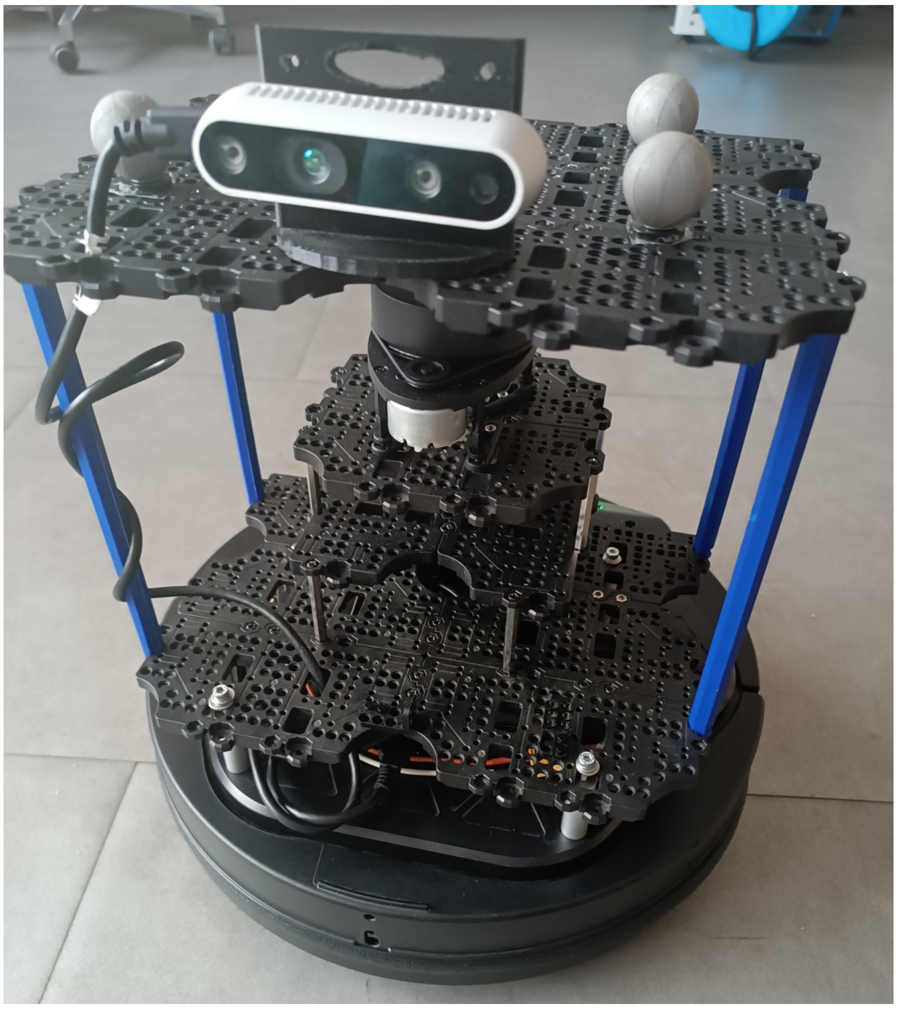


Figure 4.6: Turtlebot2 Kobuki mobile robot used for tests

3.1 Rplidar A1

In order to perform the localization on 2D map, the robot is equipped with a laser scanner device calles Lidar (Light Detection and Ranging), which is a sensing technology that uses laser beams, generated by a laser emitter that pulses at a high frequency, to measure the distance between the robot and the surrounding objects. The 2D Lidar is composes of a mobile part which rotates thanks to a motor, and a fixed part that contains the laser emitter and the receiver. On the turtlebot2 kobuki is mounted a Rplidar A1, a relatively low cost 2D lidar designed by Slamtec, that is also fully supported by ROS2 ecosystem.

Size	354 x 354 x 420 mm
Weight	6.3 kg
Max speed	0.65 m/s
Max rotational speed	180°/s
Max payload	5 kg
Battery	2200 mAh Li-Ion (4400 mAh Li-Ion extended)
IMU	gyroscope, accelerometer, magnetometer
Odometry	52 ticks/enc rev

Table 4.1: Technical specifications Turtlebot2 Kobuki

Figure 4.7: Rplidar A1 used for mapping and localization

3.2 Realsense D435i camera

Intel Realsense D435i is a popular depth camera that provides RGB images and depth data at the same time. The depth part consist of a Time-of-Flight technology that measures the distance between the camera and the objects in

Resolution distance output	< 1%
Range	12 m
Scan rate	5.5 Hz
Angular resolution	1°

Table 4.2: Technical specifications of Rplidar A1

the scene based on the time it takes for an emitted infrared light to return to the camera. The camera is also equipped with an IMU (Inertial Measurement Unit) that provides data about the robot's orientation and acceleration. Like the Rplidar, the Realsense D435i is upported by ROS2 ecosystem. For the purpose of this thesis, only the RGB module has been used to perform the visual place recognition.

