FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

AI-Based Global Localization Based On Natural Features Detected On Point Clouds

João Miguel de Carvalho Oliveira

Master in Electrical and Computer Engineering

Supervisor: Armando Jorge Sousa (Ph.D.)

Second Supervisor: Héber Miguel Sobreira (Ph.D.)

November 4, 2025

Resumo

A robótica tem registado um crescimento significativo em diversas áreas, incluindo ambientes domésticos, industriais e de alto risco. Nestes contextos, os sistemas autónomos oferecem melhorias em segurança, eficiência e conveniência ao assumirem tarefas repetitivas, perigosas ou intensivas em trabalho humano. Um fator essencial para essa autonomia em robôs móveis é a capacidade de se localizarem com precisão dentro de um ambiente.

Enquanto a localização em ambientes exteriores tipicamente recorre a tecnologias como o Sistema Global de Navegação por Satélite (GNSS), os ambientes interiores apresentam desafios únicos. Os sinais GNSS são tipicamente indisponíveis ou pouco fiáveis em interiores devido à atenuação do sinal, e a presença de características ambíguas ou dinâmicas complica ainda mais a estimativa precisa da pose. Como resultado, a localização em espaços interiores deve basear-se em estratégias alternativas que consigam lidar com estas condições. Abordagens baseadas em marcos (landmarks), que utilizam pontos de referência com posições conhecidas, oferecem uma solução, mas apresentam limitações relevantes. Estes sistemas requerem uma colocação e manutenção cuidadosa dos marcadores, limitando a escalabilidade e flexibilidade, especialmente em ambientes grandes ou sujeitos a alterações frequentes.

Esta dissertação apresenta um sistema de localização global inovador e livre de marcos artificiais, desenvolvido para ultrapassar essas limitações. A abordagem proposta utiliza dados 3D de um sensor LiDAR em combinação com perceção baseada em inteligência artificial (IA) para identificar cantos formados pela interseção entre paredes e tetos. Estas características são estáveis e relevantes em termos de mapeamento, tornando-as especialmente adequadas para uma localização robusta em ambientes interiores complexos. A perceção também codifica a leitura do LiDAR num descritor, que é então comparado com um mapa de descritores para inferir diretamente uma estimativa grosseira da posição do robô ao longo da operação.

Em vez de alimentar o filtro de partículas com o scan completo do LiDAR, o sistema processa a nuvem de pontos para extrair apenas as características mais relevantes. O objetivo é reduzir a carga computacional do filtro mantendo a informação necessária para uma estimativa de pose precisa. O conjunto resultante de características é utilizado para atualizar um filtro de partículas, que mantém múltiplas hipóteses de pose e converge gradualmente para a localização verdadeira do robô.

Os testes demonstraram que o sistema proposto lida de forma fiável com cenários interiores desafiantes, incluindo o problema do robô raptado, em que o robô é subitamente movido para outra pose no mapa durante a operação. Ao contrário do AMCL, que não conseguiu recuperar nesses casos, a nova abordagem convergiu consistentemente para a pose correta utilizando um conjunto compacto de características. Estes resultados evidenciam a robustez do sistema e o seu potencial para uma aplicação fiável no mundo real em ambientes interiores complexos.

Palavras-Chave: Localização Interior, LiDAR, Filtro de partículas, Localizaçao Monte Carlo, Inteligência Artificial, Localização Global, Robô Raptado

Abstract

Robotics has experienced significant growth across a wide range of application domains, including domestic environments, industrial settings, and hazardous locations. In these varied contexts, autonomous systems offer improvements in safety, efficiency, and convenience by taking on tasks that are repetitive, dangerous, or labor-intensive. A key enabler of such autonomy in mobile robots is the ability to localize accurately within a given environment.

While outdoor localization can often rely on technologies such as the Global Navigation Satellite System (GNSS), indoor environments present unique challenges. GNSS signals are typically unavailable or unreliable indoors due to signal attenuation, and the presence of ambiguous or dynamic features further complicates accurate pose estimation. Landmark-based approaches, which use predefined markers with known positions, offer one solution but come with notable drawbacks. These systems require careful placement and maintenance of the landmarks, limiting scalability and flexibility, especially in large or frequently changing environments.

One of the most widely adopted methods for landmark-free indoor localization is the particle filter, exemplified by tools like the Adaptive Monte Carlo Localization (AMCL) algorithm. This system uses Light Detection and Ranging (LiDAR) sensors to scan the robot's surroundings and match the scan against a known map to estimate the robot's pose. While effective, such systems process the full scan for each particle, leading to high computational demands. Additionally, they are vulnerable to discrepancies between the map and the actual environment—such as unmapped or dynamic obstacles—which can cause incorrect pose estimates and reduced reliability.

This dissertation introduces a novel, landmark-free global localization system designed to address these limitations. The proposed approach leverages 3D LiDAR data in combination with artificial intelligence (AI)-based perception to identify corners formed by the intersection of walls and ceilings. These features are stable and map-relevant, making them particularly suitable for robust localization in complex indoor spaces. This perception also encodes the LiDAR reading into a descriptor, which is then compared to a descriptor map to directly infer a coarse estimation of the robot position throughout the operation.

Instead of feeding the entire LiDAR scan into the particle filter, the system processes the raw point cloud to extract only these meaningful features. This aims to reduce the computational load of the filter while preserving the necessary information for accurate pose estimation. The resulting feature set is used to update a particle filter, which maintains multiple pose hypotheses and gradually converges to the robot's true location.

Experiments demonstrated that the proposed system reliably handled challenging indoor scenarios, including robot kidnapping. Unlike AMCL, which failed to recover in such cases, the new approach consistently converged to the correct pose using a compact set of features. These results highlight the system's robustness and its potential for reliable real-world deployment in complex indoor environments.

Keywords: Indoor Localization, LiDAR, Particle Filter, Monte Carlo Localization, Artificial Intelligence, Global Localization, Kidnapped Robot

UN Sustainable Development Goals

The United Nations Sustainable Development Goals (SDGs) establish a global framework for addressing the most pressing challenges facing humanity, including climate change, sustainable cities, and technological innovation. This dissertation contributes to this mission by advancing localization systems for autonomous mobile robots (AMRs) through the development of an AI-driven, feature-based localization approach using 3D LiDAR.

By proposing a scalable, efficient, and landmark-free localization method that performs well in dynamic, cluttered indoor environments, this work supports cleaner, smarter, and more adaptable automation solutions. These contributions have the potential to reduce reliance on fossil fuel-based transport systems, improve productivity in smart infrastructure, and promote innovation in robotics and automation.

The specific Sustainable Development Goals addressed by this dissertation include:

SDG	**Target**	**Contribution**	**Performance Indicators and Metrics**
9	9.4	Enhancing robotics infrastructure through intelligent and adaptable localization systems, reducing the need for artificial markers.	Use of AI-based feature extraction and particle filtering in a 3D LiDAR framework.
	9.5	Supporting research and innovation in robotic perception and navigation.	Development of a novel feature-based global localization algorithm using high-level semantic cues.
11	11.6	Reducing environmental impact of urban logistics through automation.	Contribution to indoor mobile robotics systems capable of operating in shared spaces without external infrastructure.
	11.B	Promoting sustainable and efficient robotics solutions for smart infrastructure and mobility.	Scalable solution for warehouse, hospital, or office automation without dependence on GPS or markers.
13	13.2	Integrating climate-conscious technologies in automation and transport systems.	Lower energy footprint through improved localization accuracy and reduced infrastructure dependency.

Agradecimentos

Obrigado aos meus amigos, aos meus pais e irmão, e à Lola pela constante companhia e paciência para desabafos meus ao longo destes meses de dissertação. Foram sem dúvida o fator mais importante para a recarga de energia, motivação e vida nos momentos mais difíceis. Um obrigado e abraço adicionais à Margarida Biltes por toda a partilha de ideias, ajuda e otimismo.

Obrigado também aos meus orientadores, Armando Sousa e Héber Sobreira, por todo o apoio e pela oportunidade de trabalhar num tema tão interessante e abrangente quanto este.

João Miguel de Carvalho Oliveira

Hofstader's Law - "It always takes longer than you expect,
even when you take into account Hofstadter's Law"

Douglas R. Hofstader

Contents

List of Figures

List of Tables

Abbreviations and Symbols

1D	One dimensional
2D	Two Dimensional
3D	Three Dimensional
ADAM	Adaptive Moment Estimation
AI	Artificial Intelligence
AMCL	Adaptive Monte Carlo Localization
AMR	Autonomous Mobile Robot
CNN	Convolutional Neural Network
ESS	Effective Sample Size
FAISS	Facebook Artificial Intelligence Similarity Search
FPFH	Fast Point Feature Histograms
GNSS	Global Navigation Satellite System
ICP	Iterative Closest Point
iiLab	Industry and Innovation Laboratory
IMU	Inertial Measurement Unit
IoU	Intersection over Union
ISS	Intrinsic Shape Signature
KLD	Kullback-Leibler Distance
LiDAR	Light Detection and Ranging
LSTM	Long Short-Term Memory
MAP	Maximum A Posteriori
ML	Machine Learning
MLP	Multi-Layer Perceptron
NDT	Normal Distributions Transform
PCA	Principal Component Analysis
PFH	Point Feature Histograms
PGM	Portable Gray Map
PIMM	Pose Inference Miguel Margarida
PDF	Probability Density Function
RANSAC	Random Sample Consensus
ReLu	Rectification Unit
RGB	Red-Green-Blue
RGB-D	Red-Green-Blue + Depth (camera type)
RNN	Recurrent Neural Network
ROS / ROS2	Robot Operating System (Version 2)
RSSI	Received Signal Strength Indicator
SDG	Sustainable Development Goal
SHOT	Signature of Histograms of Orientations
SLAM	Simultaneous Localization and Mapping
SIFT	Scale-Invariant Feature Transform
VLAD	Vector of Locally Aggregated Descriptors

Chapter 1

Introduction

1.1 Context

Robotics has experienced rapid growth in recent decades, significantly impacting various domains of daily life and industry. In domestic settings, robots are increasingly used for tasks such as automated vacuuming, lawn mowing, and food preparation, offering enhanced convenience and time savings. In hazardous environments, robotic systems help minimize human exposure to danger by taking on high-risk operations. In industrial contexts, the adoption of robotics has contributed to a reduction in workplace accidents and injuries [1, 2], while also improving productivity and operational efficiency across sectors such as warehouse logistics, transportation, machining, and assembly lines.

Autonomous mobile robots (AMRs) are robots capable of navigating and operating within an environment without direct human control. These robots have seen notable evolution. Early industrial AMRs typically relied on physical guidance systems such as magnetic strips or visual markers. But, advancements have enabled modern robots to function without dedicated artificial infrastructure, slowly moving closer to the perceptual capabilities of human operators. This shift aligns with the broader goals of the Industry 4.0 movement, which emphasizes intelligent, interconnected, and adaptive automation. In this context, mobile robots must be capable of navigating complex, unstructured, and dynamic environments.

As shown in figure 1.1 AMR's operation can typically be divided into four main functional blocks[3]:

- **Perception:** The robot gathers data from its environment and interprets this information to build an understanding of the surrounding space.
- **Localization:** Using the perception input and prior knowledge, the robot estimates its position and orientation within the environment, answering the critical question: "Where am I?"
- **Path Planning:** Once localized, the robot determines a feasible and often optimal path to reach a given destination.

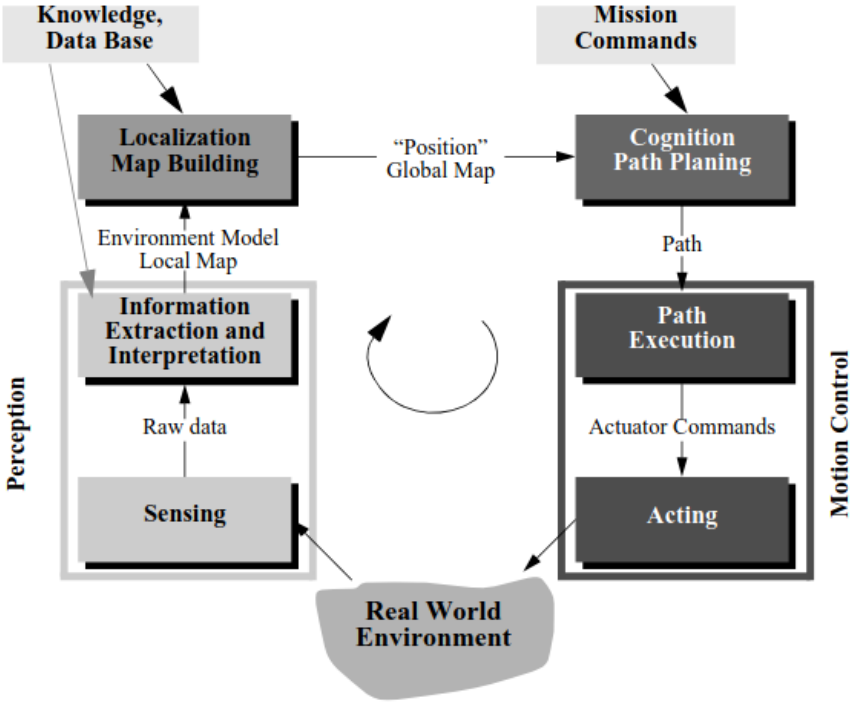


Figure 1.1: High-level scheme for mobile robotics, from [3].

- **Control:** High-level plans are translated into low-level motor commands, enabling the robot to execute motion.

So a fundamental requirement for autonomous mobile robots (AMRs) is the ability to perceive their surroundings and localize themselves accurately and robustly within them. This dissertation focuses on these two critical functions - perception and localization - which form the foundation for reliable AMR operation. Furthermore, the environment type -indoor or outdoor- presents different challenges when it comes to localization, so this work tackles only indoor environments.

1.2 Motivation

Localization typically involves solving two distinct problems: pose tracking and global localization [4]. In pose tracking, the robot's initial pose is known, and the system iteratively refines this estimate using sensor data. This data typically includes proprioceptive measurements from sensors like wheel encoders and Inertial Measurement Units (IMUs), which estimate only the change in the pose of the robot over time. On the other hand, global localization addresses the more challenging scenario where the robot's initial pose is unknown, so the system must be able to estimate it by associating the measurements from exteroceptive sensors to the known map of the environment. Additionally, the kidnapped robot problem refers to a variant of the global localization problem, which involves suddenly changing the robot's pose mid operation

A key challenge in global localization, particularly in indoor environments, is ambiguity. This occurs when distinct locations in the environment generate similar sensor readings, making it difficult to determine the robot's pose. Additionally, dynamic and cluttered surroundings are common for an AMR operating in an indoor scenario, where the widespread GNSS based systems are also often unreliable.

A solution for the these challenges of indoor global localization involves deploying artificial landmarks - distinctive and recognizable features in the environment - that the robot can detect and use as reference points to locate itself. Additionally, to deal with the environmental ambiguity, Monte Carlo based techniques are often employed - the so called called particle filters. These filters enable the system to handle multiple hypotheses for the robot's pose and solve the ambiguities over time, with the matching of more landmarks. Despite simple, landmarks require a direct line of sight to the robot, and their placement is time consuming and not scalable.

These challenges create the need for localization systems that do not require artificial landmarks, and this is often seen as the natural next step in the evolution of this field. These types of systems have already been deployed, with modern sensors like the Light Detection and Ranging Device (LiDAR) being heavily used as they combine a detailed description of the surroundings of the robot with a very high precision, both very important factors for perception. A widespread example is the Advanced Monte Carlo Localization (AMCL) package, from the Robot Operating System 2 (ROS 2) framework. It uses a planar LiDAR device to scan the surroundings of the robot, feeding the entire scan into a particle filter, where each particle is assigned a score based on how well the scan fits given its pose in the map, which is often updated via proprioceptive sensors. The particles with the highest score over time are presumably the most likely to be the real pose of the robot.

A clear disadvantage of this system arises when looking at the problem of dynamic or unmapped objects in the environment, which will be scanned and wrongfully matched by some particles as being part of the map. Furthermore, by feeding the entire scan to each one of the particles, this also creates a significant computational overhead.

To solve these issues, one could develop a system capable of detecting and ignoring unmapped objects in the scan, while condensing the information of the entire scan into high-level features. Each particle would then just have to match these features with the map, instead of matching the entire scan, and so a 3D LiDAR could be used without a significant impact on the computational overhead. That is exactly what this dissertation aims to develop: a localization system deploying a 3D LiDAR to ignore unmapped objects and detect valuable natural features in the environment, more specifically corners between walls and ceilings, with the use of AI. These corners are then fed into a feature based particle filter, capable of maintaining multiple hypotheses and able to deal with ambiguities. This system offers the possibility to converge to the correct pose faster than the AMCL approach, due to the use of AI-powered point cloud processing, while presenting a lower computational overhead.

1.3 Objectives

The objectives for the localization system are as follows:

- **Landmark-free localization**, only using odometry and 3D LiDAR readings to solve global and local localization problems.
- **Fast convergence time**, leveraging AI powered 3D LiDAR processing to help in collapsing ambiguities faster and more reliably than the AMCL package.
- **Low filter computational overhead**, by processing the point cloud first and feeding only the extracted features to the particle filter for matching, instead of the entire point cloud.
- **Robust** to dynamic and unmapped objects in the environment.

1.4 Document structure

The remainder of this document is divided into 4 distinct chapters:

- Chapter 2 presents typical approaches to localization, various techniques that are used in point cloud processing and probabilistic sensor fusion, and an discussion of a few AI-driven localization works that already exist in the literature.
- Chapter 3 introduces the architecture of the proposed system and the context of its development, followed by a detailed explanation of each one of its modules.
- Chapter 4 provides a direct comparison between AMCL and the developed system, describing the experimental methodology and discussing the outcomes of each experiment.
- Chapter 5 concludes the document by revisiting each of the proposed objectives, discussing the main contributions and presenting suggestions for future work in this theme.

Chapter 2

Background and State of the Art

This section presents the background knowledge and literature review regarding mobile robot localization, point cloud processing, deep learning and probabilistic sensor fusion.

2.1 Introduction

As mentioned in 1.2, localization problems in robotics are typically divided into three categories [4], based on the knowledge of the robot's initial pose (position and orientation):

- **Pose Tracking**: The initial pose of the robot is known. The goal is to continuously track how this pose changes over time.
- **Global Localization**: The initial pose of the robot is unknown. The system must determine the robot's pose within the environment from scratch.
- **Kidnapped Robot Problem**: The robot is unexpectedly moved to a different location during operation. The system must detect this and recover by re-estimating the correct pose.

A complete localization system should be capable of addressing all these challenges simultaneously. This involves continuously tracking the robot's pose, while analyzing its environment to estimate and correct it when necessary.

A common high-level structure for indoor localization systems emerges in the literature: motion estimation through odometry, environmental sensing via feature extraction or matching, and sensor fusion using probabilistic methods [5].

In line with this common structure, the present work addresses the problem of 2D global localization in indoor environments by adopting a modular architecture composed of motion estimation, geometric sensing, and probabilistic pose inference. The following sections examine each component in detail, outlining relevant methods from the literature.

2.2 Motion estimation

Motion estimation, also referred to as *pose tracking* or *relative localization*, refers to methods in which a robot estimates its current pose (i.e., position and orientation) based on incremental changes observed over time. These changes can be derived from internal measurements—such as inertial or wheel sensors—or from external perception sources, like visual or LiDAR-based odometry [6].

Regarding on methods based on internal measurements, these are typically implemented using two types of sensors: IMU's and wheel encoders:

Inertial Measurement Units (IMUs) measure linear acceleration, angular velocity, and, in some cases, orientation. They typically consist of accelerometers, gyroscopes, and sometimes magnetometers. By integrating acceleration and angular velocity over time, it is possible to estimate changes in pose.

Wheel Encoders measure the rotational displacement or speed of a wheel. In wheeled robots, this information enables the estimation of traveled distance and turning angle, which can be used to compute the robot's change in pose.

