GRADO EN INGENIERÍA MATEMÁTICA E INTELIGENCIA ARTIFICIAL

TRABAJO FIN DE GRADO

Visual Topological SLAM using Deep Learning Techniques

Autor: Manuel Rodríguez Villegas

Directores: Jaime Boal Martín-Larrauri y Jesús Tordesillas Torres

Madrid

Junio de 2025

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título **Visual Topological SLAM using Deep Learning Techniques** en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2024/25 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente, y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Manuel Rodríguez Villegas **Fecha:** 9 de junio de 2025

Autorizada la entrega del proyecto

LOS DIRECTORES DEL PROYECTO

Fdo.: Jaime Boal Martín-Larrauri **Fecha:** _____

Fdo.: Jesús Tordesillas Torres **Fecha:** _____

To my family, for their unconditional support.

To my friends, for always being there, even when I was far away.

To the amazing people I met at Cornell.

And to Jaime and Jesús, my supervisors, for their help and flexibility during the year.

Contents

List of Figures

List of Tables

***Abstract*—Topological maps represent an environment as a graph, with nodes corresponding to points of interest and edges representing traversable paths. They offer efficient integration with natural language modules and are computationally lighter than metric maps, which, while more precise, demand greater processing power. This work presents a system for constructing a topological map of an indoor environment using odometry measurements and camera images. Visual inputs are processed through a deep learning model to extract feature representations, which are then used for similarity comparisons during node extraction. Odometry data is used to refine node positions, enhancing robustness. A loop closure detection and rewiring mechanism is proposed to update loop edges when a closure is detected, improving the connectivity between affected nodes. Two different trajectories are aligned and their nodes fused to increase robustness and correct detection errors. Finally, semantic information is incorporated by first detecting the objects in each node's image, then listing them in a sentence and lastly obtaining the embedding of the sentence with a language model. This process enables voice-controlled navigation.**

I. Introduction

This section covers the contextual information necessary for the correct understanding of the project, its main objectives, and the structure of the project.

A. Context and Motivation

Comillas ICAI participates in the UNIJES SocialTech Challenge, a robotics competition with social purposes. The goal for the 2024/2025 edition is to build an autonomous wheelchair capable of navigating efficiently within an office environment. The competition is structured around three challenges: first, the wheelchair must visit a sequence of specified points of interest in a designated order. The second challenge introduces static obstacles, and the third challenge adds dynamic obstacles to further test navigation capabilities. While the primary focus of the 2024/2025 competition is on office environments, the system proposed in this work is adaptable to other indoor spaces such as museums, schools, or supermarkets.

The main navigation system of the wheelchair uses a LiDAR[1]-based metric map, which stores precise information about the distances between obstacles. While this provides an accurate and detailed representation of the environment, it comes with a high computational cost. To complement the metric map, this work introduces an algorithm that builds a topological map: a graph-based structure that stores only key locations and the connections between them. Topological maps are more efficient, mimicking how humans perceive their surroundings: by focusing on relative positions rather than exact distances, enabling navigation without collisions. They are also less sensitive to localization errors, as they do not depend on precise coordinates, and can be easily combined with semantic information to support voice-controlled navigation.

The main motivation behind this project lies in the advantages of combining a metric map with a topological one. Furthermore, the introduction of deep learning features provides a more robust environment representation for both navigation and voice-control.

The use of a topological SLAM algorithm allows the autonomous wheelchair to create a more efficient environment representation for global navigation. When combined with a metric map, the system can achieve accurate navigation based on natural language processing (NLP) commands, a feature that would be much harder to implement without a graph-based structure.

B. Objectives

The purpose of this project is to build a topological map that can be used for efficient navigation. To achieve this goal, the process is divided into three subobjectives: feature extraction, node extraction, and loop closure with map alignment. On top of that, the integration of semantic information will be set as an extra objective once the rest of the system is completely functional.

The pipeline begins when the camera captures an image of the environment. The first subojective is to train a model capable of providing a consistent representation of its environment that can used in the following stages. This feature vector has to be both low-dimensional compared to the original images and also robust against diverse lightning conditions.