Max Frame Resolution	1920x1080
Frame Rate	30 fps
Field of View	64°x 41° x 77°
Resolution	2 MP

Table 4.3: Technical specifications of the RGB module of Intel Realsense D435i

Figure 4.8: Realsense D435i camera used for visual place recognition

Chapter 5

Experimental Work

In this chapter, the experimental work will be discussed in detail from a software and a hardware point of view. The Turtlebot2 Kobuki platform has been used as mobile platform. The overall system runs on a NUC computer with 8GB RAM and an Intel(R) Core(TM) i5-7260U CPU @2.20 GHz with architecture x86_64 64-bit. Two kind of test has been performed: one in the PIC4SeR laboratory, aimed to test relocalization phase using Vicon motion capture system as ground truth in static and dynamic scenario, while the second one has been performed on a larger environment in order to test the overall system even with the trigger mechanism. All tests were carried out at PIC4SeR in indoor environment.

1 Tests on Relocalization Phase

As already discussed, the kidnapped robot problem is composed by two phase: kidnap detection and the relocalization phase. The most important part of the kidnapped robot is the relocalization phase, because it allows the robot to recover its position on the map. The first type of tests were performed in order to evaluate the relocalization phase in a static and dynamic scenario. In static scenario, nothing has been changed between the mapping phase and the tests, while the dynamic scenario has been emulated with some furniture that has been moved in other positions and with people walking around the robot. These tests were performed to evaluate if the relocalization pipeline was able to recover the robot position in a real scenario. The tests were performed in the PIC4SeR laboratory, where the Vicon motion capture system has been used as ground truth. The Vicon system is composed by 9 cameras distributed around the laboratory and it is able to track the position of the robot with millimetric precision. The robot has been equipped with 4 markers, the small balls on the top that can be seen in figure 4.6, that are coated with an highly reflective material that allows the cameras to track them; the 4 markers has to be positioned on the robot asymmetrically in order to be able to track also the orientation. First of all, an occupancy

Figure 5.1: Vicon cameras used for ground truth

grid map of the laboratory has been built with SLAM Toolbox and then 10 different points in the environment has been chosen to test the relocalization pipeline. In parallel, the database for the VPR has been built. In these tests the kidnap detection has been triggered manually through a service called *trigger_relocalization* that can be called from the terminal with the command *ros2 service call /trigger_relocalization std_srvs/srv/SetBool "{data: true}"*. The tests start with the service call, after which the robot starts to rotate to acquire the query images, and finish when the Max Current Weight reaches the threshold, set to 0.85. The distance error has been calculated as the

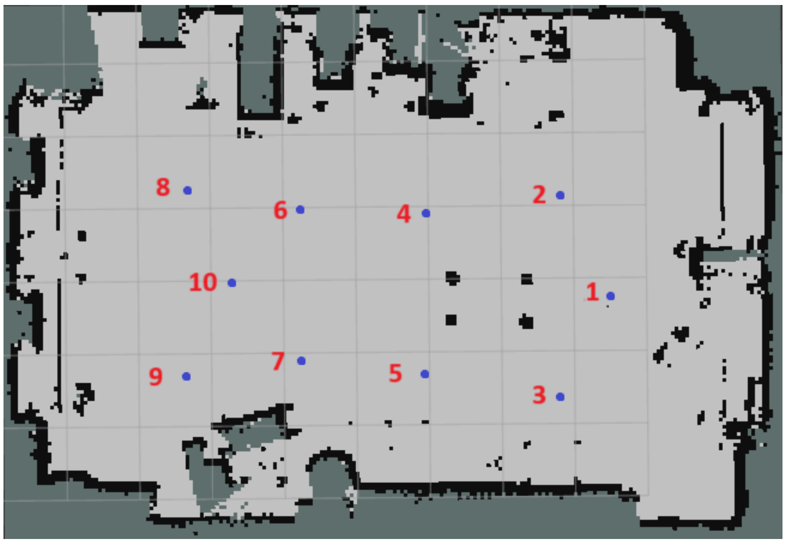


Figure 5.2: Occupancy grid map of the laboratory with the 10 positions where the relocalization tests have been performed

euclidean distance between the ground truth position of the Vicon and the estimated position of the robot. In the following table, data from the analysis of the tests is presented. As we can notice, the dynamic scenario has

	Error Initial Pose		Error Final Pose	
	Static Env.	Dynamic Env.	Static Env.	Dynamic Env.
Success (%)			85.0	82.0
Mean (m)	**0.666**	**0.734**	0.340	0.328
Min (m)	0.087	0.218	0.095	0.095
Max (m)	1.810	1.806	0.744	0.690

Table 5.1: Results of the tests on relocalization phase in laboratory

just slightly higher error, that demonstrate the robustness of relocalization pipeline to dynamic scenario and its ability to achieved good results in the initialization of the pose with the VPR. The success rate has been calculated as the percentage of the tests where the robot has reached good convergence.