LiDAR and **visual** odometry rely on estimating motion between consecutive point clouds or images by tracking salient points or pixels across frames.

The principal limitation of motion estimation is its open-loop nature: it relies exclusively on internal measurements without reference to external observations. As a result, small sensor errors accumulate over time, leading to drift in the estimated pose. This makes it unsuitable for long-term localization unless it is combined with external corrections.

Despite its limitations, motion estimation methods remains useful in scenarios where global localization is unavailable. It provides fast and continuous pose estimates, which can serve as a short-term solution or as a prediction model for fusion with more accurate global methods.

2.3 Environmental representation

In the context of indoor global localization, there are many possible ways to represent the environment. In this work, these representations are grouped into three categories: landmark-based, dense geometric, and fingerprint-based.

2.3.1 Landmark-based

Landmark-based representations involve mapping features in the environment, known as landmarks. These landmarks — either artificial or natural — must be uniquely identifiable by the robot's sensors and have known positions within the map [7, 8]. By applying techniques such as trilateration (using distance measurements) or triangulation (using angle measurements) the robot can infer its pose relative to these landmarks, and therefore, to the global map.

2.3.1.1 Artifical Landmarks

Among the various landmark types, artificial landmarks are particularly favored in indoor settings for their robustness and ease of detection. The most widely used categories of artificial landmarks include RFID tags and fiducial markers[8]:

RFID tags use passive or active radio signals to provide localization information. Passive tags, which do not require an onboard power source, are especially attractive due to their low cost and small size. However, RFID-based systems can be sensitive to environmental interference (e.g., from metal surfaces) and depend heavily on the density and placement of both tags and readers to ensure adequate coverage.

Wireless antennas, using Bluetooth or Wi-Fi, are also often used as active landmarks, broadcasting unique signals that can be captured by the robot, that can be able to extract the distance and arriving angle of these signals.

Fiducial markers, are visual patterns placed in known positions in the environment. When detected by a camera-equipped robot, these markers allow for precise pose estimation through computer vision algorithms [9]. Examples of these markers are shown in figure 2.1. While they offer high accuracy, they typically require good lighting and an unobstructed line of sight. To address these limitations, recent works such as ArTuga [10] propose multimodal markers detectable by visual, thermal, and LiDAR sensors, improving robustness in low-light or occluded environments.

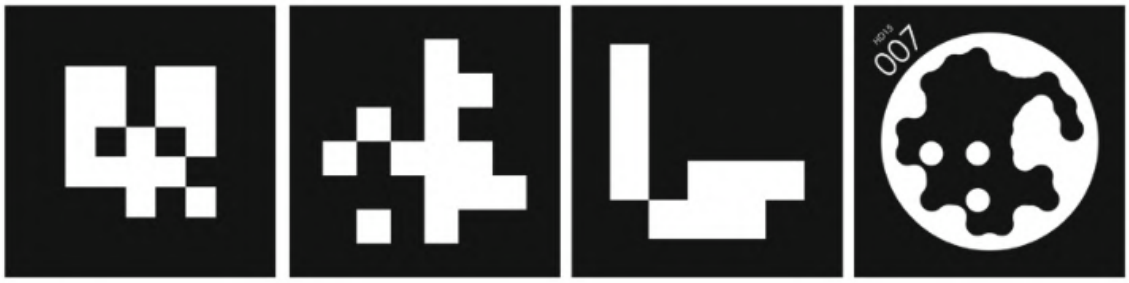


Figure 2.1: Fiducial markers examples, from [9]. From left to right, ARTAg, AprilTag, ArUco and STag

2.3.1.2 Natural Landmarks

Natural landmarks are structural or semantic elements of the environment—such as corners, walls, columns, windows—that are not artificially added for localization purposes but are instead inherent to the environment. Increasingly, research in robotics and computer vision has focused on leveraging such features for global localization, particularly in indoor scenarios where installing artificial markers may be impractical or undesirable.

Similar to artificial landmarks, natural features can serve as reference points if their positions in the environment are known. However, they introduce additional challenges: (i) their detection is often more complex, requiring robust feature extraction; and (ii) they are often not uniquely identifiable, which creates ambiguity during the data association phase.

To address this, probabilistic localization frameworks such as particle filters are commonly employed. These methods allow for maintaining and updating multiple pose hypotheses, which is essential when matching ambiguous observations to known map features [11].

2.3.2 Metric maps

Rather than relying on sparse or discrete features, metric maps continuous spatial representations of the environment, such as occupancy grids, 3D meshes, or point clouds. These representations encode detailed information about the environment's shape and structure, enabling the robot to compare real-time sensor data—typically from LiDAR or depth cameras—with the stored map. This approach allows for flexible and general localization even in unstructured or feature-poor environments, at the cost of increased memory and computational demands.

The AMCL ROS2 package is an example of a method which uses a dense geometric representation, in this case a 2D occupancy grid, for its environment representation. It presents a particle filter approach, where each particle's score is given by comparing the received and the expected LiDAR scans. This is done by transforming the points in the received scan to global coordinates and then computing a per point score based on the proximity to the occupied cells in the map.

2.3.3 Fingerprints

Fingerprinting-based localization relies on matching current sensory observations to a pre-recorded database of environmental measurements, referred to as fingerprints, each associated with a known position. These fingerprints may consist of various signal modalities, such as Wi-Fi signal strength (RSSI), magnetic field variations, or even global feature descriptors extracted from LiDAR or vision data. Unlike geometry-based methods, fingerprinting does not explicitly require modeling spatial relationships. Localization is achieved by simply identifying the closest match between the current observation and the database. While this enables fast localization in perceptually distinct environments, performance heavily depends on the quality, density, and temporal stability of the fingerprint map.

2.4 Point Cloud processing

A point cloud is a discrete set of points in 2D or 3D space. Besides the coordinates, each point may also have RGB information, normals and timestamps. These are typically generated by 3D scanners like LiDAR's or RGB-D cameras, and are broadly used in tasks like autonomous driving to provide an accurate representation of vehicle's surrounding environment.

This section will explain some existing methods used in point cloud processing, covering handcrafted descriptors, registration pipelines, outlier robust methods and deep learning approaches.

2.4.1 Keypoint detection and description

The process typically begins by detecting important points within each point cloud, the so called keypoints. Descriptors are then computed for each keypoint, with the goal of encoding each keypoint's unique geometric properties [12]. These keypoint descriptors can then be used as distinctive signatures that facilitate the comparison, alignment, or differentiation of point clouds based on their local and global structures.

When it comes to keypoint detection, there are 3 common methods in the literature:

- **3D Scale Invariant Feature Transformm (SIFT)** [13] is an extension of the original 2D SIFT algorithm [14] to 3D point clouds. It utilizes multi-scale representations and principal curvature analysis to detect scale-invariant keypoints.

- **Intrinsic Shape Signature (ISS)** [15] detects keypoints based on local geometric saliency. It computes the eigenvalues of the covariance matrix from a point's local neighborhood, which capture the spread of neighboring points along principal directions. A saliency score is derived by comparing these eigenvalues. A point is considered a keypoint if it has the highest saliency score in its neighborhood.

- **Harris 3D** [16] extends the 2D Harris corner detector to 3D by also using the covariance matrix of local neighborhoods. Instead of eigenvalue ratios, it calculates the *Harris response*, defined as the difference between the product of the eigenvalues and the squared sum of the eigenvalues scaled by a tunable factor k. Points with high Harris responses are selected as keypoints.

As for the descriptors, there have been a few methods developed over the years, with the most prominent being the following:

- **FPFH:** The PFH [17] or Point Feature Histograms method shows a computational complexity of $O(nk^2)$, for a pointcloud with n keypoints, each with k neighbors. To reduce the computational complexity the Fast PFH [18] was developed, while still retaining most of the discriminative power of the PFH.

  This method differs to PFH by first calculating the so called Simplified PFH (SPFH), which is the PFH applied only to the pairs containing the keypoint itself, to all the keypoints. In the second step, the SPFH values each keypoint are weighted using the weighted average SPFH values of other nearby keypoints and a weight that is inversely proportional to the distance to each of those points.

  Another improvement on complexity is the fact that the histograms created by SPFH are not b^f in size like for PFH, but $b \times f$, with b being the number of subdivisions, and f being the number of features considered, usually 3. This is due to the treatment of each feature as independent. These simplifications made it possible to achieve a complexity of $O(nk)$ for FPFH, making it better suited for time sensitive applications.

- **SHOT:** The Sigature of Histograms of Orientations, or SHOT for short [19], starts by defining a local reference frame, LRF, using the eigenvalue decomposition of the covariance matrix of the neighboring points. This frame needs to be robust to noise and invariant to translation and rotation. After obtaining the LRF, a spherical grid is formed with 2 subdivisions in the elevation, 2 in the radial and 8 in the azimuth, creating a grid with 32 volumes. For each volume's points, a binning process is applied, assigning a bin to each point based on the cosine of the angle between the normal of that point and the z axis of the LRF. This process creates a histogram of 11 bins per volume. For each keypoint, the histograms from each volume are combined, creating a signature of histograms with 33×11 positions, hence the name of the method.

2.4.2 Registration

Point cloud registration is the process of aligning two or more point clouds, typically representing the same object or scene from different viewpoints, into a shared coordinate frame.

This process can be achieved by first estimating a coarse alignment through an initial transformation, typically computed using feature correspondences such as the descriptors discussed in the previous subsection. To refine this alignment, fine registration methods are then applied, which directly minimize the spatial discrepancy between the point sets. Two widely used techniques for this refinement are the Iterative Closest Point (ICP) algorithm and the Normal Distributions Transform (NDT):

- **ICP:** The Iterative Closest Point algorithm is a method used to align two Point Clouds by iteratively minimizing the distance between them [20].

  The algorithm works by first identifying the closest corresponding points between the two Point Clouds. Then, it computes, typically through Single Value Decomposition, a translation and a rotation, that minimizes the sum of squared errors between the two. This is repeated, and the process finishes when the error is below a given value, the matrix transformation between iterations has a translation/rotation below the specified thresholds or when the maximum allowed number of iterations is reached.

  Due to limitations of the original algorithm, which is computationally inefficient and shows poor performance under noisy data, or large initial misalignment, several variants aiming to mitigate these issues have appeared [21] that aim to mitigate these issues.

- **NDT:** The Normal Distributions Transform (NDT)[22] algorithm is a method used to align two Point Clouds by representing each Point Cloud as a probability density function and then maximizing the likelihood of the transformed points.

  The algorithm works by dividing the target Point Cloud into a voxel grid. Within each voxel, a 3D Gaussian distribution is fitted to the point cloud data. This creates a probabilistic representation of the target Point Cloud. Then, the source Point Cloud is transformed and projected onto the PDF of the target Point Cloud. The likelihood of the transformed source

points given the target PDF is calculated. The transformation that maximizes this likelihood is determined, typically through optimization techniques such as gradient descent.

NDT is generally more robust to noise and outliers compared to ICP because it considers the probabilistic distribution of points within each cell instead of relying on direct point-to-point correspondences. This makes it less susceptible to errors caused by noisy data. However, NDT can be computationally more expensive than ICP due to the need to estimate and evaluate all the different probability density functions

2.4.3 RANSAC

In point cloud processing, data often contain noise, outliers, and partial overlaps, which can significantly hinder accurate alignment and geometric inference. To address these challenges, Random Sample Consensus (RANSAC) is widely employed as a robust model estimation technique, particularly in the context of point cloud registration and geometric primitive detection.

RANSAC operates by iteratively selecting a minimal subset of correspondences or points to hypothesize a model, typically a transformation matrix in the case of registration. The model is then evaluated against the full dataset to determine how many inliers (i.e., points consistent with the model within a given tolerance) it supports. The process is repeated for a fixed number of iterations or until a satisfactory model is found. The final model with the highest inlier count is selected as the best estimate.

In 3D registration tasks, RANSAC is often used in conjunction with keypoint descriptors to estimate an initial transformation between two point clouds. By randomly sampling a small number of descriptor correspondences, RANSAC can robustly estimate the transformation matrix even in the presence of false matches and noisy data. Once a reliable initial alignment is found, it can be refined using local methods such as ICP or NDT.

Beyond its use in robust registration, RANSAC is also widely applied in the detection of geometric primitives, such as planes, spheres, cylinders, and lines, within unstructured 3D point clouds [23, 24, 25].

2.4.4 PointNet architectures

PointNet [26] is a neural network architecture specifically designed to process and analyze unstructured point cloud data directly. Unlike the widely used Convolutional Neural Networks (CNNs), which rely on structured grid-like inputs such as images [27], PointNet can handle the unordered and irregular structure of point clouds directly, making it a pioneering approach for tasks like classification, segmentation, and object detection.

At the core of PointNet is a shared Multi-Layer Perceptron (MLP) applied to each point individually. This MLP learns to extract features from individual points, capturing their geometric properties. Since point clouds are unordered, PointNet uses a symmetric operation like max-pooling to aggregate these per-point features into a global feature vector, ensuring the network's

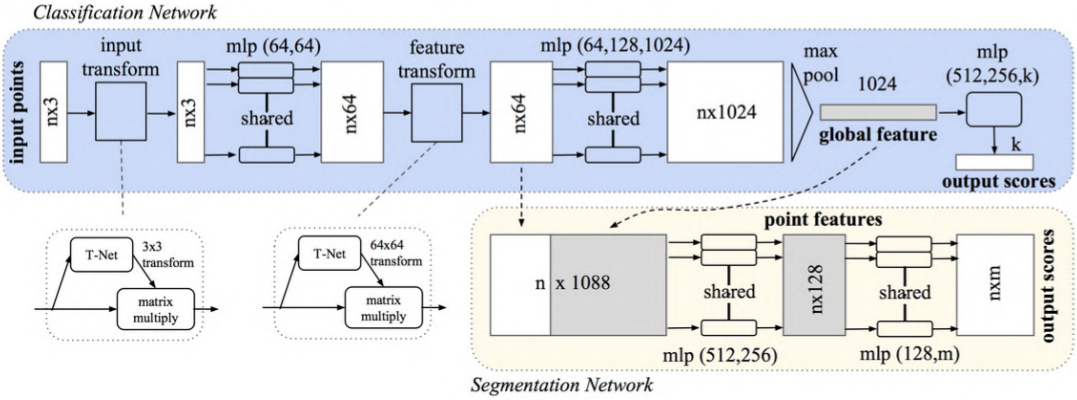


Figure 2.2: PointNet architecture. Taken from [26]

output is invariant to the order of the points. This global feature vector represents the overall structure of the point cloud and can be used for tasks such as classification or regression. Additionally, PointNet includes a transformation network to align the input points into a canonical space, further improving robustness.

In Figure 2.2, the original PointNet segmentation and classification architectures are shown.

PointNet++ [28], an extension of PointNet, enhances the original architecture by incorporating local geometric context. While PointNet treats each point independently before pooling, PointNet++ groups points into local neighborhoods based on their spatial proximity. These neighborhoods are hierarchically processed, similar to how CNN's extract local features at different scales. By applying PointNet to these smaller groups and progressively aggregating features across different scales, PointNet++ captures both local and global structures in the point cloud, improving its performance on more precise tasks like segmentation or fine-grained classification.

Both PointNet and PointNet++ introduced groundbreaking methods for processing point clouds, enabling direct analysis without requiring voxelization or other preprocessing steps, and, at the date of writing, are the most cited works in the field of deep point cloud processing.

2.5 Probabilistic Localization

A comprehensive introduction to the principles of probabilistic localization in robotics can be found in [5]. The following section draws heavily from the methods and theoretical background presented in that work, with the aim of summarizing the key ideas relevant to this dissertation.

Bayesian Localization Framework

In mobile robot localization, the goal is to estimate the robot's current state at time step k, given knowledge of its initial state and all sensor measurements up to time k, denoted by $Z_k = \{z_i \,|\, i = 1 \ldots k\}$. The robot's state is typically defined as a three-dimensional vector $\mathbf{x}_k = [x, y, \theta]^T$, representing its position and orientation.

This estimation task is a specific instance of the general *Bayesian filtering* problem, where we aim to compute the posterior probability density function (PDF) $p(\mathbf{x}_k \mid Z_k)$—the distribution of the current state conditioned on all past measurements. This posterior represents all available information about the state, from which we can derive an estimate, such as the maximum a posteriori (MAP) estimate or the mean. However, during *global localization*, the posterior may be highly multi-modal, making point estimates less meaningful.

The Bayesian localization algorithm proceeds recursively in two main phases:

Prediction Phase

In this phase, the robot's predicted state at time k is computed using a motion model. Assuming a Markov process, the current state $\mathbf{x}_k$ depends only on the previous state $\mathbf{x}_{k-1}$ and the control input $\mathbf{u}_{k-1}$. The motion model is defined by the conditional distribution $p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_{k-1})$, and the prediction is obtained by marginalizing over the previous state:

$$p(\mathbf{x}_k \mid \mathscr{Z}_{k-1}) = \int p(\mathbf{x}_k \mid \mathbf{x}_{k-1}, \mathbf{u}_{k-1})\, p(\mathbf{x}_{k-1} \mid Z_{k-1})\, d\mathbf{x}_{k-1} \tag{2.1}$$

Update Phase

In the update step, new sensor information is used to correct the predicted belief. Assuming that the latest observation z_k is conditionally independent of previous measurements given the current state, and using a sensor model defined by the likelihood $p(z_k \mid \mathbf{x}_k)$, we apply Bayes' rule to update the posterior:

$$p(\mathbf{x}_k \mid \mathscr{Z}_k) = \frac{p(z_k \mid \mathbf{x}_k)\, p(\mathbf{x}_k \mid \mathscr{Z}_{k-1})}{p(z_k \mid \mathscr{Z}_{k-1})} \tag{2.2}$$

This recursive process continues over time. At the initial time step t_0, the state $\mathbf{x}_0$ is described by a prior distribution $p(\mathbf{x}_0)$. In global localization, this prior is typically uniform over the entire space, reflecting complete uncertainty. In contrast, in tracking scenarios, the prior is usually a Gaussian distribution centered on a known initial estimate. In this work, as in, the transition from global localization to tracking is handled naturally, with the posterior evolving from a broad, multi-modal distribution to a sharp, localized peak as more sensor data becomes available.

2.5.1 Particle Filter

An effective class of algorithms for probabilistic localization is based on sampling methods, commonly known as *particle filters*. These filters approximate the posterior distribution $p(x_k \mid Z_k)$ using a finite set of weighted samples, or particles, rather than representing the distribution analytically. This approach is particularly well-suited for handling non-linear motion and observation models, as well as non-Gaussian uncertainty, which are common in mobile robotics.

The core idea is to represent the robot's belief at time step k as a set of N particles:

$$S_k = \{s_k^i \mid i = 1, \ldots, N\},$$

where each particle s_k^i is a sampled hypothesis of the robot's state x_k. These particles are used to reconstruct an approximation of the belief distribution, typically using histogram estimation techniques.

Particle filters operate in two main phases, executed recursively at each time step:

Prediction Phase. In the prediction step, each particle $s_{k-1}^i \in S_{k-1}$ is propagated forward using the motion model:

$$s_k^{i\prime} \sim p(x_k \mid s_{k-1}^i, u_{k-1}),$$

where u_{k-1} is the control input applied since the previous step. This results in an intermediate set of predicted particles S_k', representing a sample from the predictive distribution $p(x_k \mid Z_{k-1})$.

Update Phase. In the update step, sensor measurements z_k are incorporated by computing an importance weight for each predicted particle:

$$w_k^i = p(z_k \mid s_k^{i\prime}),$$

which reflects how likely the observation z_k is given the particle's state hypothesis. A resampling step is then performed to draw a new set S_k of unweighted particles from the weighted set $\{s_k^{i\prime}, w_k^i\}$. Particles with higher likelihoods are more likely to be selected, concentrating the belief around probable states.