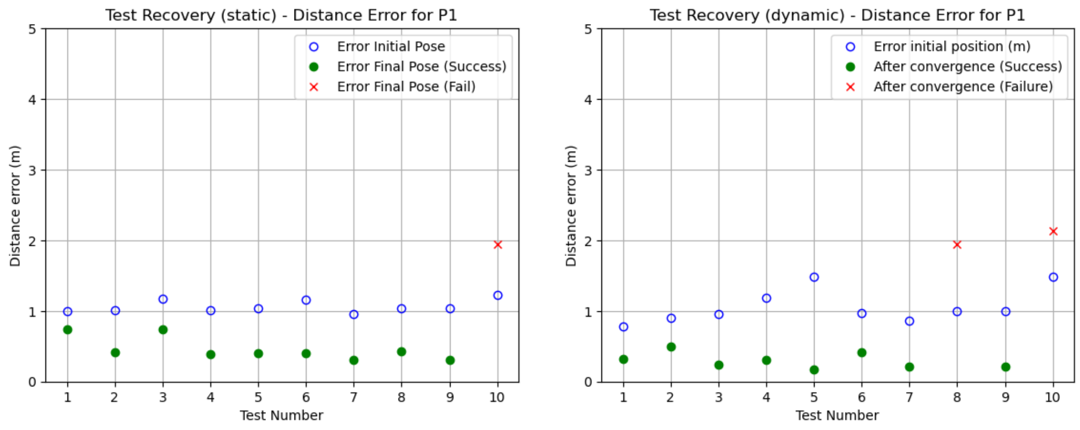


Figure 5.3: Distance error in position 1 in static and dynamic scenario

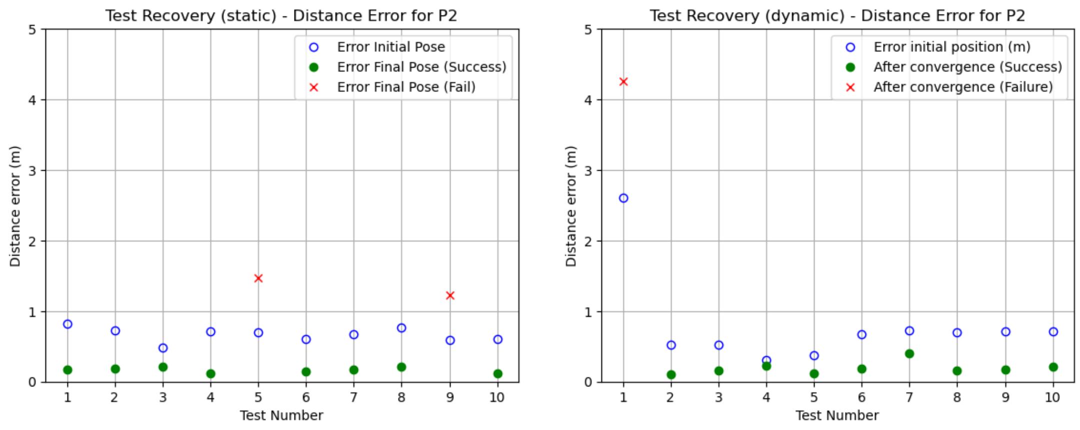


Figure 5.4: Distance error in position 2 in static and dynamic scenario

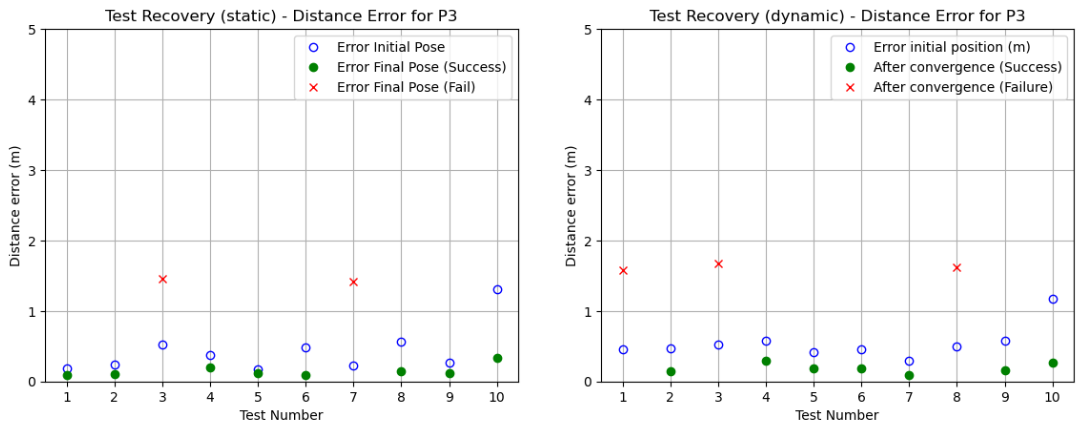


Figure 5.5: Distance error in position 3 in static and dynamic scenario