Resampling Step. To address the degeneracy problem—where after several iterations, most particles have negligible weight—a *resampling* procedure is applied. This step generates a new set of particles $S_k = \{s_k^i\}$ by sampling with replacement from the predicted set S_k' according to the normalized weights $\tilde{w}_k^i$. This effectively concentrates particles in regions of high posterior probability, while discarding unlikely hypotheses.

There are several resampling methods, such as multinomial, systematic, stratified, and residual resampling. Among these, *residual resampling* is an efficient method that reduces variance introduced by the stochastic sampling process.

Residual resampling combines deterministic and stochastic selection to minimize sampling variance while maintaining proper particle diversity. The procedure works as follows:

Step 1: Compute Expected Copies. For each particle i, calculate the expected number of copies it should receive based on its normalized weight:

$$N_i = \lfloor N\tilde{w}_k^i \rfloor, \tag{2.3}$$

where $\lfloor \cdot \rfloor$ denotes the floor function and N is the total number of particles. This represents the *guaranteed* number of copies that particle i will receive in the resampled set.

Step 2: Deterministic Replication. Create N_i exact copies of each particle i deterministically. This ensures that particles with large weights are preserved proportionally to their importance, without any randomness in this initial allocation.

Step 3: Calculate Remaining Particles. After deterministic replication, compute the number of particles still needed to reach the target population size N:

$$R = N - \sum_{i=1}^{N} N_i. \tag{2.4}$$

Since we used the floor function in Step 1, we typically have $R > 0$ particles remaining to be allocated.

Step 4: Compute Residual Weights. For the remaining R particles, calculate the residual (fractional) weights that were "left over" from the deterministic step:

$$r_i = \frac{N\tilde{w}_k^i - N_i}{R}. \tag{2.5}$$

These residual weights represent the probability that particle i should receive an additional copy beyond its guaranteed N_i copies. Note that $\sum_{i=1}^{N} r_i = 1$, forming a proper probability distribution.

Step 5: Stochastic Sampling. Use multinomial sampling with the residual weights r_i to randomly select which particles receive the remaining R copies. Each of the R remaining slots is filled by drawing a particle index according to the probability distribution defined by $\{r_i\}$.

The combination of deterministic and probabilistic selection makes residual resampling particularly effective for maintaining both accuracy and diversity in particle filter implementations.

Initialization. At time step $k = 0$, the filter is initialized with a set of samples drawn from the prior distribution $p(x_0)$. For global localization, this prior may be uniform over the entire allowable space, while in tracking scenarios it is often a Gaussian centered around a known initial pose.

In figure 2.3, an illustrative example of a 1D particle filter is shown.

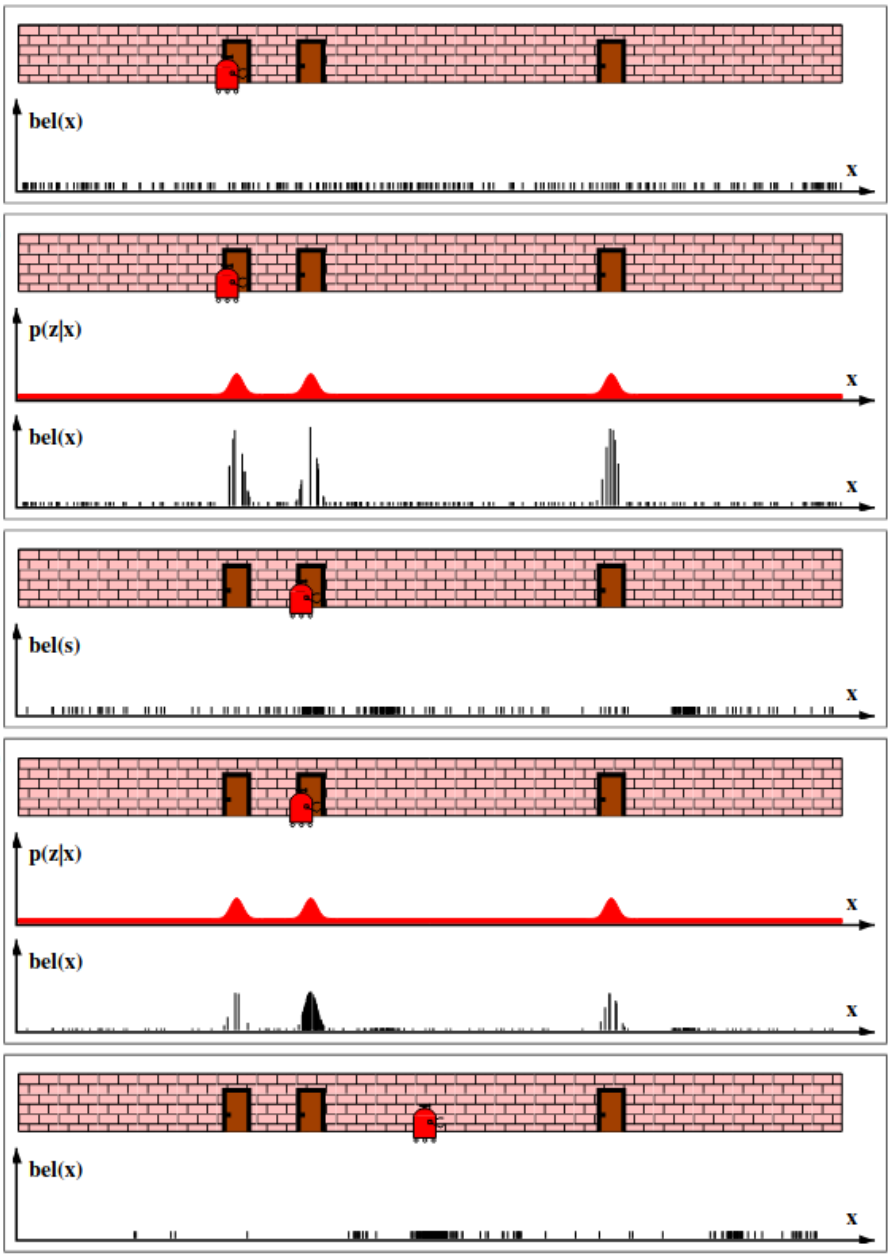


Figure 2.3: Illustrative example of the operation of a 1D particle filter regarding a door detecting robot. Taken from [5]

2.5.2 AMCL

Adaptive Monte Carlo Localization (AMCL) is a particle filter-based localization algorithm implemented in a ROS2 package with the same name [1] that addresses several key limitations of the standard particle filter approach described above. While the basic particle filter provides a robust framework for probabilistic localization, it suffers from computational inefficiency when using a fixed number of particles and lacks adaptability to varying levels of localization uncertainty.

[1] https://index.ros.org/p/nav2_amcl/

AMCL enhances the standard particle filter through two primary innovations: *adaptive sampling* and the *KLD-sampling* technique. These modifications allow the algorithm to dynamically adjust its computational resources based on the current state of localization uncertainty, leading to improved efficiency and robustness.

The Fixed Particle Problem. In conventional particle filters, the number of particles N remains constant throughout the localization process. This approach presents several inefficiencies:

- During periods of high certainty (e.g., when the robot is well-localized), maintaining a large number of particles wastes computational resources without significant improvement in accuracy.
- During periods of high uncertainty (e.g., during kidnapping scenarios or ambiguous sensor readings), a fixed number of particles may be insufficient to adequately represent the multimodal posterior distribution.
- The computational cost remains constant regardless of the localization task's difficulty, preventing real-time adaptation to changing conditions.

KLD-Sampling. AMCL addresses these limitations through Kullback-Leibler Distance (KLD) sampling, which dynamically determines the optimal number of particles based on the current distribution of the particle set. The key insight is that when particles are tightly clustered (indicating high certainty), fewer particles are needed to represent the belief accurately. Conversely, when particles are spread across multiple modes (indicating uncertainty), more particles are required.

The KLD-sampling algorithm works as follows:

Step 1: Spatial Discretization. The state space is discretized into a grid of bins, where each bin represents a small region of the (x, y, θ) space. The discretization resolution can be adjusted based on the desired precision.

Step 2: Bin Occupancy Tracking. During the resampling process, AMCL tracks which spatial bins contain at least one particle. Let k denote the number of occupied bins after drawing n particles.

Step 3: Statistical Test. The algorithm uses a statistical test based on the KLD between the true posterior and the particle approximation. Specifically, it employs the upper $(1-\delta)$ quantile of the chi-square distribution with $k-1$ degrees of freedom:

$$n \geq \frac{k-1}{2\varepsilon}\left(1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right)^3, \tag{2.6}$$

where ε is the maximum error tolerance, δ is the confidence level, and $z_{1-\delta}$ is the $(1-\delta)$ quantile of the standard normal distribution.

Step 4: Dynamic Particle Count. The resampling process continues until either:

- The statistical condition is satisfied (sufficient particles have been drawn), or

- A predetermined maximum number of particles $N_{\max}$ is reached to prevent excessive computation.

Adaptive Behavior. This adaptive mechanism enables AMCL to exhibit intelligent behavior across different localization scenarios:

Tracking Mode: When the robot is well-localized, particles cluster tightly around the true pose. Few bins are occupied, so the algorithm uses a minimal number of particles (often as few as dozens), significantly reducing computational overhead while maintaining accuracy.

Global Localization Mode: During initial localization or after kidnapping, particles spread across multiple hypotheses, occupying many bins. The algorithm automatically increases the particle count to adequately sample the multi-modal distribution, ensuring robust localization recovery.

Transition Scenarios: During ambiguous situations (e.g., passing through symmetric environments), the particle count adapts smoothly as uncertainty increases or decreases, providing efficient resource allocation without manual parameter tuning.

The result is a localization system that automatically adapts its computational complexity to match the difficulty of the localization problem, achieving both efficiency and robustness across diverse operating conditions. This adaptive behavior makes AMCL particularly well-suited for real-world robotic applications where computational resources are limited and environmental conditions vary significantly.

2.6 Datasets and data augmentation

The supervised learning dataset consists of a large set of examples of a given task that one wants the model to learn. This dataset is typically divided into 3 parts: the training set, the validation set and the testing set. The training set and validation set are used in the training of the model. The examples from the training set are used to update the model's parameters, and the examples from validation are used while training to evaluate the model's accuracy on new data, and often to stop the training if the model's convergence speed slows down to below a threshold, in a process called early stopping. The testing set is used after the training phase to evaluate the final model on data it wasn not trained on.

This dataset split can be done multiple times in multiple training cycles using different and unique divisions of the dataset. This technique is called k-fold cross validation, where the dataset is split into k different subsets, and each one is used as the test dataset exactly once. By taking into account results from all folds, this technique provides a more reliable and consistent measurement of the model's performance on a given dataset, minimizing the chance that a given test result is due to lucky or unlucky selections of the test subset.

The variety and size of the dataset have a very significant impact on the performance of every supervised learning algorithm, neural networks included. If a dataset is not varied or large enough, the model might learn the relationships within that training dataset, and have a good performance within it, but, when provided with a different dataset of the same relationship, the model will show

Table 2.1: LiDAR localization datasets.

Name	LiDAR model	Scenario
KITTI [30]	Velodyne HDL-64E	Road and Person (outdoor)
MulRan [31]	Ouster OS1-64	Road (outdoor)
Apollo-SouthBay [32]	Velodyne HDL-64E	Road (outdoor)
NCLT [33]	Velodyne HDL-32E	Road (outdoor)
Oxford RobotCar [34]	SICK LD-MRS	Road (outdoor)
vReLoc [35]	Velodyne HDL-32E	Office (indoor)

a poor performance. This phenomenon is called overfitting and is one of the biggest challenges in supervised learning.

To overcome overfitting, one needs to increase the size of the dataset, gathering more examples. Increasing the size of the dataset implies labeling a large amount of examples, which takes human effort and time.

The artificial expansion of the dataset by means of transforming the already existing examples is called data augmentation [29], and it is a technique commonly used to quickly increase the size of a given dataset, especially in computer vision applications. Common techniques include flipping the image, altering its colors, adding noise, cropping or rotating. This has been shown to prevent the learning of undesirable patterns by the model, thus mitigating the overfitting problem.

Although not as abundant as image datasets, there are some well-known LiDAR datasets with the purpose of training AI models for the task of localization. Most datasets for LiDAR based localization purposes are made with autonomous driving in mind, so they are taken in the context of outdoor road scenarios. The most notable examples of existing datasets are shown in table 2.1

2.7 AI-Driven localization examples

Some surveys have been done on the state of localization techniques, with both classical and machine learning approaches [36][37] [38].

LocNet [39] employs a CNN-based architecture that is initially trained within a Siamese network framework. Siamese networks are specialized neural architectures designed to assess the similarity between two inputs [40]. In this setup, LocNet learns to generate a feature vector from a handcrafted rotation invariant global descriptor. This feature vector is both rotation invariant and optimized for comparison with other vectors using the Euclidean distance. Additionally, the loss function used is designed in such a way that similar point clouds, or the so called positive pairs, originate feature vectors that are close in the feature space, and different point clouds, so called negative pairs, have feature vectors far away form each other.

During the map-building phase, LocNet processes the map data to create a global prior map. Each pose is associated with a generated feature vector and organized within a kd-tree structure, allowing for efficient retrieval during localization.

For online localization, new measurements are transformed into feature vectors using LocNet and compared to the prior map based on Euclidean distances. A particle filter then refines the pose estimation by identifying the most likely positions for the LiDAR in successive iterations. Once the filter converges, ICP is applied as a final step to achieve precise alignment within the global map.

PointNetVLAD [41] is an extension of the PointNet architecture, optimized for place recognition. Unlike LocNet, it is entirely trainable in an end-to-end manner, not needing handcrafted descriptors.

Features extracted using PointNet are processed using a NetVLAD layer. The NetVLAD layer is originally designed to aggregate local image features learned from CNN's into the VLAD[42] (Vector of Locally Aggregated Descriptors) bag-of-words global descriptor vector. This effectively creates a global feature extraction pipeline, taking as input any raw point cloud. This global descriptor is further processed via a neural network or PCA (principal component analysis) in order to perform dimensional reduction, thus creating a compact descriptor.

During training, the network is fed point cloud tuples consisting of an anchor point cloud (the reference), a positive point cloud (structurally similar to the anchor), and multiple negative point clouds (structurally different from the anchor). Using specialized loss functions, the training process minimizes the distance between the global descriptors of the anchor and positive examples while maximizing the distance between the anchor and negative examples, similarly to LocNet.

PCAN [43] works in a similar way to PointNetVLAD, deploying PointNet layer, a NetVLAD layer and a fully connected layer to be able to extract descriptors, in a trainable end-to-end manner. The core innovation is the introduction of a contextual attention mechanism, inspired in the self-attention mechanism [44], which assigns a score to each local feature, representing the relevance of that feature to the task, before aggregating the features.

L3-Net [32] presents a novel learning-based LiDAR localization system that achieves centimeter-level localization accuracy. It uses PointNet networks to extract descriptors from pre-selected keypoints and their neighborhood. This process is applied to the online point cloud and a 4D matching cost matrix is constructed, with each cell representing the likelihood of a given pose hypothesys in the discretized solution space, for the values of each keypoint. Afterwards, a 3D matrix is constructed, representing the consensus of all the previously calculated keypoints. This 3D matrix represents the discretized 2D solution space, where each cell is the likelihood of a given pose hypothesis. This hypotheseis is then fed into a RNN, the LSTM model, which also takes into account previous hypothesis to ensure temporal smoothness. The pose hypothesis pipeline and the temporal smoothness pipeline are trained separately.

PointLoc [35] presents an end-to-end localization framework, using PointNet to encode the point cloud, an attention model to remove outliers, like moving objects, from the point cloud encoding, an aggregation layer consisting of two Multi-Layer Perceptrons and a max pooling layer. The output of this sequence is a compact feature vector that describes the point cloud globally, giving more significance to features that appear more often. This feature vector is then

sent to a pose regressor, which consists of two branches of MLP's, each with 4 fully connected layers, one for the translation and one for the rotation.

This work also collected and released publicly an indoor localization dataset, vReLoc.

In table 2.2, the strengths and weaknesses of each model are discussed

2.8 Critical Review

This chapter presented the fundamental knowledge to understand the problem of localization and more specifically, global localization. Additionally, it also provided some insights on the inner working of current approaches for point cloud processing and Machine Learning.

In the end, an overview of existing approaches was made to show some common themes in the literature. One of these common themes was the use of PointNet in the first stage of feature extraction, usually with some processing of these features to deal with outliers and dynamic environments. Finally, it is clear that most approaches focus on the problem of outdoor localization, in road environments. This can be problematic, as this scenario is associated with longer ranges in the measurement and lower precision requirements, when compared to indoor. Additionally, the problem of ambiguity is also rarely dealt with. Finally, the lack of indoor localization datasets is very clear.

This dissertation aims to tackle the problem of indoor localization, having in mind the problem of ambiguity, as well as to create an indoor localization dataset from simulation and from the real world.

Name	Description	AI Role	Strengths	Weaknesses
LocNet	Siamese network-based LocNet (modified CNN) architecture to generate descriptors for efficient map matching	- Feature descriptor learning	-Robustness to ambiguity -Computational efficiency in matching phase	-PF can be computationally heavy -Not ready for dynamic environments -Handcrafted descriptors
PointNetVLAD	Extends PointNet with VLAD aggregation to encode point clouds into a single global descriptor	- Feature extraction and description	-End-to-end training	-Focus on place recognition and not precise localization -Ambiguity not approached
PCAN	Encoding local features into a discriminative global descriptor, introducing a self-attention module to take into account each point's context	-Feature extraction and description -Feature significance prediction	-End-to-end training -Robustness to dynamic environments through attention mechanism	-Focus on place recognition and not precise localization -Ambiguity not approached
L3-Net	End-to-end learning approach, with PointNet feature extraction, CNN cost volume regulatization and RNN to yield temporal smoothness across estimates	-Feature extraction and description -Volume regularization -Temporal smoothness	-End-to-end training -Probability assignment for each possible pose in cost volume	-Computationally heavy, with three ML models working concurrently
PointLoc	Deep network to predict 6-DoF poses directly from LiDAR data	-Feature extraction -Outlier removal -Feature aggregation -Pose regression	-End-to-end training -Robustness to outliers via attention mechanism -Indoor scenario	-Ambiguity problem not approached -Computationally demanding, implementing 4 ML models

Table 2.2: Existing AI-driven LiDAR localization approaches in the literature

Chapter 3

System Architecture and Implementation

3.1 Architecture and tools

This section outlines the overall parts of the system, giving the context of the environment where it was developed, the various tools used and choices made regarding the high-level architecture.

The proposed system, from now on referred to as PIMM, comprises three main modules: point cloud processing, corner detection and particle filter. Each one of these modules is explained in detail in the following chapters.

The point cloud processing model implements an AI model to classify each point as part of a wall, a ceiling, a floor or clutter. This module aims to give some robustness to the system, by filtering irrelevant points for the task of wall-ceiling corner detection. Additionally, this step also generates a global descriptor of each LiDAR reading, which was proven to be useful as a way to directly infer some possible positions for the robot, based on the proximity of these descriptors with previously stored ones, effectively working as an AI-powered fingerprint map.

The corner detection model leverages RANSAC to extract the main planes from just the wall points and the ceiling points, separately. The intersections between these planes are then calculated and validated as corners. Furthermore, the orientation of the corners is also calculated, to add more information regarding the placement of the robot relative to the corner.

Finally, a custom particle filter is used to receive the odometry, corner detections and the poses from the descriptor matching and fuse all this information into a position and orientation for the robot.

The parts of the pipeline and how they interact are shown in the diagram 3.1.

This pipeline was implemented in C++ and Python and is fully modular within a ROS2 framework. The ROS2 framework was chosen as it is a common tool for robotics applications, and so it provides some basic functionalities which helped in development.

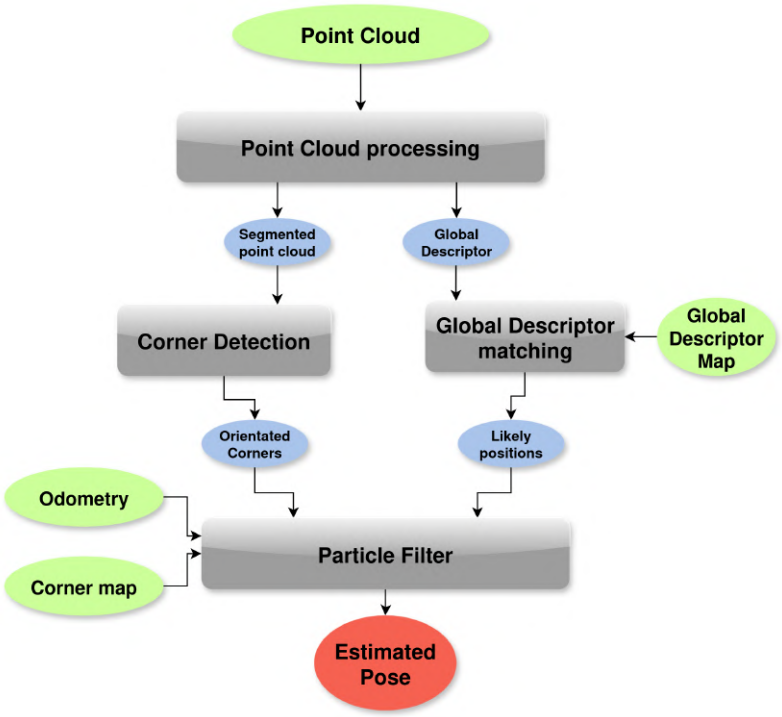


Figure 3.1: PIMM pipeline.

3.1.1 Webots

Webots [1] is an open-source robot simulation platform developed by Cyberbotics. It provides a comprehensive environment for modeling, programming, and simulating mobile robots in both indoor and outdoor settings. Webots supports a wide range of robot types, sensors, actuators, and communication protocols, making it suitable for research, education, and rapid prototyping. It was used in this work for simulating the 3D iiLab environment, the robot and all the required sensors.

3.1.2 ROS

The Robot Operating System (ROS) [2] is a widely adopted open-source middleware framework designed to simplify the development of robotic systems. While not a traditional operating system, ROS provides a structured communication layer on top of an existing OS (typically Linux) and offers a suite of tools, libraries, and conventions to facilitate modular, scalable, and reusable software design for robotics.

ROS abstracts much of the complexity involved in robot software integration by introducing a publish-subscribe communication model, where software components (called *nodes*) exchange information via *topics*. Nodes can also expose *services* for synchronous communication or *actions* for long-running, preemptible tasks.

ROS is used extensively in both academia and industry due to its:

[1] https://cyberbotics.com/ accessed on July 21 2025
[2] https://docs.ros.org/en/jazzy/index.html accesed on July 21 2025

- **Modularity**: Components can be developed, tested, and deployed independently.
- **Hardware abstraction**: Supports diverse sensor and actuator interfaces.
- **Community support**: Thousands of contributed packages and active community maintenance.
- **Tooling**: Includes visualization (e.g., RViz), simulation (e.g., Gazebo, Webots), and debugging tools.

In this work, **ROS 2** was used to coordinate the localization pipeline and simulation infrastructure. The robot and its sensors were integrated with ROS 2 using the `webots_ros2` bridge, enabling access to wheel encoder data, odometry, command velocity control, and ground truth pose. ROS 2 allowed for flexible experimentation by connecting the different system components via clearly defined interfaces.

Key ROS concepts utilized include:

- **Topics**: Asynchronous communication channels.
- **Nodes**: Independent processes such as the robot controller, feature extractor, or localization node.
- **Services**: Synchronous calls used for tasks like manually resetting the robot's position.

Additionally, custom ROS 2 packages were developed to implement various stages of PIMM, including feature extraction from point clouds and particle filter-based pose estimation. These packages adhere to ROS 2 conventions, promoting modularity and interoperability with future extensions or alternative robot platforms.

ROS Package Architecture and Organization

All the PIMM's components are organized into a modular ROS2 package structure, each with distinct functions. The modules were implemented as follows:

- **robot_localization_package:** Contains the particle_filter node and the launch files which enable the entire system's operation.
- **robot_worlds:** Contains the robot's controller and the path tracker, used for logging the ground truth and estimated trajectories into a csv file. It also contains all the world files used for simulation,
- **robot_msgs:** Custom message interface package defining Feature.msg (individual landmark with 2D pose, semantic class, confidence, and covariance) and FeatureArray.msg (Array of features with a single timestamp and coordinate frame) that compacts every Feature seen and processed in a single scan.

- **robot_pointnet:** Defines and implements the AI segmentation model, using PyTorch, the data structure logic for the global descriptor map and a ground-truth segmentator pipeline based on Kd_Tree nearest neighbor search of the LiDAR points, to act as the AI model.
- **robot_feature_detector:** Implements the the corner detecting process.
- **robot_waypoint_follower:** Implements a controller in order to make to robot follow a predefined trajectory through a list of waypoints.

The transform tree

The ROS framework defined a transform tree, made up of frames and spatial transformations between frames. Each frame is connected to others through time-stamped transforms, forming a directed graph of relative poses In this work, the transform tree is composed of the following frames:

- `map`: The global reference frame. All localization estimates are expressed relative to this frame.
- `base_footprint`: The frame fixed to the base of the robot, located at ground level.
- `lidar3D`: The frame fixed to the LiDAR device
- `estimated_pose`: The frame fixed at the position and orientation that PIMM estimates
- `base_footprint_real`: A special frame published by the simulator, representing the ground-truth pose of the robot within the environment.

The transforms between these frames are listed below:

1. `map` → `base_footprint`: This transform is computed from wheel encoder data and updated at every control step. It provides the robot's estimated motion based on odometry alone.
2. `base_footprint` → `lidar3D`: This transform is made using a static transform broadcaster, as it is contant
3. `map` → `estimated_pose`: This frame is broadcasted by the particle filter node.
4. `map` → `base_footprint_real`: This transform is published using the Webots supervisor API to reflect the robot's actual position and orientation in the simulated world. It is used for ground-truth comparison during evaluation.

The Maps

Three types of maps are used in this system, each serving a distinct role in the localization pipeline:

Occupancy Map (PGM Map). The first map used is a 2D occupancy grid stored in `.pgm` format, accompanied by a `.yaml` metadata file. This map is generated from the simulated environment and encodes free space and obstacles using grayscale pixel values. It is primarily used for visualization and for initializing particles in the global localization process, allowing them to be spread over valid, navigable areas. The occupancy map also helps in interpreting the robot's pose within the environment during debugging and evaluation.

Feature Map. The feature map is a custom YAML file containing a list of corners formed at the intersection of walls and ceilings, each with the position (x, y) and orientation (theta). These features are manually extracted offline and are represented in the global `map` frame. The feature map acts as the reference for data association during the observation update step of the particle filter.

AI-Powered Fingerprint Map. The fingerprint map is a dense, AI-generated representation of the environment based on point cloud features. Unlike the sparse feature map, this fingerprint map is used to encode learned descriptors at different spatial locations using an AI model. Each location in the map is associated with a compact vector that characterizes the local geometry, allowing fast similarity matching against incoming observations.

3.1.3 The Robot

The robot used in the simulation is a differential-drive mobile robot designed for indoor navigation tasks, shown in figure 3.2. It serves as the platform for evaluating PIMM within the controlled virtual environment. It is equipped with 2D and 3D LiDAR sensors and wheel encoders, with which the robot collects the sensory data required for point cloud processing and motion estimation. Although this work only requires the 3D LiDAR for its functionalities, the 2D LiDAR was added for the use with the AMCL ROS2 package, which works with 2D planar scans of the environment.

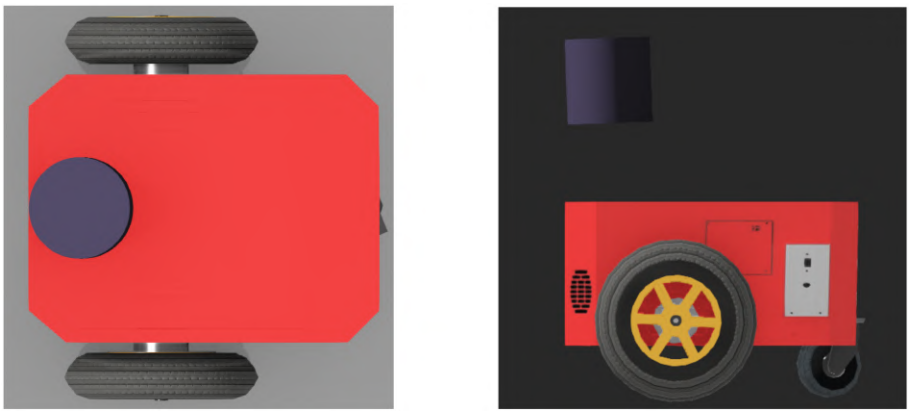


Figure 3.2: Robot and LiDAR from top and side views.

The robot is controlled via a custom Webots/ROS controller that interfaces Webots with ROS 2. This controller has the following responsibilities:

- Tele-operation of the robot via the cmd_vel topic, where a user can send linear and angular velocity commands using messages of type geometry_msgs/msg/Twist, adhering to ROS conventions.
- Spawning the robot and setting its position and orientation in the environment, via a ROS service and the supervisor API of Webots, which provides direct manipulation of the objects present in the simulation.
- Broadcasting the true pose of the robot in the environment via the base_footprint_real frame, relative to the map frame, also using the Webots supervisor API.
- Converting the ticks from the two wheel encoders into odometry messages, which are broadcasted via the `base_footprint` frame.
- Enabling the 2D and 3D LiDAR devices and publishing the point clouds to the designated topics.

The odometry of the robot was calculated using the following equations. The wheel encoders provide the angular displacement of the left and right wheels, from which the linear and angular displacements of the robot are estimated as:

$$\Delta d = \frac{r}{2}(\Delta\phi_R + \Delta\phi_L) \tag{3.1}$$

$$\Delta\theta = \frac{r}{2L}(\Delta\phi_R - \Delta\phi_L) \tag{3.2}$$

where r is the wheel radius = 9.4 cm, L is half the distance between the wheels = 16.5 cm, and $\Delta\phi_R$, $\Delta\phi_L$ are the changes in encoder readings for the right and left wheels, respectively, between each time step, which is 32 ms.

These values are then used to update the robot's pose (x, y, θ) incrementally:

$$x_{t+1} = x_t + \Delta d \cdot \cos\left(\theta_t + \frac{\Delta\theta}{2}\right) \tag{3.3}$$

$$y_{t+1} = y_t + \Delta d \cdot \sin\left(\theta_t + \frac{\Delta\theta}{2}\right) \tag{3.4}$$

$$\theta_{t+1} = \theta_t + \Delta\theta \tag{3.5}$$

As for the LiDAR devices, their parameters are listed below:

- **2D LiDAR**: 360º of FOV, 360 points of horizontal resolution and 10 meters of maximum range.
- **3D LiDAR**: 360º of horizontal FOV, 120º of vertical FOV, 40 layers of 360 points each, with 10 meters of maximum range.

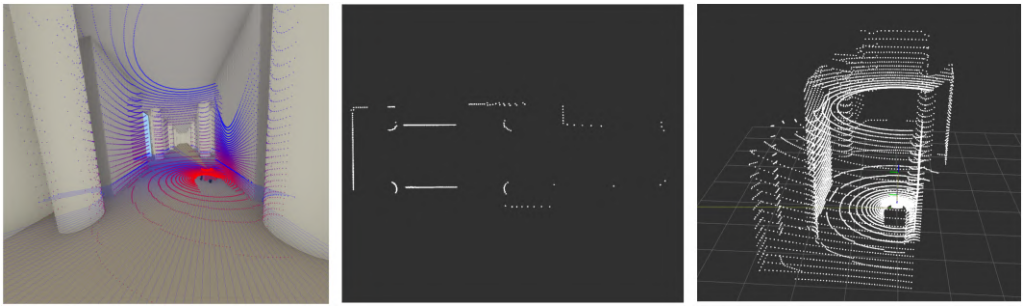


Figure 3.3: LiDAR scans (center - 2D, right - 3D) originated from the simulation (left).

3.1.4 The Environment

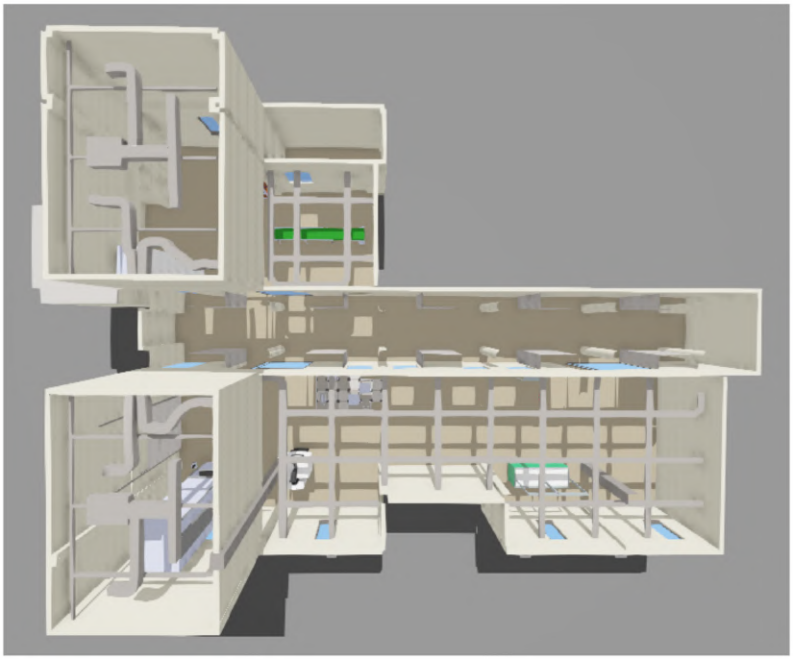


Figure 3.4: Top view of entire simulated iiLab environment in Webots.

The development and validation of PIMM were conducted in a high-fidelity simulated environment modeled after iiLab (figure 3.4, a large indoor area aiming to emulate an industrial setting for research purposes. It consists of multiple interconnected areas (in this work defined as A, B, and C, shown in figure 3.5 with examples from each in figure 3.6). The environment presents realistic challenges for localization, such as structural ambiguity, partial observability due to occlusions and unmapped objects.

Each area presents distinct geometrical characteristics:

- **Area A** is the largest of the three, and with the largest wall and ceiling surfaces, which make it ideal for corner detection and use as natural landmarks. It does show some symmetry in its floor plan as it almost resembles a rectangle. This symmetry makes it more challenging to locate the robot given only the LiDAR reading. Furthermore, some unmapped objects are

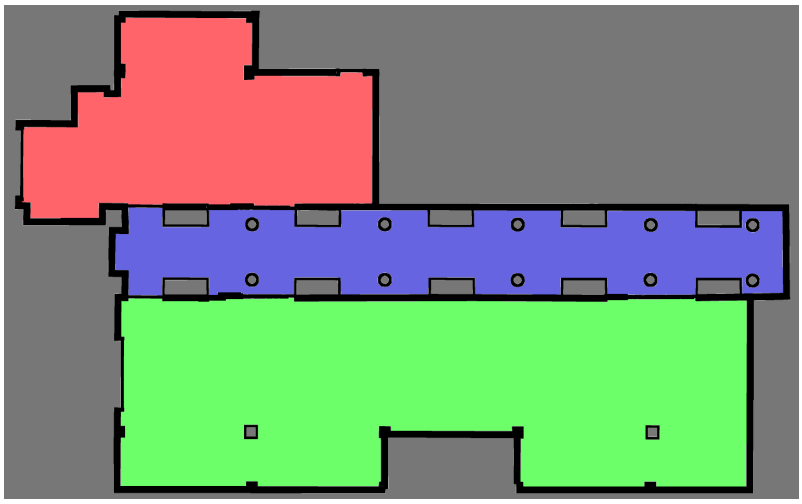


Figure 3.5: Areas A, B, and C (green, red and blue) of the iiLab map.

also placed throughout it, which increases the challenge level for the system to accurately detect the wall and other structural classes.

- **Area B** is the smallest area, and it also consists of smaller walls and a more compact environment, with more density of unmapped objects, which make it harder to extract the planes necessary for corner detection. Its floor plan also shows much less symmetry when compared to area A.

- **Area C** is a corridor, and it shows very high symmetry in its floor plan, as well as very small walls. Although it does not have any unmapped objects, it is the most challenging area to accurately locate the robot.

3.2 Point Cloud Processing

This chapter describes the point cloud processing pipeline, consisting of an unified framework, based on PointNet, able to perform point-wise segmentation and global place recognition. The objective of this module is to be able to not only detect relevant semantic features in the environment but also to filter out points that are not relevant, like those belonging to unmapped or dynamic objects.

As discussed in 2, PointNet is a neural network architecture specifically designed for directly processing point clouds. Unlike conventional image-based networks, PointNet is able to operate on unordered sets of data, being invariant to permutations between points. It is also able to give the same output, regardless of the rotation of the input point cloud, showing rotation invariance, which is valuable for both tasks of segmentation and place recognition.

As previously mentioned, in this work, the PointNet architecture is used to perform two complementary tasks:

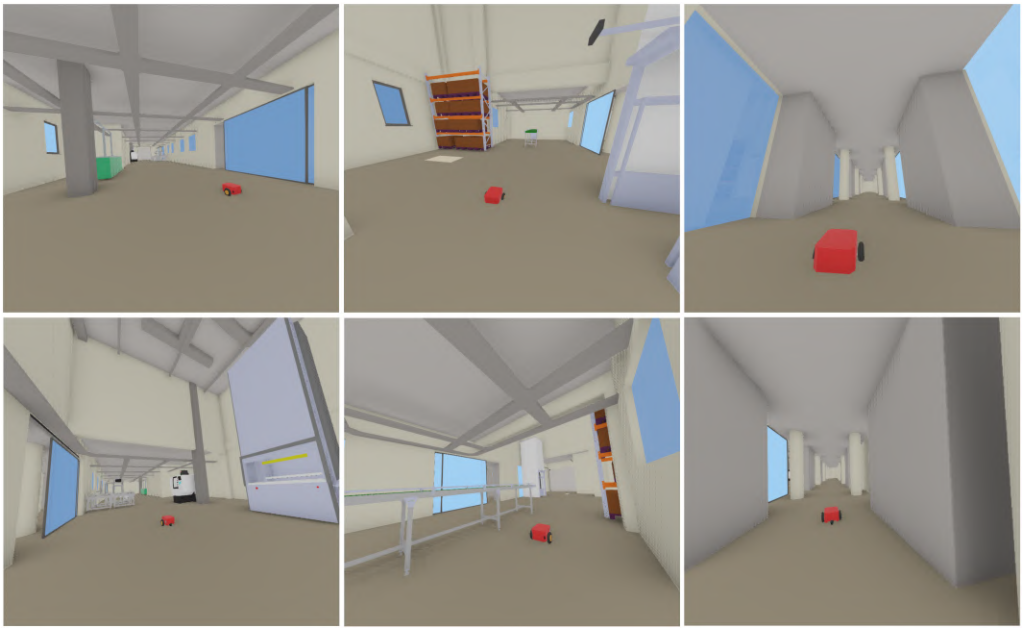


Figure 3.6: Examples from the inside of iiLab (Areas A, B and C, from left to right).

- Semantic segmentation: Each point is classified into structural categories (ceiling, floor, wall and clutter).
- Place recognition: The global feature vector generated by the network is used to identify the location of the robot at the time of capture.

To support the training of this model, a custom dataset was generated in simulation using the Webots robotics environment. A custom ROS pipeline was developed to automate the collection of labeled point clouds.

3.2.1 Dataset

The PointNet model requires a labeled dataset of point clouds for supervised training. To meet this requirement, a custom dataset was generated using simulated data from the Webots environment. This section describes the dataset creation process, including the sampling of robot poses within the environment and the labeling of points according to the structural categories (wall, floor, ceiling and clutter).