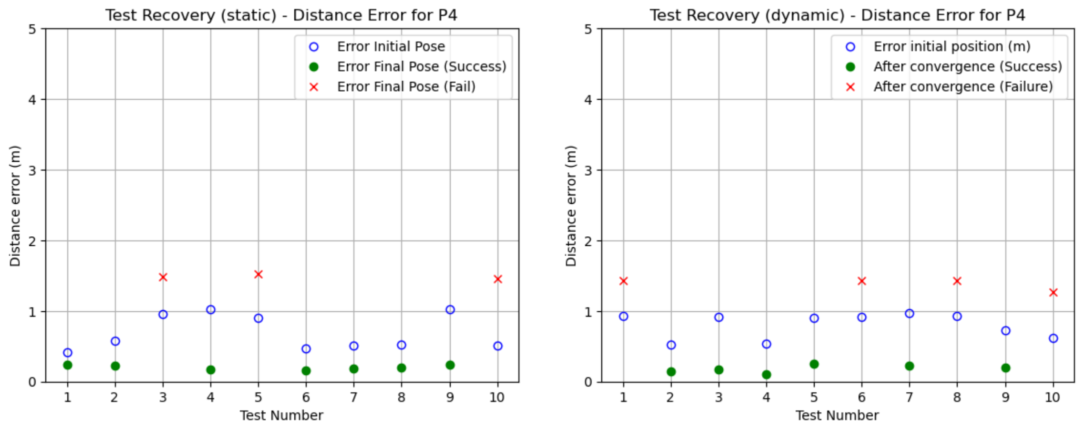


Figure 5.6: Distance error in position 4 in static and dynamic scenario

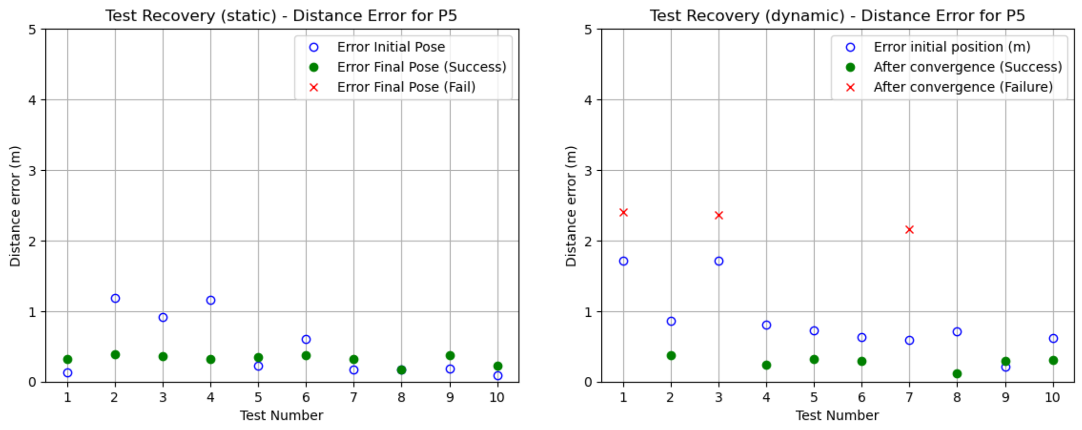


Figure 5.7: Distance error in position 5 in static and dynamic scenario

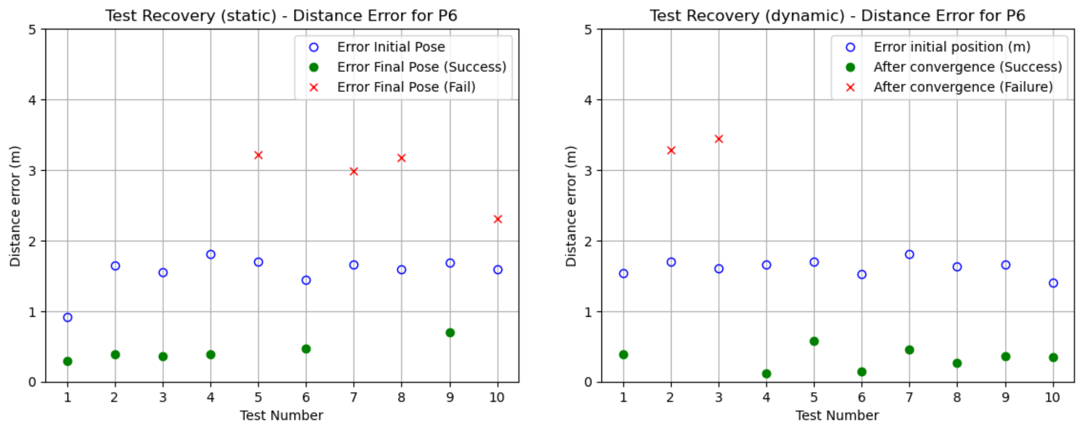


Figure 5.8: Distance error in position 6 in static and dynamic scenario