Dataset creation

Firstly, a custom ROS node was developed to automate the data collection process. It interfaces with the occupancy grid stored in the .pgm map file to identify free-space regions where the robot can be safely positioned, these being regions where the pixels are white in color. A predefined number of valid locations are randomly selected from these regions. At each selected location, the robot is placed within the simulation, with a constant orientation, and a point cloud is acquired

using the LiDAR sensor. Both the raw point cloud and the corresponding ground truth robot pose are recorded for each sample. This pipeline is visually explained in figure 3.7.

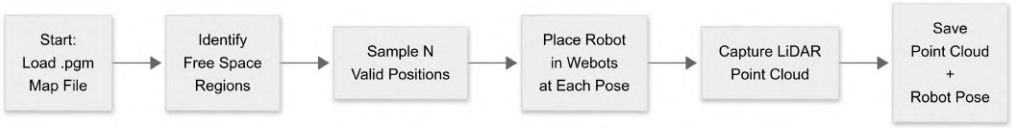


Figure 3.7: Dataset Creation pipeline.

This process was carried out separately for each defined area within the iiLab map, using a distinct .pgm file corresponding to each. The areas were defined arbitrarily but with the intentional consideration that each one was chosen to be structurally distinct, ensuring that point clouds captured within a given area would share similar geometric characteristics.

For the purpose of labeling the dataset, separate 3D mesh files (.obj) were created for each semantic class in the environment: walls, ceilings, floors. An additional mesh representing clutter was also created to make the labeling more accurate. This clutter mesh includes columns, windows, beams, air vents and all the machines and equipment. The 4 meshes are shown together in figure 3.8. From each mesh, points were uniformly sampled to create a representative point cloud for that class. These points were then indexed using a Kd-Tree structure for efficient nearest-neighbor queries.

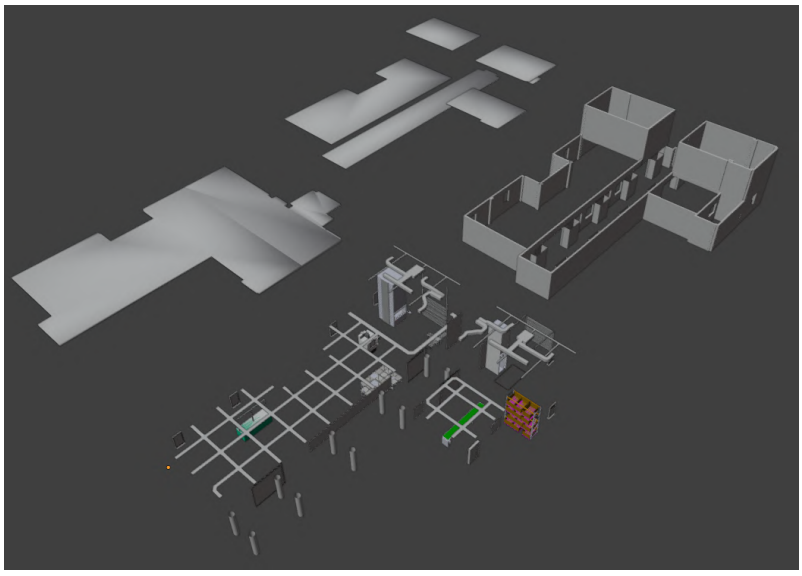
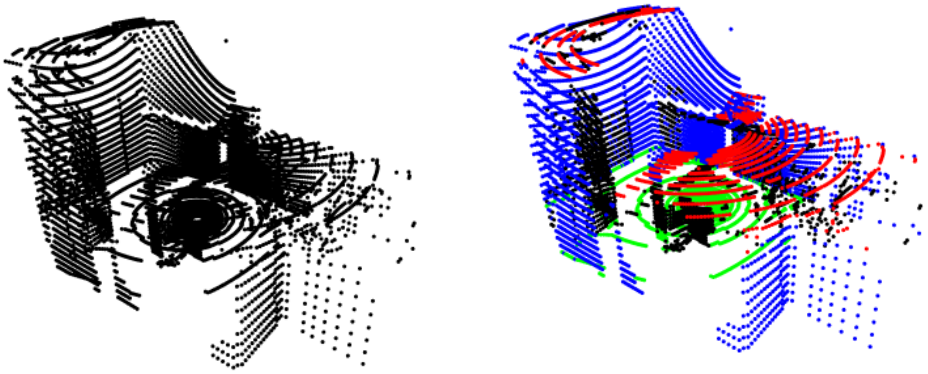


Figure 3.8: The four classes used for the dataset: ceiling, floor, wall and clutter.

During labeling, each recorded point cloud was first transformed into global coordinates using the corresponding ground truth pose. For every point in the transformed cloud, the nearest point in the Kd-Tree was identified, and its class label was assigned. If no nearby labeled point was found within a predefined distance threshold, the point was classified as clutter. This method enabled

automated segmentation of all collected point clouds without manual annotation, while preserving high labeling accuracy. An example of this labeling pipeline is illustrated in figure 3.9.

This entire procedure resulted in the labeling of 2300 LiDAR readings, thus creating an appropriate dataset for the subsequent training of the area inference and segmentation models.



(a) Raw Point Cloud.

(b) Ceiling (red), wall (blue), floor (green) and clutter (black) classes.

Figure 3.9: Labeling example from Area A.

3.2.2 Segmentation task

Following dataset generation and labeling, the PointNet based model was trained to perform point-wise semantic segmentation. This section describes the entire training process.

Data preparation and augmentation

Firstly, the dataset was divided into two distinct subsets, to be used as training and validation datasets, with an 80/20 ratio. The created datasets had 1840 and 460 labeled LiDAR readings, respectively, each with 10,000 points.

To mitigate class imbalance during training, class-specific weights were computed based on the relative frequency of each label in the dataset. For each point cloud, the corresponding label distribution was calculated using a histogram over the label values. These counts were accumulated across the dataset to obtain the total number of occurrences for each class.

The resulting frequency vector was then normalized to form a probability distribution. To derive the final weighting factors, an inverse frequency scaling was applied. Specifically, each class weight was computed as the cube root of the ratio between the maximum class frequency and the individual class frequency:

$$w_c = \left(\frac{\max(\mathbf{f})}{f_c} \right)^{1/3} \tag{3.6}$$

where f_c is the normalized frequency of class c, and f is the entire class frequency distribution. This cube root scaling reduces the dominance of very rare classes while still emphasizing their importance during training. The computed weights were later used in the loss function to balance the contribution of each class.

To prevent the model from learning unexpected correlations between the absolute spatial coordinates of the points and their semantic labels, each point cloud is preprocessed through centering and normalization. Centering is performed by subtracting the mean of all points, aligning the point cloud's centroid with the origin. Normalization then scales the point cloud such that its spatial extent fits within a unit sphere, achieved by dividing all points by the maximum Euclidean distance from the origin to any point in the cloud:

$$\mathbf{p}'_i = \frac{\mathbf{p}_i - \bar{\mathbf{p}}}{\max_j \|\mathbf{p}_j - \bar{\mathbf{p}}\|}, \quad \forall i \tag{3.7}$$

where $\mathbf{p}_i$ denotes a point in the original point cloud, $\bar{\mathbf{p}}$ is the mean of all points, and $\mathbf{p}'_i$ is the transformed point.

Finally, a random rotation around the vertical (z) axis is applied to each point cloud during training. The rotation angle is uniformly sampled from the interval $[0, 2\pi[$, with the objective of promoting rotation invariance in the learned feature space.

Training procedure

The PointNet encoder architecture and logic have already been described in detail in Chapter 2. In this work, the full set of 10,000 points from each LiDAR reading is used directly as input to the PointNet encoder. This constitutes a notable deviation from the original PointNet implementation, which processes the input point cloud in smaller, spatially partitioned blocks. The motivation behind using the entire point cloud was through empirical evaluation that showed a reduced inference time by eliminating the need for overlapping block processing and post-aggregation steps.

The segmentation network architecture then builds upon the global features extracted by the PointNet encoder. Each point is represented by a 1088-dimensional feature vector, composed of 64-dimensional local features and a 1024-dimensional global feature that is concatenated and shared across all points. This feature vector is the input for the classification head, which consists of a multi layer perceptron.

The classification head consists of four layers: the first reducing the feature size from 1088 to 512, followed by layers of size 256, 128, and finally 4, where 4 corresponds to the number of semantic classes (e.g., wall, ceiling, floor, object). Each intermediate layer is followed by batch normalization and a ReLU activation function, except for the final layer which outputs the class scores per point, by applying a softmax function.

During training, batches of point clouds are processed simultaneously. For each batch, a forward pass through the network is performed to produce point-wise predictions, which are then compared to the ground truth labels using a cross-entropy loss function. The parameters of the entire network are then updated using the ADAM optimizer.

To improve the robustness of the learned feature transformations, a feature transform regularizer was also employed, as proposed in the original PointNet paper. This regularizer encourages the learned transformation matrices to be close to orthogonal, thereby preventing degenerate or trivial transformations that could harm performance.

The final loss returned during training is the sum of the cross-entropy loss and the scaled regularization loss. This approach effectively constrains the learned transformation matrices to approximate orthogonal matrices, improving generalization and stability during training.

The model was trained for 100 epochs in total on a workstation with an AMD Ryzen Threadripper PRO 7965 2 and an NVIDIA RTX 6000 Ada Generation 3, using the hyperparameters listed in table 3.1:

Table 3.1: Training hyperparameters used for PointNet segmentation

Hyperparameter	**Value**
Batch size	16
Learning rate	0.001
Learning rate decay	0.7 every 20 epochs
Loss function	Cross-entropy + regularization
Feature transform regularization weight	0.001

At the end of the training, the model showed the results in table 3.2.

Table 3.2: Segmentation performance metrics

Metric	**Value**
Training Accuracy	0.85
Validation Accuracy	0.82
Mean IoU	0.69
Floor Class IoU	0.930
Ceiling Class IoU	0.648
Wall Class IoU	0.619
Clutter Class IoU	0.483

3.2.3 Place recognition task

This section introduces the hypothesis that, by training on a task which implicitly would require some type of scene understanding and description like structural segmentation within a map (e.g., a point in the same coordinates in the LiDAR's reference can be classified as different classes in different readings, it is only the global context that changes), the encoder of the network learns to describe the unique geometrical characteristics of a reading.

On the context of this work, similar geometrical characteristics between two readings typically mean that the readings were captured near each other, or at least in places that share many geometrical characteristics, like places inside the same area in the map. This set of learned geometrical characteristics is encoded in a vector of size 1024, the global descriptor, generated by the encoder. One can imagine a set of readings, represented by global descriptors in a 1024-dimensional hyperspace, where proximity between two descriptors in that space is associated with spatial proximity, or at least similar geometric characteristics, between the places where those readings where captured. A representation of this logic is shown in figure 3.10.

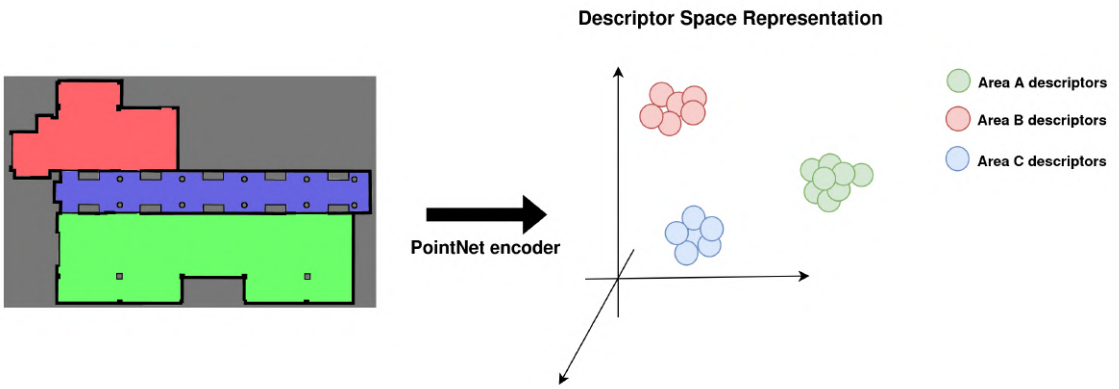


Figure 3.10: Simplified Descriptor space representation for each area of the map.

To validate that the learned global descriptors form distinct clusters for different areas, the average pairwise distance between descriptors within each area (intra-area), and between descriptors of different areas (inter-area), excluding self-distances, was calculated. In all cases, intra-area average distances were lower than inter-area distances, indicating that the descriptor space effectively encodes spatial proximity and forms separable clusters per area. These distances are shown in table 3.3.

Table 3.3: Average Euclidean distance between pairs of global descriptors. Diagonal values represent intra-area averages.

Area	**A**	**B**	**C**
A	**4.47**	4.90	4.65
B	4.90	**4.44**	5.26
C	4.65	5.26	**4.12**

This property of the descriptor space was used in this work to focus the particle distribution of the particle filter around likely positions. To achieve this, a data structure was created to associate each global descriptor with its corresponding position in the map, using all available readings from the dataset. Each point cloud was randomly rotated before global descriptor generation, to

prevent an orientation bias. This structure effectively constitutes a fingerprint map, an environment representation discussed in chapter 2.

This idea of learning global descriptors from raw point clouds for the purpose of localization is inspired by methods such as PointNetVLAD [41], which demonstrated that aggregating local features into a compact global representation can yield state-of-the-art results in large-scale place recognition tasks.

For efficient nearest-neighbor retrieval in the 1024-dimensional descriptor space, the FAISS [3] (Facebook AI Similarity Search) python library was employed. FAISS enables fast similarity search across large datasets of dense vectors by constructing an optimized index structure. During runtime, the global descriptor computed from the current LiDAR reading is queried against this index, returning the closest matches from the previously computed fingerprint map. The corresponding positions of these nearest descriptors are then used to seed particles of the filter, in an attempt at improving convergence and robustness in global localization.

To evaluate this place recognition pipeline, several metrics were used and tests were made for each area separately and for the total dataset, using 5 as the number of closest matches for each query. The metrics used are listed below:

- **Mean error**: The average distance from the ground truth position to the 5 retrieved positions.
- **Mean minimum error**: The average distance from the ground truth position to the closest of the 5 retrieved positions.
- **Mean maximum error**: The average distance from the ground truth position to the furthest of the 5 retrieved positions.

The results are summarized in table 3.4

Table 3.4: Pose retrieval error statistics per area.

Area	Mean Error (m)	Mean Min Error (m)	Mean Max Error (m)
Area A	1.69	0.28	3.54
Area B	1.06	0.10	2.49
Area C	2.88	0.42	6.14
Total	**1.87**	**0.27**	**4.01**

These results, despite being overly optimistic due to using the same point clouds the model was trained on, show that the place recognition task is achievable by the model, and also present an interesting metric for how geometrically ambiguous each position inside each area is, with Area B showing the least amount of ambiguity and area C showing the most. This is also clear by looking at the map and analyzing the geometry of each area.

[3] https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

3.2.4 Global descriptor matching pipeline

The pose suggestion pipeline leverages the place recognition capabilities described in Section 3.2.3 to provide initial pose estimates for the particle filter. This pipeline operates in real-time, processing incoming LiDAR point clouds and generating pose suggestions based on similarity matching in the global descriptor space.

Real-time Descriptor Generation and Matching

Upon receiving a new LiDAR point cloud reading, the system first processes it through the trained PointNet encoder to generate a 1024-dimensional global descriptor. This descriptor encapsulates the geometric characteristics of the current observation and serves as a compact representation for similarity matching against the pre-computed fingerprint map.

The similarity search is performed using the FAISS library, which enables efficient exact nearest-neighbor retrieval in the high-dimensional descriptor space. For each query descriptor, the system retrieves the k closest matches from the fingerprint map, where $k = 3$ was chosen to balance between providing sufficient pose diversity and maintaining computational efficiency.

Spatial Consistency Validation

To ensure the quality of pose suggestions, the pipeline implements a spatial consistency check based on the spread of retrieved poses. After obtaining the k nearest descriptors and their corresponding 2D positions (x, y), the system calculates all pairwise Euclidean distances between these positions using:

$$d_{max} = \max_{i,j} \|p_i - p_j\|_2 \tag{3.8}$$

where p_i and p_j represent the 2D positions of the retrieved poses. A spread threshold of 10.0 meters is applied to filter out inconsistent suggestions:

$$\text{Valid suggestions} = \begin{cases} \{p_1, p_2, \ldots, p_k\} & \text{if } d_{max} \leq 10.0 \text{ m} \\ \emptyset & \text{otherwise} \end{cases} \tag{3.9}$$

This validation mechanism prevents the particle filter from being biased with contradictory pose estimates that could arise from geometric ambiguities or descriptor matching errors.

Integration with Particle Filter

The validated pose suggestions are published as a PoseArray message in the ROS2 framework, making them available to the particle filter localization node. These suggestions serve as seed positions for particle initialization, particularly beneficial during:

- **Global localization**: When the robot's initial position is unknown

- **Kidnapped robot scenarios**: When the robot is moved to an unknown location
- **Recovery from tracking failures**: When the particle filter loses track of the robot's position

The pipeline also provides visualization support through RViz markers, displaying the suggested poses as red spheres in the map frame.

Computational Performance

The pose suggestion pipeline operates with minimal computational overhead. The global descriptor generation through the PointNet encoder typically completes within the inference time reported in the segmentation task, while the FAISS similarity search provides sub-millisecond query times for the fingerprint map size used in this work. This efficiency enables real-time operation without significantly impacting the overall system performance.

3.3 Corner detection

The module works by processing the previously segmented point cloud. Plane models are extracted independently from the wall and ceiling point sets using a RANSAC algorithm. Then, potential corners are generated by computing intersections between combinations of two wall planes and one ceiling plane. These candidate intersections are validated based on the proximity of the intersection point to the three planes' inliers. For each valid corner, the orientation and covariance are computed. The resulting information is encapsulated in a custom ROS message and published to a designated topic. The full processing pipeline is illustrated in Figure 3.11.

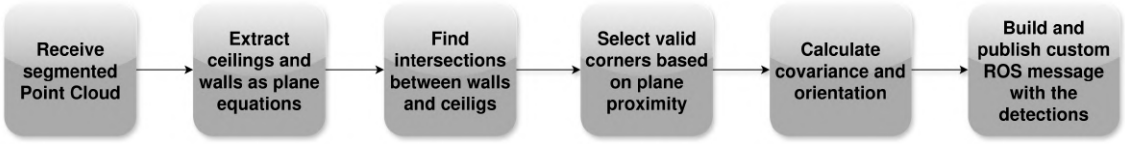


Figure 3.11: Corner Extractor Pipeline.

3.3.1 Plane extraction

To extract the dominant wall and ceiling surfaces, the RANSAC algorithm was employed. More specifically the implementation by the python library Open3D was used.

The core idea behind RANSAC is to iteratively fit a model to randomly sampled subsets of the data and evaluate its support from the remaining points. In the case of plane fitting, each model is defined by the plane equation:

$$ax + by + cz + d = 0 \tag{3.10}$$

The algorithm starts by sampling three points from the point cloud and computing the plane equation defined by them. After this, the inliers of that plane are selected, by calculating the

perpendicular distance from each point to the plane. Points whose distance fall below a user-defined threshold are considered inliers. Then, the model is scored based on the number of inliers, and if that score exceeds the current best model's score, it is saved as the new best model. This process is repeated for a number of iterations, defined by the user, and the final model is the one with the best score. Once the dominant model is found, its inliers are removed from the point cloud and the process can be repeated to detect additional planes. The algorithm is summarized in figure 3.12

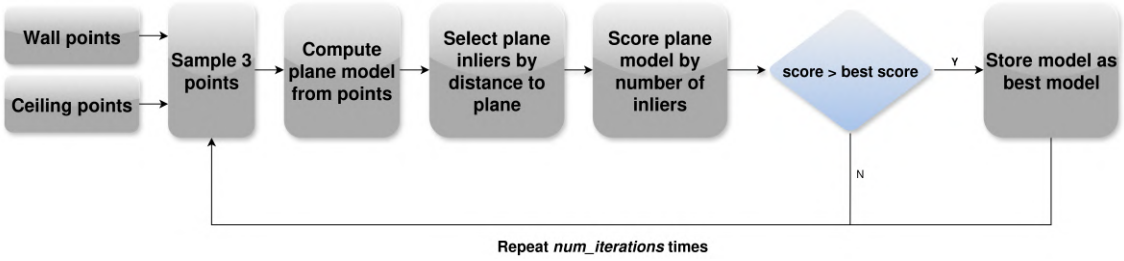


Figure 3.12: The RANSAC plane extraction algorithm

In this work, this algorithm is applied only to the points classified as wall and ceiling, separately, as the visual example in Figure 3.13 shows.

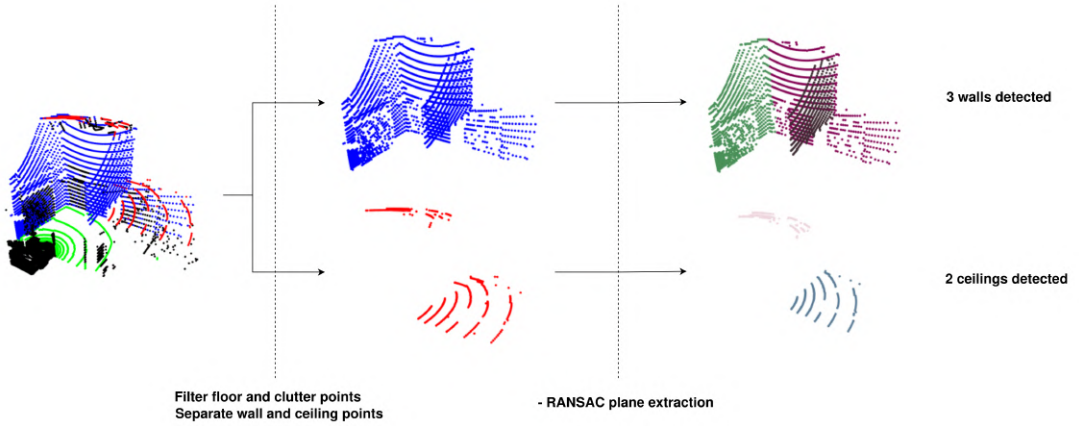


Figure 3.13: Example of plane extraction from area A

3.3.2 Corner Validation

Once the dominant wall and ceiling planes have been extracted, the next step in the pipeline involves detecting their intersections, as potential structural corners. In the context of this work, a corner is defined as the intersection point between two vertical wall planes and one horizontal ceiling plane.

Given three planes, their general equations can be expressed as:

$$a_i x + b_i y + c_i z + d_i = 0, \quad \text{for } i = 1,2,3 \tag{3.11}$$

where (a_i, b_i, c_i) is the normal vector of the i-th plane, and d_i is the plane's offset.

The goal is to find the point $p = [x, y, z]^T$ that satisfies all three equations. This can be written in matrix form as:

$$A\mathbf{p} = \mathbf{b} \tag{3.12}$$

where

$$A = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -d_1 \\ -d_2 \\ -d_3 \end{bmatrix} \tag{3.13}$$

To solve for $\mathbf{p}$, the system is solved using

$$\mathbf{p} = A^{-1}\mathbf{b} \tag{3.14}$$

and in this work, via the numerical solver `np.linalg.solve` in Python.

This method is valid when matrix A is invertible, which occurs if no planes are parallel and their normal vectors are linearly independent. If the system is singular or nearly singular, the planes are coplanar or intersect along a line, so no unique solution exists, and the intersection point is undefined.

As the method to detect the intersections uses the plane equations, which define infinite planes, intersections may appear between walls and ceilings which are disconnected in the real environment. On the other hand, the ability to estimate a corner when its vertex is not fully visible to the LiDAR is valuable, as the corner itself might just be occluded from view. The optimal solution found was to validate the intersection based on a proximity threshold to the plane inliers.

3.3.3 Corner orientation and covariance

After the intersection point has been validated, the next step is to estimate the orientation of the corner. This is done by first computing the centroids of the inlier points for each wall plane. Vectors are then formed from the intersection point to each centroid and subsequently normalized. The final corner orientation is defined as the bisector vector, obtained by summing the two normalized vectors. Since this work assumes a 2D feature map, the resulting orientation vector is projected onto the horizontal plane (z=0). This projected vector provides a representative direction of the corner, lying midway between the two intersecting wall surfaces. For visualization purposes, the diagram in Figure 3.14 represents the elements involved in the calculation.

The covariance values associated with the corner's position and orientation estimates were determined empirically. Rather than being derived from a formal statistical model, these values were manually tuned based on qualitative observations of the system's behavior during simulation. The selected values reflect a balance between confidence in the detections and the need for robustness in the sensor fusion process.

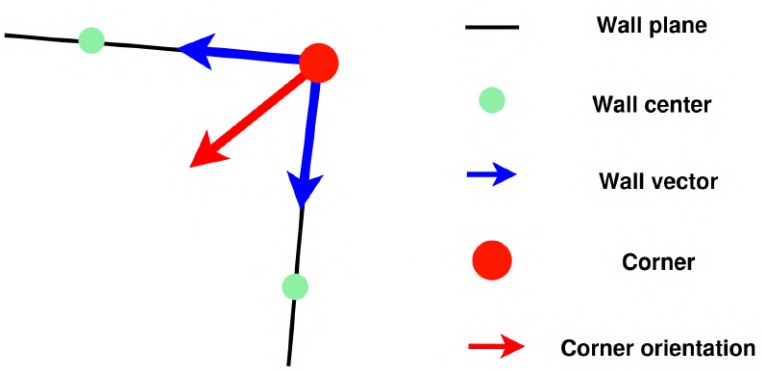


Figure 3.14: Orientation estimation visualization.

Finally, for the purpose of connecting this module to the rest of the localization system, specifically to the particle filter, a custom ROS message type named Feature.msg was defined. This message encapsulates the relevant information for a single detected corner, including its position, orientation, and covariance. Multiple features detected in a single processing cycle are aggregated into a FeatureArray.msg, which is then published to the designated topic.

3.3.4 Parameters

The corner detection algorithm's behavior is controlled by several key parameters that affect plane extraction, intersection computation, and corner validation:

- **Maximum planes**: Maximum number of planes to extract from each point cloud segment. Set to 20 for walls and ceilings to ensure comprehensive plane detection while maintaining computational efficiency.
- **Distance threshold**: RANSAC distance threshold in meters for determining point-to-plane membership during plane extraction. Value of 0.20 m provides robust plane fitting while tolerating sensor noise.
- **Minimum inliers**: Minimum number of points required to consider a detected plane as valid. Set to 150 points to filter out spurious small plane segments and ensure statistical significance.
- **RANSAC iterations**: Number of RANSAC iterations for plane fitting. Value of 100 iterations balances computational cost with plane detection accuracy.
- **Verticality threshold**: Threshold for determining if a plane is vertical based on the z-component of its normal vector. Set to 0.1 for wall detection, meaning planes with $|n_z| < 0.1$ are considered vertical. This because it is possible to extract horizontal planes from the wall points, instead of the desired vertical planes.

- **Plane conditioning threshold**: Angular threshold in radians for ensuring three planes are well-conditioned for intersection. Set to 45 ($\pi/4$ rad) to avoid degenerate intersections from nearly parallel planes.
- **Corner-to-plane distance threshold**: Maximum allowed distance in meters from a computed corner to any of its constituent planes. Set to 2.0 m to validate that corner intersections are geometrically consistent with the detected planes.
- **Corner proximity threshold**: Minimum separation distance in meters between detected corners to avoid duplicates. Set to 0.5 m to merge nearby corner detections that likely represent the same physical corner.

3.4 Particle Filter

3.4.1 Main functional blocks

As previously mentioned, the detected corners are used as the external measurements for a custom particle filter. The process of this filter will be explained, separating it in 5 main functional blocks:

- **Motion Update:** Each particle is updated with the detected motion from the wheel encoders of the robot.
- **Measurement Update:** Each particle's weight is recalculated by matching the detected corners with the mapped corners.
- **Resampling:** Particles are redistributed with the aim of creating more particles around more likely poses and destroying those in unlikely poses.
- **Particle Injection:** The particles with the least weight are moved to different poses. A part of the set of new poses is calculated from the pose inference pipeline, explained in 3.2.3, while the other is randomly selected from the map.
- **Pose estimation:** A pose is estimated from the distribution of the particles.

3.4.2 Process

Motion Update

The odometry pose of the robot relative to its starting pose is retrieved from the ROS transform tree via the `map` $\rightarrow$ `base_footprint` transform , and the displacement in position and angle since the previous motion update is calculated . If this displacement is above a given threshold, each particle is updated with by adding those displacements (with added gaussian noise in position and angle), the pose of the robot is saved for future comparison and the measurement update block is activated.

The implementation of motion thresholds prevents unnecessary computation, assuming that if no significant motion is detected through odometry, the pose of the robot and the state of its surrounding environment have not changed. Furthermore, it also prevents premature convergence to a given pose if the robot does not move but keeps receiving measurements.

Measurement Update

if the robot has moved enough, this block is activated, and the last stored feature array message is used to update the weights of each particle by matching the received features with the stored features.

Each feature array message consists of a timestamp and a set of features, each with the position (x, y), the angle (θ) and the covariance values for each (σ_{xx}, σ_{yy} and $\sigma_{\theta\theta}$). Each message also contains the confidence of the detection and the type of feature, which in this case always take the values of 1 and *"corner"* respectively. The covariances between the two position coordinates are assumed to be 0.

```
1 string  type                     # "corner"
2 float64 confidence               # 1.0
3 float64 x
4 float64 y
5 float64 theta
6 float64[4] position_covariance   # [cov_xx, 0.0, cov_yx, 0.0]
7 float64 orientation_variance     # cov_theta
```

Listing 3.1: Feature.msg structure

The feature array message timestamp is regarding the moment of capture of the scan that originated that set of features. This timestamp is used to lookup the robot's odometry pose, `map` $\rightarrow$ `base_footprint` transform ,at that moment, calculating the transformation to the current pose and subtracting it from each particle. Finally, for each particle, the features are transformed to the particle's corrected frame of reference. This means effectively going back in time to the moment of scanning, because the robot may have moved since, and therefore guaranteeing temporal coherence between the particle's pose and the received features.

This temporal coherence is often not a problem in systems working directly with raw scans, because there is no processing step. As the segmentation and corner detection steps occur between the moment of scanning -*t_scan*- and the moment of updating the particles -*t_update*- the motion of the robot between those moments would alter the matching scores. Without the synchronization, the filter would be matching features from *t_scan* using the pose of the robot in *t_update*, which would imply a drift in the estimated pose, lagging behind the real pose.

After the features are transformed to each particle's corrected frame of reference, each feature is matched via proximity to a corner in the map. The distance to that corner - Δ_{pos} - and the normalized difference in orientation regarding that corner - Δ_{α} - are subsequently calculated and

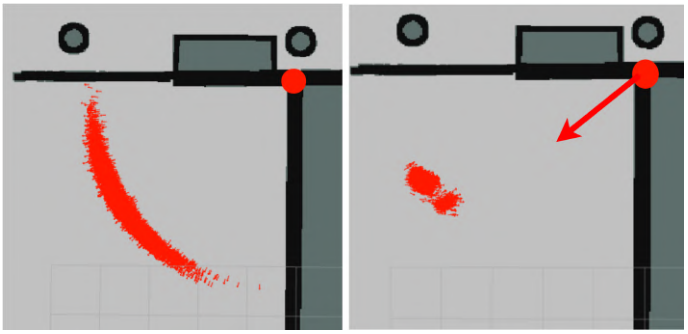


Figure 3.15: Particle distribution in the case of a corner without orientation (left) and a corner with orientation (right).

converted into a per feature likelihood using equation 3.15, where σ_{pos} is defined as $\sqrt{\sigma_{xx}^2 + \sigma_{yy}^2}$. The equation's two exponents are referring to the distance likelihood and the angle likelihood, respectively. By combining the angle likelihood, the set of likely poses is no longer a circumference around the feature but only a smaller arc of that circumference, as shown in figure 3.15.

$$L = \exp\left(-\frac{\Delta_{\text{pos}}^2}{2\sigma_{\text{pos}}^2}\right) \cdot \exp\left(-\frac{\Delta_{\alpha}^2}{2\sigma_{\theta}^2}\right) \tag{3.15}$$

After all the features are converted into likelihoods, the particle's weight is updated. Typically, each feature's likelihood would be added or multiplied together to obtain a complete measurement likelihood, which would then be multiplied with the particle's weight to calculate the new weight. However, in the operation of the filter, there was a need to directly control the significance that each measurement should have and the extent that each measurmeent affects the weight of the particle. With this in mind, a learning rate - α - was introduced so that this significance could be a parameter of the filter. The equation 3.16 employs this mechanism.

Additionally, the total measurement likelihood was defined as the sum of each individual feature's likelihood cubed. This is less rewarding for good particles but also less punitive for worse particles that might be close to the right pose.

$$w' = \alpha \cdot (w \cdot L_{\text{total}}) + (1 - \alpha) \cdot w \tag{3.16}$$

where w' is the new weight for the particle, L_{total} is the total measurement likelihood and w is the current weight of the particle.

To further guide the filter toward the correct pose, particles whose positions fall within known occupied areas of the map are penalized by reducing their weights. This mechanism, for example, explains why the particle distribution on the left side of Figure 3.15 does not form a full circumference around the corner—particles that entered occupied regions were down-weighted and effectively removed from consideration.

Resampling

After the measurement update is applied to all particles, their weights are normalized by dividing each by the sum of all updated weights. The degeneracy of the particle set is then evaluated. Degeneracy refers to the phenomenon in which, over successive iterations, the importance weights become increasingly imbalanced—most particles carry negligible weight, and only a few contribute meaningfully to the estimate. This effect is quantified using the Effective Sample Size (ESS), which provides a measure of how well the weighted particle set represents the underlying distribution. The formula 3.17 was used to calculate the ESS.

$$\text{ESS} = \frac{1}{\sum_{i=1}^{N} w_i^2} \tag{3.17}$$

When ESS approaches small values relative to the total particle count N, the filter's representational power degrades significantly. The implementation monitors ESS continuously and triggers resampling when ESS $\leq N \times \tau_{ESS}$, where the threshold τ_{ESS} (`resample_ess_threshold`) is a configurable parameter.

If the ESS condition is met, the resampling block is activated, and a new, more representational set of particles is created. This is done using the residual resampling method, which is explained in detail in chapter 2.

Particle injection

While the resampling block deals with degeneracy, it can lead to a lack of diversity in the set, where there are large areas of the map with no particles.

To mitigate the lack of spatial coverage of the particle set, parts of the map are repopulated with particles via worst-particle replacement.This method relocates the worst particles in the set to other positions. These positions are defined by the suggested positions from the pose retrieval pipeline. Gaussian noise is added to each particle to generate a normal distribution of particles around each suggested position. The orientation of these particles is randomly set, since the pipeline only suggests positions. This relocation, assuming the position suggestions are fairly accurate, increases the coverage of the distribution in more likely positions, theoretically enabling faster convergence to the right pose. Since only a small percentage of particles is relocated, the effect on the filter is very small, but enough to collapse ambiguities.

Pose estimation

The final stage of the filter estimates the robot's pose based on the current particle distribution. A dual-mode estimation strategy is employed: the pose can either be inferred from the odometry since the last estimate or computed as a weighted average of the positions and orientations of the top N particles, where N is a configurable parameter.

After resampling, particle weights are discarded, making weighted averaging no longer possible. Therefore, the choice of estimation method is guided by the Effective Sample Size (ESS).

When ESS falls below a defined threshold, indicating that the filter has likely converged, the estimate is taken from the best N particles. Conversely, if ESS is above the threshold—suggesting uncertainty in the distribution—the estimate is extrapolated from odometry to avoid premature reliance on an unstable particle set.

3.4.3 Parameters

The particle filter is governed by a set of configurable parameters, stored in a ROS 2 yaml file, that influence its behavior and performance. Table 3.5 presents the configurable parameters used in the implementation.

Table 3.5: Key particle filter parameters

Parameter	**Description**	**Value**
`num_particles`	Total number of particles in the filter	10000
`motion_threshold`	Minimum motion (m or rad) to trigger update	0.1 m / 0.01 rad
`resample_ess_threshold` (τ_{ESS})	ESS ratio to trigger resampling	0.3
`alpha`	Measurement update learning rate	0.8
`motion_x_variance`	Variance of motion in x-direction	0.001
`motion_y_variance`	Variance of motion in y-direction	0.001
`replace_worst_percentage`	Fraction of particles replaced using smart injection	0.05
`replace_worst_smart_sigma`	Positional noise standard deviation for smart injection	0.5 m
`estimate_num_particles`	Number of top particles used for pose estimation	100

Chapter 4

Results and discussion

4.1 Overview

PIMM was tested using 3 trajectories, designed to test the performance of the system in areas A and B separately and its performance on a trajectory passing through the 3 areas.

Each trajectory's description, length and duration are listed below:

Table 4.1: Overview of evaluated trajectories.

Trajectory	Length (m)	Duration (s)	Description
Area A trajectory	59.49	119.08	Trajectory inside Area B
Area B trajectory	26.89	73.54	Trajectory inside Area B
A to C to B	33.84	66.28	Transition from Area A through C to B

4.1.1 Evaluation metrics

Each approach was evaluated with the aim to evaluate their ability to converge to the true pose in a consistent and fast manner, while also being able to refine that pose more with subsequent measurements. Convergence time was defined as the moment when the positional and angular errors last go below their respective thresholds, 3 meters and 30 degrees. the last moment of convergence is used and not the first because it is possible for a system to converge initially to a pose only for it to converge later to another pose, especially in the initial phase of ambiguous maps.

Because the particle filter is a non-deterministic method, 10 trials were made for each trajectory to obtain a more accurate understanding of the performance of each system given the context. From the 10 trials, the metrics extracted were the following:

- **Convergence rate**: The percentage of successful trials, where the system was able to converge to right pose before the trajectory ended. This is meant to measure the capability of each system to collapse ambiguities consistently.

- **Average convergence time**: The average time in seconds it took for the system to converge in the successful trials.
- **Mean positional error:** The average positional error in meters after the moment of convergence.
- **Mean angular error:** The average angular error in degrees after the moment of convergence.

Additionally, to help visualize and understand the behavior of both systems, the best and worst trials are shown. The best trial is defined as the one with the lowest convergence time and the worst is defined as the one with the higher average positional error.

The sections 4.2,4.3, 4.4 and 4.5 show the results for each trajectory, with a visualization of the real trajectory, followed by the best/worst trajectories for each approach. Each section concludes with a table summarizing the results and a discussion.

4.2 Area A trajectory

This trajectory is executed entirely in Area A, which shows a high ambiguity in its floor plan and also in the location of the corners. The robot traverses the entirety of the area, showing, at some point, direct line of sight to each corner and each wall.