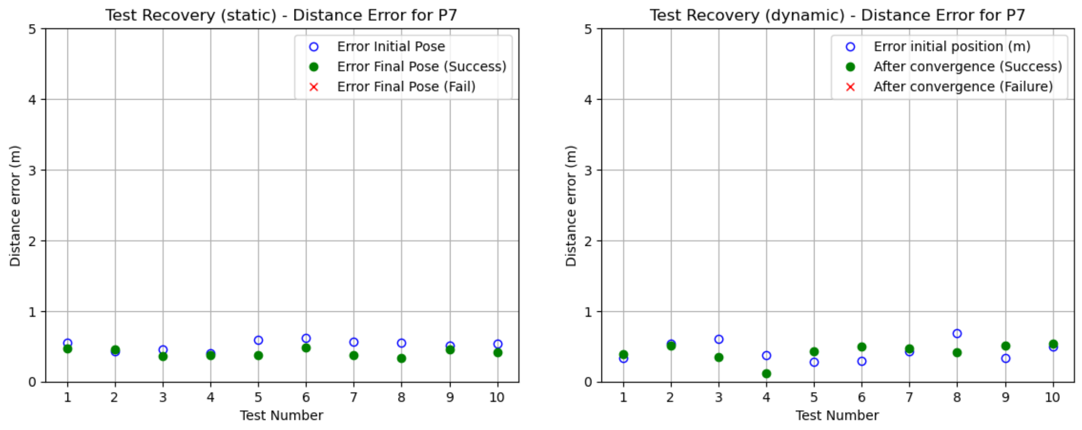


Figure 5.9: Distance error in position 7 in static and dynamic scenario

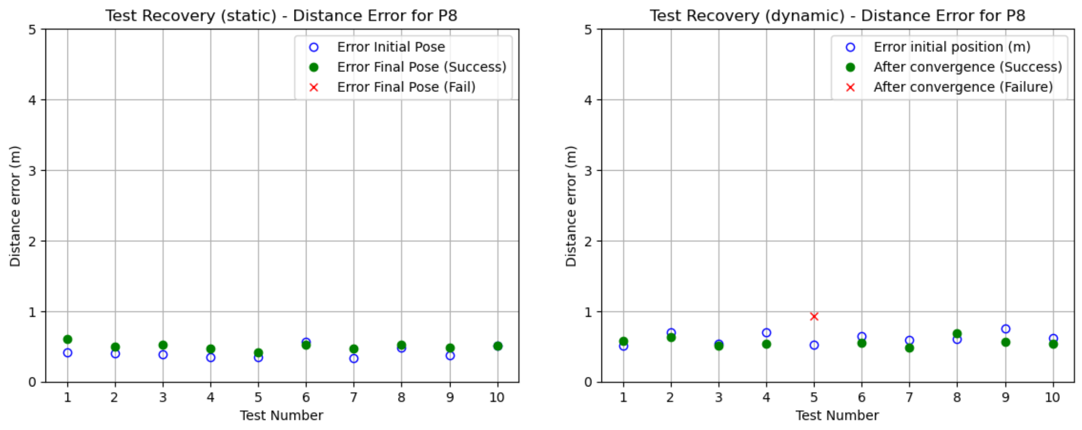


Figure 5.10: Distance error in position 8 in static and dynamic scenario

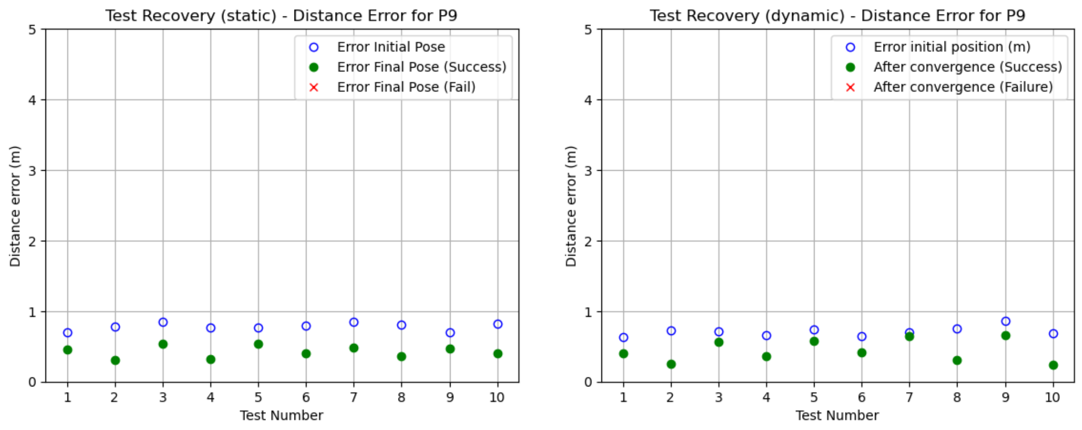
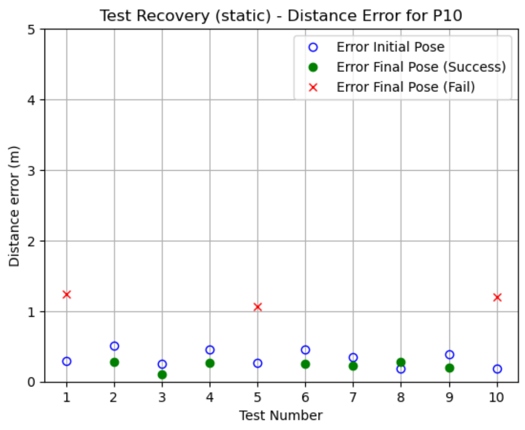