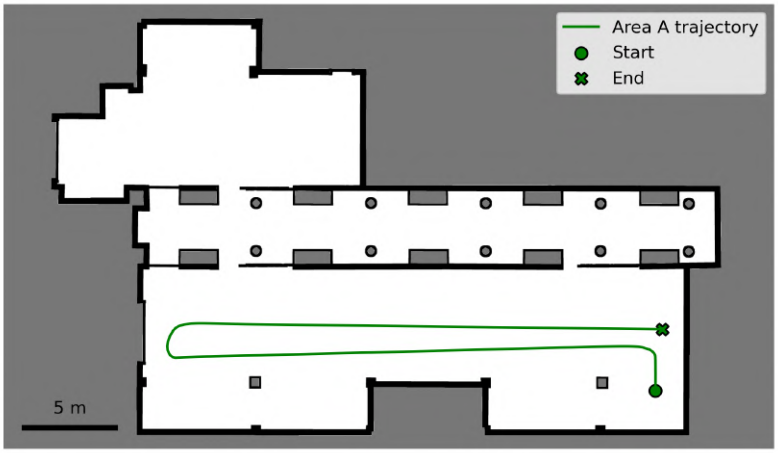
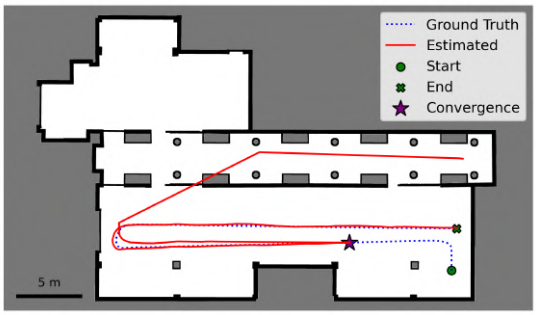
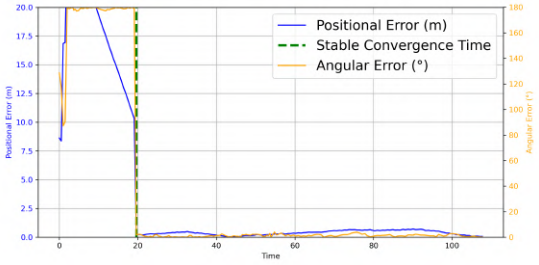


Figure 4.1: Area A trajectory.

AMCL

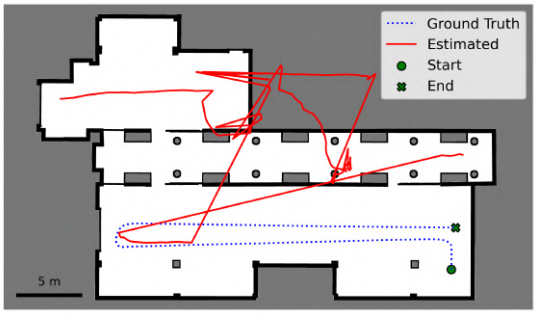


(a) AMCL best trial in Area A (trajectories).

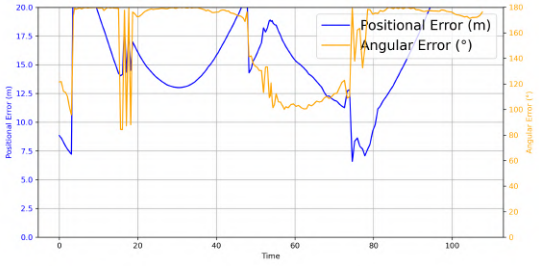


(b) AMCL best trial in Area A (errors).

Figure 4.2: AMCL best trial in Area A with no objects. (a) Trajectories. (b) Errors.



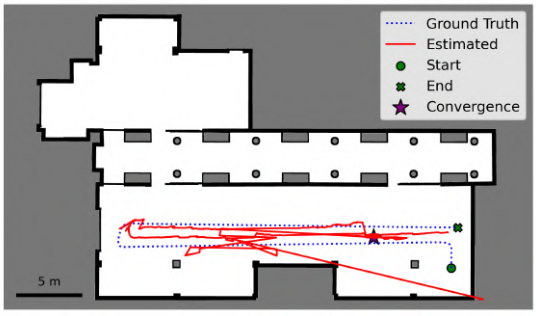
(a) AMCL worst trial in Area A (trajectories).



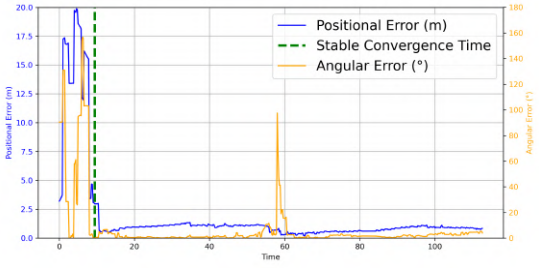
(b) AMCL worst trial in Area A (errors)

Figure 4.3: AMCL worst trial in Area A with no objects. (a) Trajectories. (b) Errors.

PIMM

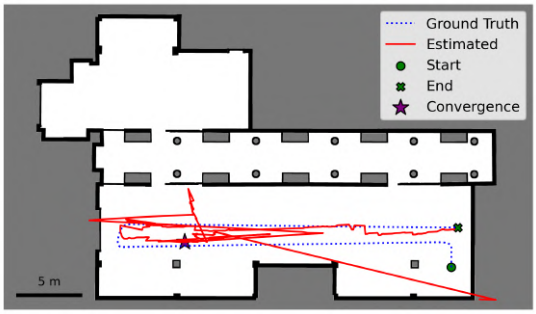


(a) PIMM best trial in Area A (trajectories).

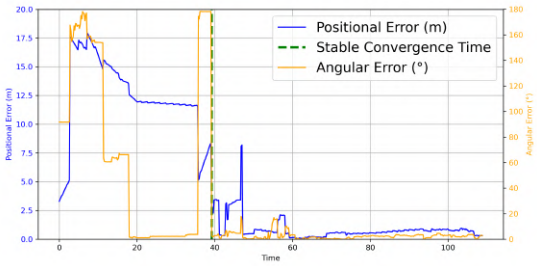


(b) PIMM best trial in Area A (errors).

Figure 4.4: PIMM best trial in Area A with no objects. (a) Trajectories. (b) Errors.



(a) PIMM worst trial in Area A (trajectories).



(b) PIMM worst trial in Area A (errors).

Figure 4.5: PIMM worst trial in Area A with no objects. (a) Trajectories. (b) Errors.

4.2.1 Performance summary

In this trajectory, PIMM outperformed AMCL in every metric except the mean post convergence angular error, with a notable increase in the consistency of convergence. This discrepancy can be explained by taking into account that PIMM receives 3D spatial information of the space surrounding the robot, allowing for it to surpass the ambiguity in the map more easily in the beginning of operation, than AMCL due to the increase in spatial information. For example, the leftmost part of Area A has a higher ceiling than the rightmost part. This is undetectable to AMCL, but the 3D point cloud captures this, and that information in encoded in the global descriptor, which allows for a larger concentration of particles in the rightmost area of the map in the beginning of operation.

Table 4.2: Performance comparison across approaches

Type	**Conv. Rate (%)**	**Conv. Time (s)**	**Pos. Error (m)**	**Ang. Error (°)**
AMCL	40	55.93	1.1	1.08
PIMM	100	20.88	0.84	3.2

4.3 Area B trajectory

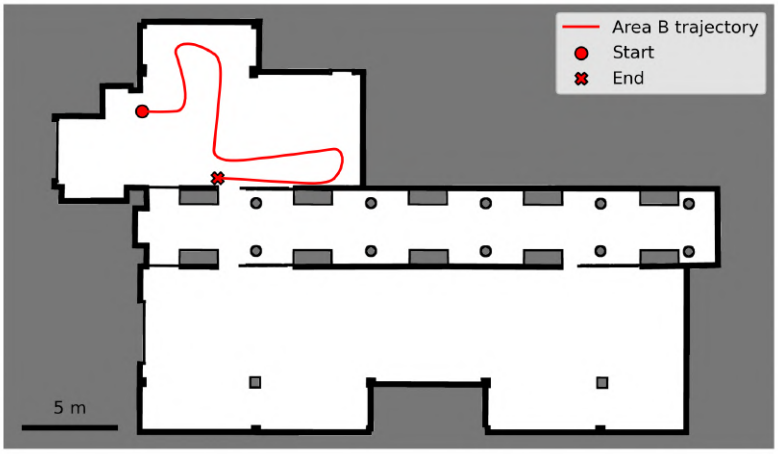
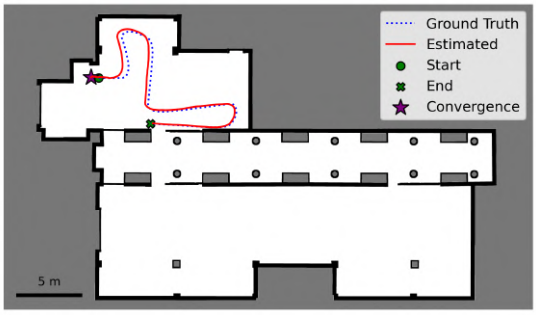
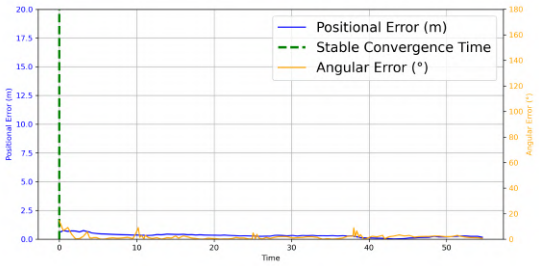


Figure 4.6: Area B trajectory.

AMCL

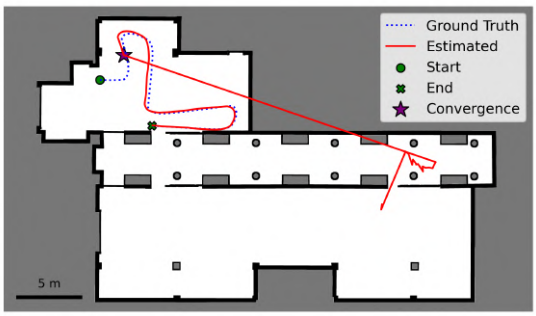


(a) AMCL best trial in Area B (trajectories)

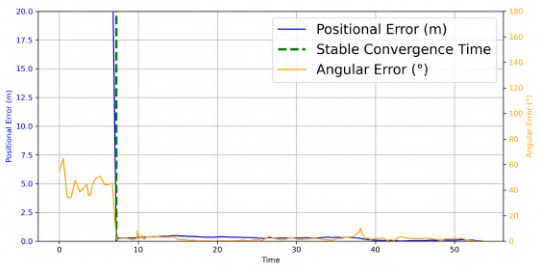


(b) AMCL best trial in Area B (errors).

Figure 4.7: AMCL best trial in Area B with no objects. (a) Trajectories. (b) Errors.



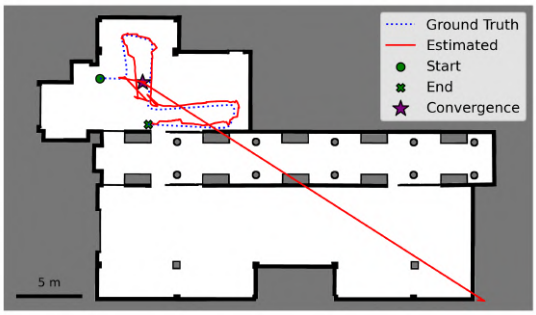
(a) AMCL worst trial in Area B (trajectories).



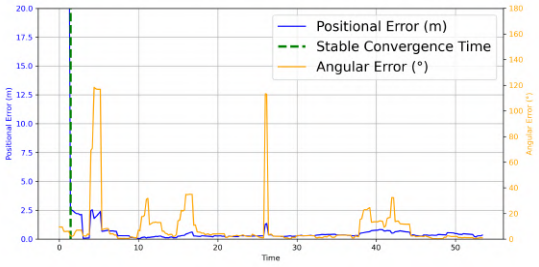
(b) AMCL worst trial in Area B (errors).

Figure 4.8: AMCL worst trial in Area B with no objects. (a) Trajectories. (b) Errors.

PIMM

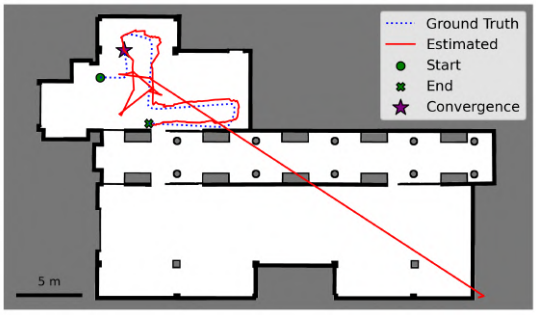


(a) PIMM best trial in Area B (trajectories).

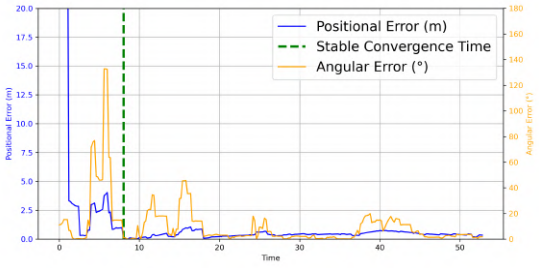


(b) PIMM best trial in Area B (errors).

Figure 4.9: PIMM best trial in Area B with no objects. (a) Trajectories. (b) Errors.



(a) PIMM worst trial in Area B (trajectories).



(b) PIMM worst trial in Area B (errors).

Figure 4.10: PIMM worst trial in Area B with no objects. (a) Trajectories. (b) Errors.

4.3.1 Performance summary

In this trajectory, AMCL slightly outperformed PIMM. This is explained by the uniqueness of the wall layout of area B within the entire map, which helps AMCL in converging. The discrepancy in overall post-convergence precision is explained by the sheer amount of features that AMCL has at its disposal to refine the reading/map match, which amounts to around 360 LiDAR points, in comparison with a few detected corners by PIMM.

Table 4.3: Performance comparison across approaches

Type	Conv. Rate (%)	Conv. Time (s)	Pos. Error (m)	Ang. Error (°)
AMCL	100	2.6	0.30	1.91
PIMM	100	4.53	0.50	7.65

4.4 A to C to B

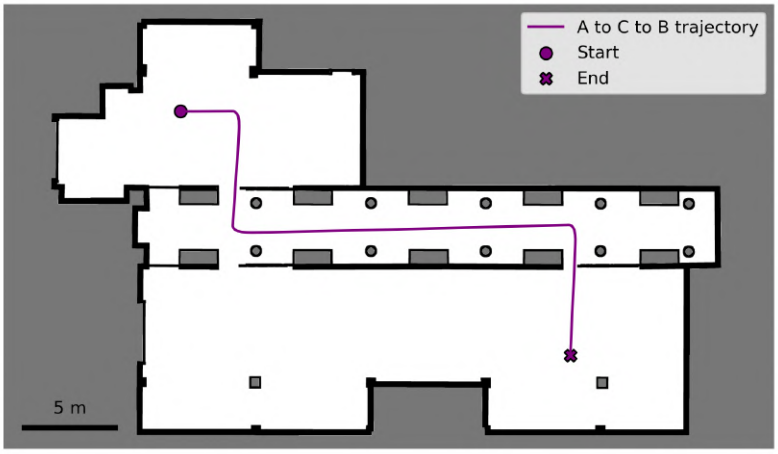
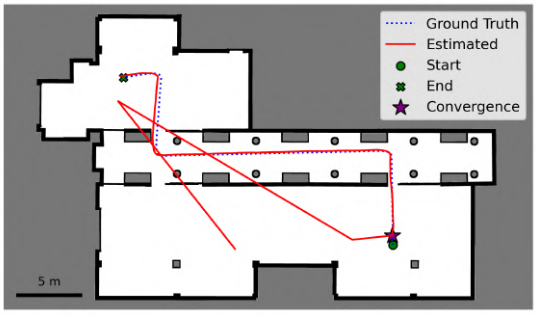
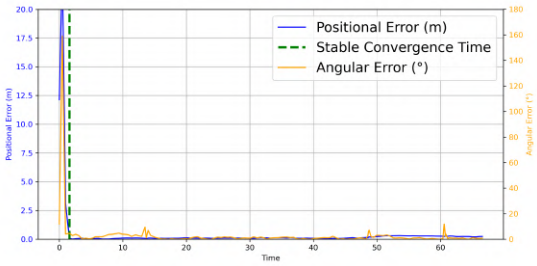


Figure 4.11: A to C to B trajectory

AMCL

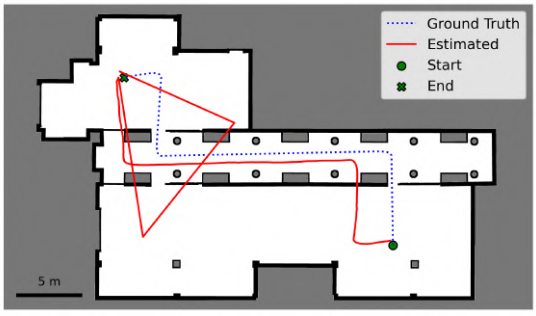


(a) AMCL best trial in A to C to B (trajectories).

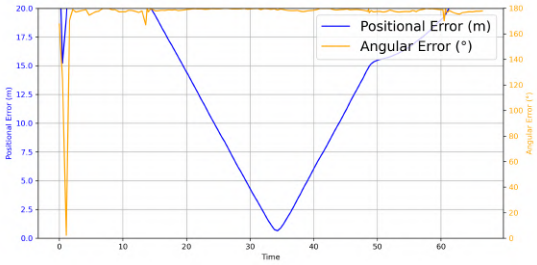


(b) AMCL best trial in A to C to B (errors).

Figure 4.12: AMCL best trial in A to C to B with no objects. (a) Trajectories. (b) Errors.



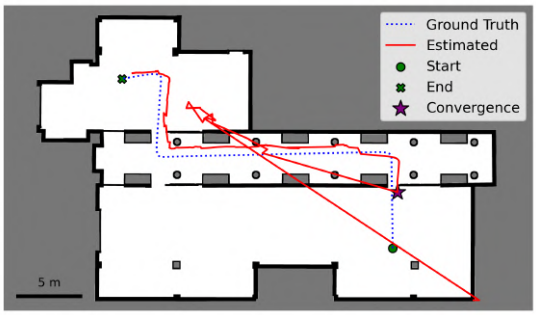
(a) AMCL worst trial in A to C to B (trajectories).



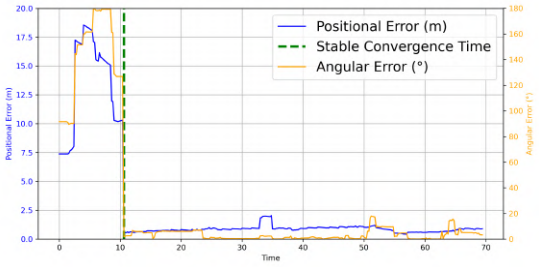
(b) AMCL worst trial in A to C to B (errors).

Figure 4.13: AMCL worst trial in A to C to B with no objects. (a) Trajectories. (b) Errors.

PIMM

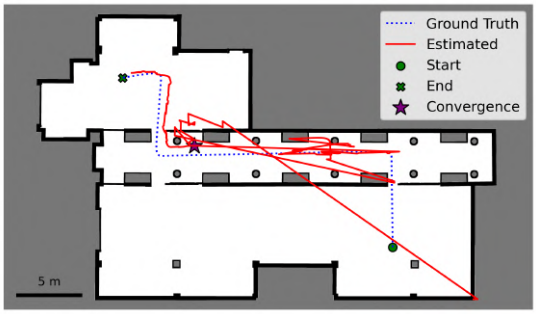


(a) PIMM best trial in A to C to B (trajectories).

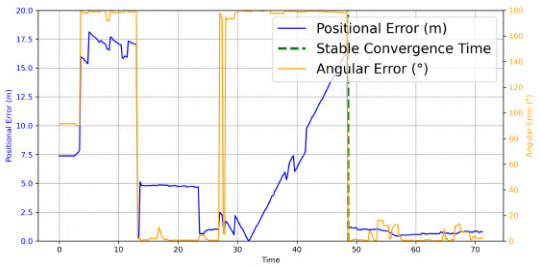


(b) PIMM best trial in A to C to B (errors).

Figure 4.14: PIMM best trial in A to C to B with no objects. (a) Trajectories. (b) Errors.



(a) PIMM worst trial in A to C to B with no objects. (trajectories).



(b) PIMM worst trial in A to C to B with no objects. (errors).

Figure 4.15: PIMM worst trial in A to C to B with no objects. (a) Trajectories. (b) Errors.