Figure 5.11: Distance error in position 9 in static and dynamic scenario



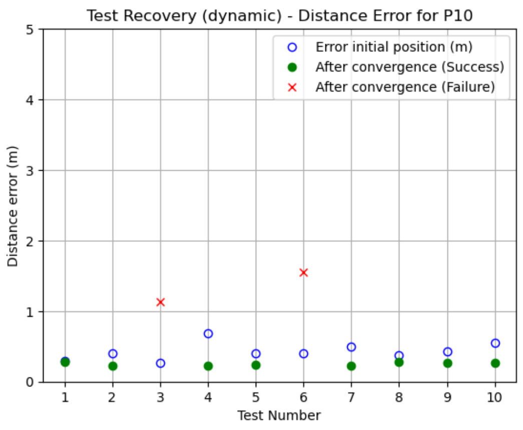


Figure 5.12: Distance error in position 10 in static and dynamic scenario

2 Tests on Overall System

Once the effectiveness of the relocalization phase has been tested in laboratory with Vicon, the overall system, that include even the trigger mechanism, has been tested in a larger environment. The tests has been carried out at PIC4SeR, comprising laboratory, corridor and thesist room. Tests has been performed simulating a navigation task, where the robot has to reach a goal position following a path calculated by the planner; the kidnapping has been triggered manually initializing a different position and orientation of the robot. After manual triggering the kidnap event, the system should have been able to detect it, recover the position and, after that, recalculate the path to the goal position. Through ROS2 launch files, three main subsystems are involved:

- **Navigation Stack**: the navigation stack has been launched thanks to the Nav2 package. The navigation stack introduced the behaviour-tree navigator, the planner and the controller. The planner is responsible for calculating the path to be followed by the robot to reach the goal position, based on the global and local costmaps. The controller is responsible for following the path calculated by the planner; actually it has been used the MPPI controller that is able to follow the path even in presence of obstacles, while the behaviour-tree navigator is responsible to manage specific complex robot behaviours.
- **Localization**: the localization subsystem launch the AMCL node that is responsible to estimate the position of the robot on the map. In the file of parameters, under the *amcl* namespace, the parameters for the AMCL node are set: *update_min_a* and *update_min_d*, that are the minimum angle and distance between two consecutive updates, are both set to 0.05 meters, in order have a faster resampling, hence a more accurate localization, while *recovery_alpha_fast* and *recovery_alpha_slow* are set to 0.0, leading to a number of particles generated equal to *min_particles*, set to 1000.
- **Kidnapped Robot System**: this is the system introduced in this project, which include the relocalization trigger mechanism and the recovery pipeline, already described in detail in chapter 3.

The system has been tested through 60 experiments, 10 for each of the 6 different kidnapping positions spread in the mapped environment. The scenario can be considered as dynamic since there were people walking around and some furniture has been moved. Every single experiments start with the robot well localized in the map; firstly, it receives a goal in the map and starts to follow the path calculated by the planner. After a while, between the starting point and the goal, in the selected position, the kidnap event has been manually triggered, forcing the robot to change its position and orientation. The kidnapped robot system should then be able to detect the kidnapping event, recover the pose and recalculate the path to the goal. The experiment is considered successful if the robot reaches the goal position within a maximum of two relocalization attempts. Since there is no ground truth outside the laboratory, the implemented system has been compared with a classical system based only on Monte carlo localization, where the same trigger mechanism has been used, but, instead of reinitialize the set of particles in a smaller part of the map, the particles are reinitialized on the whole map. The results of the tests are reported in the following table 5.13.

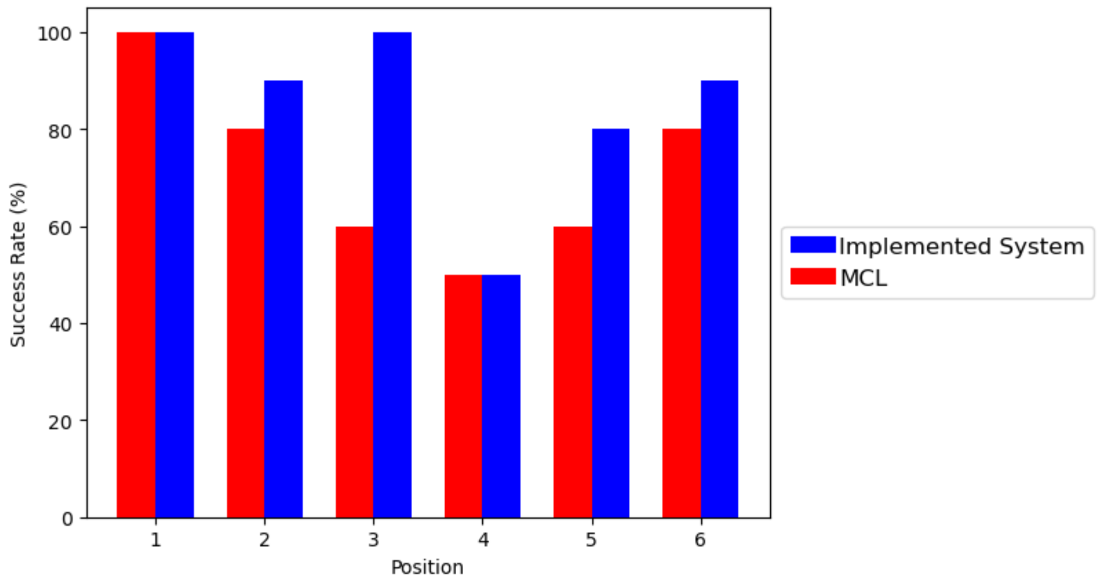


Figure 5.13: Comparison between the implemented system and the monte carlo ones for kidnapped robot

As we can notice, the implemented system has a higher success rate than the one that depends only on lidar with only Monte Carlo localization.

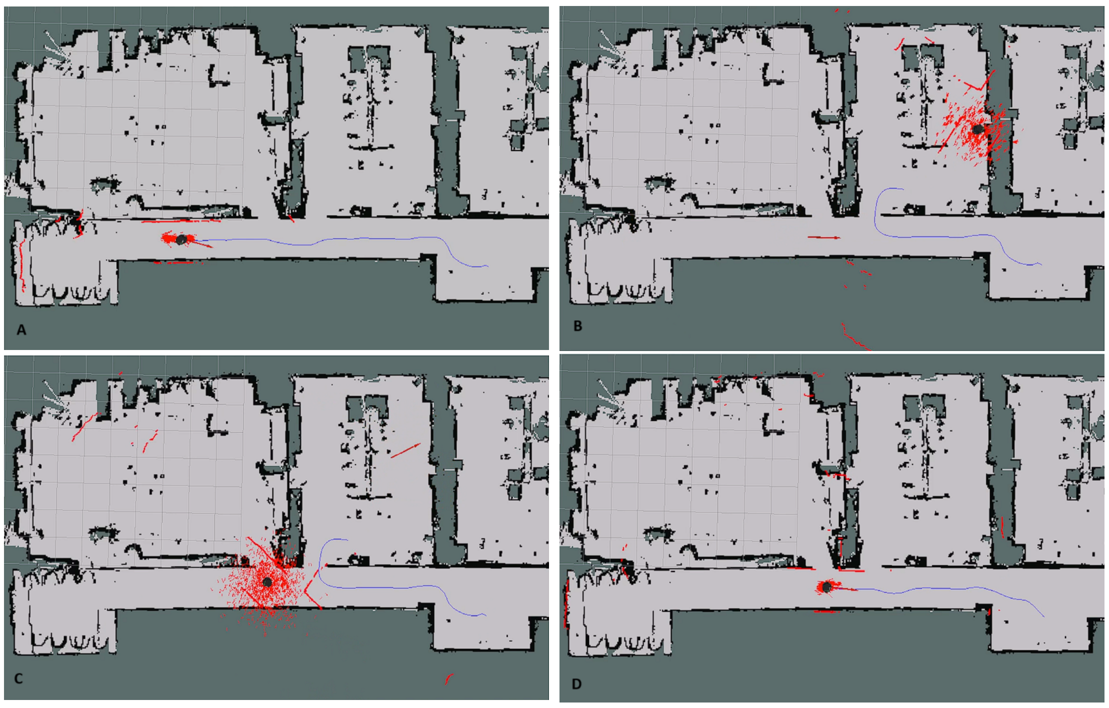


Figure 5.14: Example of an experiment where the robot has been kidnapped; in A the robot is navigating to the goal, in B it has been manually kidnapped, in C the position has been initialized and in D, after convergence, the robot resumed navigating towards the goal