4.4.1 Performance summary

This trajectory had PIMM outperforming AMCL in convergence rate but AMCL outperforming in the post-convergence metrics. This highlights the strengths and weaknesses of both approaches, with PIMM doing well in the task of ambiguity collapse and fast convergence but lacking the precision of AMCL, and AMCL excelling in the precision but showing inconsistent behaviour in terms of convergence.

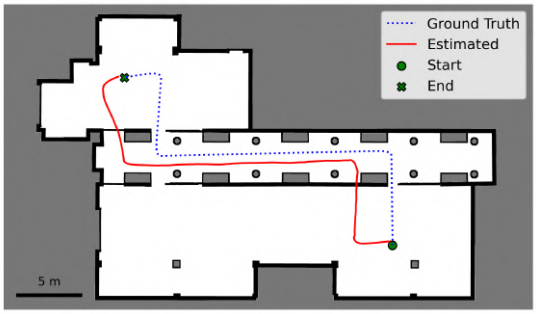
Table 4.4: A to C to B trajectory performance comparison across approaches

Type	Conv. Rate (%)	Conv. Time (s)	Pos. Error (m)	Ang. Error (°)
AMCL	70	12.54	0.17	1.46
PIMM	100	34.48	0.89	3.55

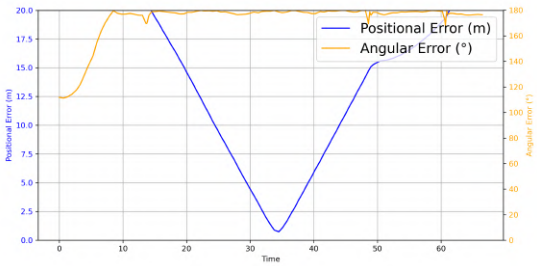
4.5 A to C to B with kidnapped robot

This test aims to evaluate the ability of both systems to converge to the right pose after the robot is suddenly displaced. To achieve this, the A to C to B trajectory is executed, allowing the system in question to converge to the right pose. After the trajectory is finished, the particle filter maintains its state but the trajectory is executed again.

AMCL



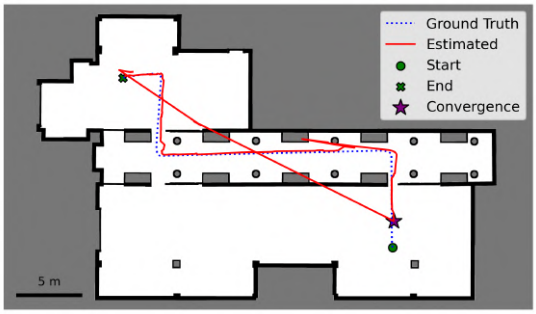
(a) AMCL example trial in A to C to B after robot kidnap (trajectories)



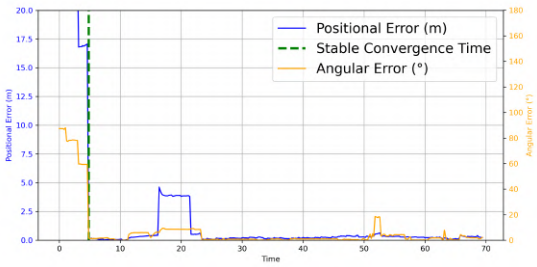
(b) AMCL example trial in A to C to B after robot kidnap (errors)

Figure 4.16: AMCL example trial in A to C to B after robot kidnap (a) Trajectories. (b) Errors.

PIMM

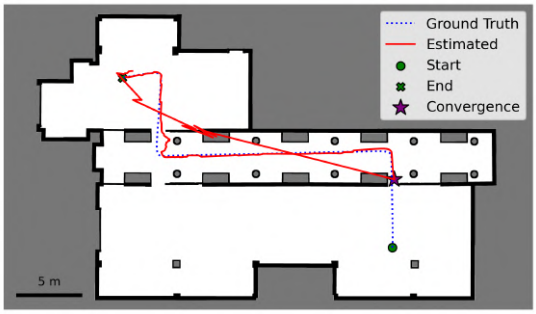


(a) PIMM best trial in A to C to B after robot kidnap (trajectories)

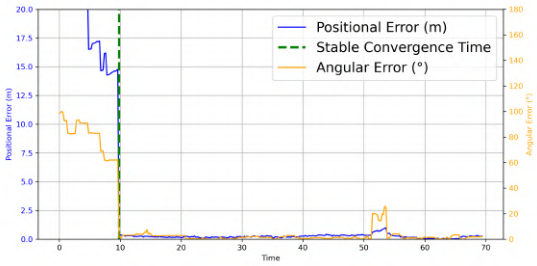


(b) PIMM best trial in A to C to B after robot kidnap (errors)

Figure 4.17: PIMM best trial in A to C to B after robot kidnap (a) Trajectories. (b) Errors.



(a) PIMM worst trial in A to C to B after robot kidnap (trajectories)



(b) PIMM worst trial in A to C to B after robot kidnap (errors)

Figure 4.18: PIMM worst trial in A to C to B after robot kidnap (a) Trajectories. (b) Errors.

4.5.1 Performance summary

AMCL was unable to recover the robot's pose after the kidnapping event, failing to converge in any of the trials. This highlights a major limitation of AMCL in scenarios involving severe localization loss. In contrast, the proposed PIMM approach consistently recovered the robot's pose, successfully converging in all runs. Moreover, PIMM achieved this recovery and with low post-convergence errors in both position and orientation. These results suggest that PIMM provides robust and reliable localization even in highly challenging conditions where traditional methods like AMCL fail.

This is explained by the fact that AMCL does not inject particles throughout the map, while PIMM does. Additionally, PIMM converges faster in this situation than in the normal A to C to B trajectory. This is due to the lack of particles in the left part of area A, which would introduce ambiguity.

Table 4.5: A to C to B trajectory performance comparison across approaches

Type	**Conv. Rate (%)**	**Conv. Time (s)**	**Pos. Error (m)**	**Ang. Error (°)**
AMCL	0	-	-	-
PIMM	100	7.33	0.33	2.16

4.6 Computational load

The tests were made on workstation with the following characteristics:

- **CPU:** AMD Ryzen 7 5700U with Radeon Graphics × 16
- **Memory:** 16 GB DDR4 RAM
- **OS:** Ubuntu 24.04 LTS (dual-boot with Windows 11)
- **Webots version:** R2025a

- **ROS 2 Distribution:** Jazzy Jalisco
- **Python Setup:**
  - Python 3.12.3
  - Numpy 1.26.4
  - Open3D 0.19.0
  - PyTorch 2.7.0
  - SciPy 1.11.4
  - SciKit-Learn 1.4.1
  - SciKit-Image 0.22.0
  - Faiss-cpu 1.11.0

To evaluate the computational efficiency of each localization approach, we measured the duration of key processing stages within both AMCL and PIMM during runtime. These timings provide insight into the overhead introduced by each method, which is especially relevant for real-time robotic applications. AMCL

For AMCL, the measurement update time was recorded as the primary indicator of computational load. This step is typically the most computationally intensive within the AMCL pipeline, as it involves evaluating sensor likelihoods across the particle set. Special attention was given to the effect of adaptive resampling, which dynamically adjusts the number of particles based on pose uncertainty. While this mechanism improves efficiency under normal conditions, its impact under kidnap-like scenarios—where the particle set can grow significantly—will be analyzed. Final results will provide average and peak measurement update durations across the A–C–B trajectory. PIMM

For PIMM, the computational load was broken down into several key components:

- **Segmentation Time:** Time required to classify each point as being part of a wall, ceiling, floor or clutter. It is the most significant component in the computational load, with an average of **563 ms** of time between the reception of the point cloud and the segmentation of every point.
- **Corner Detection Time:** Duration of extracting the corners, that are used for matching against the map. An average of **64 ms** between the reception of the segmented point cloud and the publishing of every detected corner.
- **Pose Inference Time:** Duration required to select or estimate the most likely pose based on the retrieved candidates. It was measured at an average of **1.2 ms**
- **Measurement Update Time:** Time spent matching the observed with the mapped corners and updating each particle's weight. It was measured at an average of **2.3 ms**.

For AMCL, the only measured component was the measurement update time, which starts at an average of **25.2 ms** in the beginning of operation but falls to an average of **5.8 ms** with the reduction in number of particles due to convergence.

These results show that by only processing corners for the measurement update phase, the particle filter can speed up its operation, as expected. Additionally, AMCL's mechanism of reducing the number of particles when converging also has a very significant impact on the computational time. One must also not ignore the significant cost of the segmentation and corner detection pipeline, which do not compensate for the gain in speed in the feature matching step.

Chapter 5

Conclusions and future work

This dissertation aimed, first and foremost, to develop a natural-landmark based localization system capable of resolving the global localization and kidnapped robot problems. With the use of a custom trained PointNet, global descriptor fingerprinting, a tailored corner detector and a custom made particle filter, PIMM proved sufficiently able to solve these problems in the simulated environment.

Regarding convergence time and consistency, the proposed system achieved better results than the widely used AMCL package, greatly influenced by the global descriptor matching pipeline, which enables a faster collapse of ambiguities.

In terms of computational load, the use of only corners for matching by the particle filter reduced the particle filter's computational time when comparing to AMCL. Despite this, the computation required to obtain such corners does not compensate for this gain in speed, so further improvements in the corner detection pipeline's efficiency would have to be made.

Finally, by using PointNet to detect clutter in the environment, the system shows robustness to unmapped and dynamic objects.

5.1 Future Work

While the results obtained are encouraging, several challenges remain unresolved. The following directions are proposed for future research and development:

Computational Efficiency As previously noted, the system's major bottleneck lies in the computational cost of semantic segmentation and corner extraction. Several strategies could be explored to mitigate this issue:

- **Partial Segmentation:** Instead of classifying every point, limit segmentation to a representative subset of the point cloud (e.g., via uniform downsampling or selecting points from a single LiDAR layer). This could significantly reduce processing time with minimal impact on global descriptor quality.

- **2D Corner Detection:** Use a projected or filtered 2D subset of the 3D scan—e.g., a planar slice of wall-labeled points—for faster corner extraction. This would retain robustness to clutter while significantly speeding up processing.
- **Adaptive Particle Count:** Inspired by AMCL, incorporate a dynamic particle count that adjusts based on localization confidence. This would reduce computational load during high-confidence tracking phases while retaining robustness during uncertain or ambiguous situations.

Real-World Transferability Although the system performed well in simulation, real-world deployment presents new challenges:

- **Real Datasets :** The segmentation model was trained solely on simulated data. Its performance in real-world conditions—subject to different sensor noise, lighting, and structural variations—is untested. Collecting and annotating real-world datasets or applying domain adaptation techniques could improve robustness.
- **Sensor Integration and Calibration:** Testing on real hardware (e.g., a mobile robot equipped with a 3D LiDAR) would expose integration challenges such as sensor calibration, latency, and real-time performance constraints.
- **Benchmarking Against Real Data:** Controlled experiments in structured environments (e.g., offices, warehouses) should be conducted to evaluate convergence reliability, runtime performance, and sensitivity to clutter and layout changes.

Feature Diversity The current system relies exclusively on wall-ceiling corners as natural landmarks. While these proved effective in simulation, broader feature diversity could increase robustness and reduce ambiguity in highly symmetric or minimally featured environments. Future work could explore additional semantic features detect and use other architectural elements such as columns, windows and ceiling lights. These features may provide valuable cues in environments where corners are sparse or ambiguous.

Scalability and Long-Term Deployment Finally, ensuring that the system remains scalable and robust over time is essential for real-world applications. An additional pipeline to enable the system to adapt to changes in the environment by updating its descriptor map over time, either through online learning or periodic re-training, could be implemented.

By addressing these challenges, the system could be brought closer to real-world usability, making it a valuable tool for autonomous indoor robots operating in dynamic, cluttered, and unstructured environments, without the use of artificial landmarks.

Bibliography

[1] Brian D. Lowe et al. “Case Studies of Robots and Automation as Health/Safety Interventions in Small Manufacturing Enterprises”. In: *Human factors and ergonomics in manufacturing* 33.1 (Aug. 2022), pp. 69–103.

[2] Ling Li and Perry Singleton. “The Effect of Industrial Robots on Workplace Safety”. In: *Center for Policy Research* (Feb. 1, 2021).

[3] Roland Siegwart, Illah R. Nourbakhsh, and Davide Scaramuzza. *Introduction to Autonomous Mobile Robots*. 2nd. The MIT Press, 2011.

[4] Prabin Kumar Panigrahi and Sukant Kishoro Bisoy. “Localization strategies for autonomous mobile robots: A review”. In: *Journal of King Saud University - Computer and Information Sciences* 34.8, Part B (2022), pp. 6019–6039.

[5] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[6] Sherif AS Mohamed et al. “A survey on odometry for autonomous navigation systems”. In: *IEEE access* 7 (2019), pp. 97466–97486.

[7] Beakcheol Jang, Hyunjung Kim, and Jong wook Kim. “Survey of Landmark-based Indoor Positioning Technologies”. In: *Information Fusion* 89 (Jan. 1, 2023), pp. 166–188.

[8] Faheem Zafari, Athanasios Gkelias, and Kin K. Leung. “A Survey of Indoor Localization Systems and Technologies”. In: *IEEE Communications Surveys & Tutorials* 21.3 (2019). Conference Name: IEEE Communications Surveys & Tutorials, pp. 2568–2599.

[9] Michail Kalaitzakis et al. “Fiducial Markers for Pose Estimation”. In: *Journal of Intelligent & Robotic Systems* 101.4 (Mar. 26, 2021), p. 71.

[10] Rafael Marques Claro, Diogo Brandão Silva, and Andry Maykol Pinto. “ArTuga: A novel multimodal fiducial marker for aerial robotics”. In: *Robotics and Autonomous Systems* 163 (2023), p. 104398.

[11] N. Ergin Özkucur and H. Levent Akın. “Localization with Non-unique Landmark Observations”. In: *RoboCup 2010: Robot Soccer World Cup XIV*. Ed. by Javier Ruiz-del-Solar, Eric Chown, and Paul G. Plöger. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 72–81.

[12] Xian-Feng Han et al. “3D point cloud descriptors: state-of-the-art”. In: *Artificial Intelligence Review* 56.10 (Oct. 1, 2023), pp. 12033–12083.

[13] Paul Scovanner, Saad Ali, and Mubarak Shah. “A 3-dimensional sift descriptor and its application to action recognition”. In: MM ’07. Augsburg, Germany: Association for Computing Machinery, 2007, pp. 357–360.

[14] David G. Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *Int. J. Comput. Vision* 60.2 (2004), pp. 99–110.

[15] Yu Zhong. “Intrinsic shape signatures: A shape descriptor for 3D object recognition”. In: *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*. 2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops. Sept. 2009, pp. 689–696.

[16] Ivan Sipiran and Benjamin Bustos. “Harris 3D: a robust extension of the Harris operator for interest point detection on 3D meshes”. In: *The Visual Computer* 27.11 (Nov. 1, 2011), pp. 963–976.

[17] Radu Bogdan Rusu et al. “Learning informative point classes for the acquisition of object model maps”. In: *2008 10th International Conference on Control, Automation, Robotics and Vision*. 2008, pp. 643–650.

[18] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast Point Feature Histograms (FPFH) for 3D registration”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009 IEEE International Conference on Robotics and Automation. ISSN: 1050-4729. May 2009, pp. 3212–3217.

[19] Samuele Salti, Federico Tombari, and Luigi Di Stefano. “SHOT: Unique signatures of histograms for surface and texture description”. In: *Computer Vision and Image Understanding* 125 (Aug. 1, 2014), pp. 251–264.

[20] P.J. Besl and Neil D. McKay. “A method for registration of 3-D shapes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.2 (Feb. 1992). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 239–256.

[21] Xueyao Jiang et al. “Review on improved algorithms based on ICP algorithm”. In: *2020 International Conference on Computer Engineering and Intelligent Control (ICCEIC)*. 2020 International Conference on Computer Engineering and Intelligent Control (ICCEIC). Nov. 2020, pp. 185–189.

[22] P. Biber and W. 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)*. Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453). Vol. 3. Oct. 2003, 2743–2748 vol.3.

[23] R. Schnabel, R. Wahl, and R. Klein. “Efficient RANSAC for Point-Cloud Shape Detection”. In: *Computer Graphics Forum* 26.2 (2007), pp. 214–226.

[24] Tomofumi Fujiwara, Tetsushi Kamegawa, and Akio Gofuku. "Evaluation of plane detection with ransac according to density of 3D point clouds". In: *arXiv preprint arXiv:1312.5033* (2013).

[25] Dena Bazazian, Josep R. Casas, and Javier Ruiz-Hidalgo. "Fast and Robust Edge Extraction in Unorganized Point Clouds". In: *2015 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. 2015, pp. 1–8.

[26] R. Qi Charles et al. "PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). ISSN: 1063-6919. July 2017, pp. 77–85.

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Neural Information Processing Systems* 25 (Jan. 2012).

[28] Charles Ruizhongtai Qi et al. "PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.

[29] Connor Shorten and Taghi M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (July 6, 2019), p. 60.

[30] A Geiger et al. "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research* 32.11 (Sept. 1, 2013), pp. 1231–1237.

[31] Giseop Kim et al. "MulRan: Multimodal Range Dataset for Urban Place Recognition". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020 IEEE International Conference on Robotics and Automation (ICRA). May 2020, pp. 6246–6253.

[32] Weixin Lu et al. *L 3 -Net: Towards Learning based LiDAR Localization for Autonomous Driving*. Sept. 27, 2019.

[33] Nicholas Carlevaris-Bianco, Arash K Ushani, and Ryan M Eustice. "University of Michigan North Campus long-term vision and lidar dataset". In: *The International Journal of Robotics Research* 35.9 (Aug. 1, 2016), pp. 1023–1035.

[34] Will Maddern et al. "1 year, 1000 km: The Oxford RobotCar dataset". In: *The International Journal of Robotics Research* 36.1 (Jan. 1, 2017), pp. 3–15.

[35] Wei Wang et al. "PointLoc: Deep Pose Regressor for LiDAR Point Cloud Localization". In: *IEEE Sensors Journal* 22.1 (Jan. 2022), pp. 959–968.

[36] Huan Yin et al. "A Survey on Global LiDAR Localization: Challenges, Advances and Open Problems". In: *International Journal of Computer Vision* 132.8 (Aug. 1, 2024), pp. 3139–3171.

[37] Ali Yassin et al. "Recent Advances in Indoor Localization: A Survey on Theoretical Approaches and Applications". In: *IEEE Communications Surveys & Tutorials* 19.2 (2017). Conference Name: IEEE Communications Surveys & Tutorials, pp. 1327–1346.

[38] Changhao Chen et al. *A Survey on Deep Learning for Localization and Mapping: Towards the Age of Spatial Machine Intelligence*. June 29, 2020.

[39] Huan Yin et al. "LocNet: Global Localization in 3D Point Clouds for Mobile Vehicles". In: *2018 IEEE Intelligent Vehicles Symposium (IV)*. 2018 IEEE Intelligent Vehicles Symposium (IV). ISSN: 1931-0587. June 2018, pp. 728–733.

[40] Davide Chicco. "Siamese Neural Networks: An Overview". In: *Artificial Neural Networks*. Ed. by Hugh Cartwright. New York, NY: Springer US, 2021, pp. 73–94.

[41] Mikaela Angelina Uy and Gim Hee Lee. "PointNetVLAD: Deep Point Cloud Based Retrieval for Large-Scale Place Recognition". In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018, pp. 4470–4479.

[42] Hervé Jégou et al. "Aggregating local descriptors into a compact image representation". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. ISSN: 1063-6919. June 2010, pp. 3304–3311.

[43] Wenxiao Zhang and Chunxia Xiao. "PCAN: 3D Attention Map Learning Using Contextual Information for Point Cloud Based Retrieval". In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019, pp. 12436–12445.

[44] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017.