Chapter 6

Conclusions

In this thesis, a system able to solve the kidnapped robot problem has been developped. The most used localization systems, especially in indoor environment, are based on 2D lidar sensor, that let to localize the robot in a occupancy grid map through Monte Carlo localization or lidar-based SLAM algorithms. Since for kidnapped robot problem, this systems still struggle a bit in dynamical and similar scenario, for this thesis a framework based on 2D lidar and RGB camera has been developped. The system is able to relocalize the robot in the map using only RGB images. It robustly restrict the area of the map where to find the correct pose of the robot with a fusion in the particle filter of Monte Carlo localization, correcting the pose with 2D lidar and odometry sensors. The tests have been performed in a real environment at PIC4SeR, at first evaluating the effectiveness of the recovery phase in laboratory using the Vicon system as ground truth, and then comparing results of the entire pipeline with a relocalization system based only on Monte Carlo with just the 2D lidar. Also in this second scenario, the implemented setup shows better performance. Obviously, the system can still be improved; first of all, next step should be test it in a large-scale environment; then it could be improved trying to find a solution to add a visual part in the trigger mechanism, that is just lidar-based, and also convert the 2D localization in a 3D localization to better adapt the system in an outdoor scenario.

Bibliography

[1] Ros2 service concept, https://docs.ros.org/en/humble/tutorials/beginner-cli-tools/understanding-ros2-actions/understanding-ros2-actions.html.

[2] Ros2 service concept, https://docs.ros.org/en/humble/tutorials/beginner-cli-tools/understanding-ros2-services/understanding-ros2-services.html.

[3] Ros2 topic concept, https://docs.ros.org/en/humble/tutorials/beginner-cli-tools/understanding-ros2-topics/understanding-ros2-topics.html.

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pages 404–417. Springer, 2006.

[5] Peter Biber and Wolfgang Straßer. The normal distributions transform: A new approach to laser scan matching. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 3, pages 2743–2748. IEEE, 2003.

[6] Eloi Bosse, Jean Roy, and Dominic Grenier. Data fusion concepts applied to a suite of dissimilar sensors. In *Proceedings of 1996 Canadian Conference on Electrical and Computer Engineering*, volume 2, pages 692–695. IEEE, 1996.

[7] Dylan Campbell and Mark Whitty. Metric-based detection of robot kidnapping. In *2013 European Conference on Mobile Robots*, pages 192–197. IEEE, 2013.

[8] Belur V Dasarathy. Sensor fusion potential exploitation-innovative architectures and illustrative applications. *Proceedings of the IEEE*, 85(1):24–38, 1997.

[9] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte carlo localization for mobile robots. 2:1322–1328, 1999.

[10] Wilfried Elmenreich. An introduction to sensor fusion. *Vienna University of Technology, Austria*, 502:1–28, 2002.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. 96:226–231, 1996.

[12] Dorian Gálvez-López and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[13] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[15] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 3304–3311. IEEE, 2010.

[16] S Karam, V Lehtola, and G Vosselman. Simple loop closing for continuous 6dof lidar&imu graph slam with planar features for indoor environments. *ISPRS journal of photogrammetry and remote sensing*, 181:413–426, 2021.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[18] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Ep n p: An accurate o (n) solution to the p n p problem. *International journal of computer vision*, 81:155–166, 2009.

[19] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[20] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[21] Steve Macenski and Ivona Jambrecic. Slam toolbox: Slam for the dynamic world. *Journal of Open Source Software*, 6(61):2783, 2021.

[22] Robert K McConnell. Method of and apparatus for pattern recognition,

January 28 1986. US Patent 4,567,610.

[23] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[24] Antonio Pegorelli Neto and Flavio Tonidandel. Analysis of wifi localization techniques for kidnapped robot problem. In *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 53–58. IEEE, 2022.

[25] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. Localization strategies for autonomous mobile robots: A review. *Journal of King Saud University - Computer and Information Sciences*, 34:6019–6039, 2022.

[26] Francisco Martín Rico. *A Concise Introduction to Robot Programming with ROS2*. Springer International Publishing, Cham, 2023.

[27] Open Robotics. Ros - robot operating system, 2024.

[28] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[29] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3), jul 2017.

[30] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*, chapter 8. The MIT Press, 2005.

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

[33] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*, chapter 1. The MIT Press, 2005.

[34] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*, chapter 2. The MIT Press, 2005.

[35] Sebastian Thrun, Dieter Fox, Wolfram Burgard, et al. *Monte carlo localization with mixture proposal distribution*. 2000.

[36] Wikipedia. Gaussian distribution wikimedia common.

[37] Song Xu and Wusheng Chou. An improved indoor localization method for mobile robot based on wifi fingerprint and amcl. In *2017 10th International Symposium on Computational Intelligence and Design (ISCID)*, volume 1, pages 324–329. IEEE, 2017.

[38] Chuho Yi and Byung-Uk Choi. Detection and recovery for kidnapped-robot problem using measurement entropy. In *International Conference on Grid and Distributed Computing*, pages 293–299. Springer, 2011.