The next step is node extraction, which involves selecting the most informative locations to serve as nodes in the topological map. Thus, the second subojective involves developing a system that accurately detects when a new node needs to be added to the graph. This task will be successfully accomplished if the generated nodes are located in meaningful places (rooms, corridors, doorways) and the graph structure is coherent and non-redundant.

Finally, the system must determine whether a newly captured image corresponds to a previously visited location or a new one. This process, known as loop closure detection, is carried out by comparing the current feature vector with those of existing nodes. Therefore, the final subojective will be to optimize the generated graph to ensure consistent location of nodes and avoid missing or over-represented spaces. This process involves both a functional loop detection algorithm and a graph alignment process that combines information from multiple maps to corrrect detection ald location errors.

[1]LiDAR stands for Light Detection and Ranging. It is a remote sensing method that uses laser light to measure distances to objects.

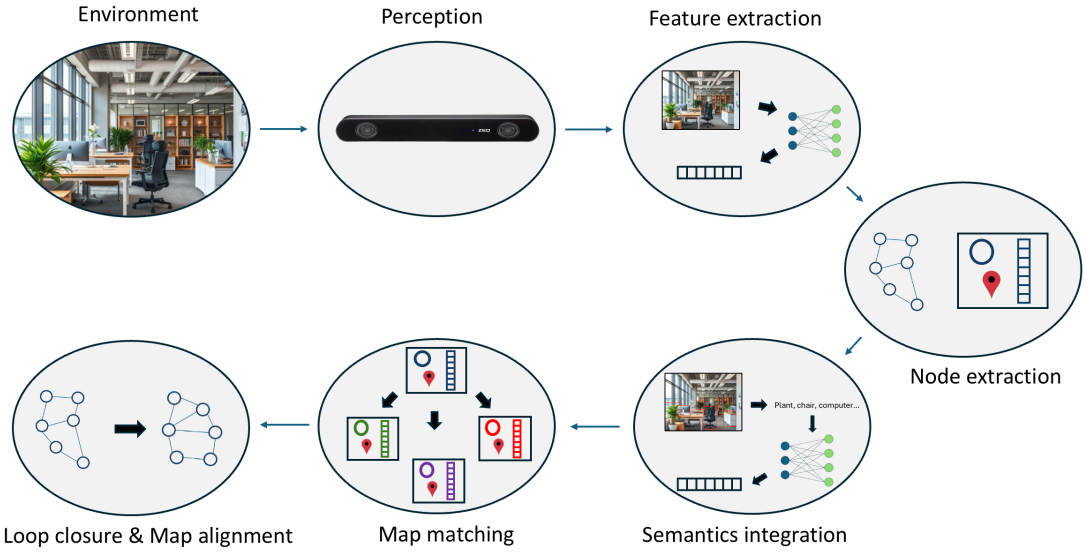


Figure 1. Summary of the proposed algorithm.

After completing the three main subobjectives, semantic information can be added to the graph's nodes as an extra goal. This enriched representation enables seamless integration of a voice-controlled navigation system.

C. Project Structure

The proposed algorithm consists of the following steps. It begins by capturing an image of the environment, which is processed by a deep learning model (CNN or AutoEncoder) to produce a feature vector representing its visual content. An object detection model is then applied to identify all objects present in the image associated with each node. These objects are combined into a descriptive sentence, which is passed through a language model to generate an embedding that captures the semantic information of the node. This enriched representation enables the integration of voice-controlled navigation.

Although the initial plan was to use odometry data for localization, the measurements from the selected dataset exhibited significant discrepancies compared to the ground truth data. As a result, the localization module was based on ground truth information rather than odometry estimates. This information is used to estimate the position of each image, which, combined with the previously extracted vectors, defines a node in the graph. The new node is then compared to existing ones, based on both pose and visual feature similarity, to determine whether it represents a new location or an update to an existing node. If the system detects a previously visited location (i.e., a loop closure), odometry data is used to adjust the positions of the affected nodes, ensuring consistent localization. Additionally, a rewiring mechanism refines the connections between nodes within the loop to improve the graph's structure.

Once two different trajectories have been completed, the map alignment module combines the resulting graphs to enhance robustness and correct errors in node detection and placement. A visual summary of this process is shown in Figure 1.

The remainder of this report is structured as follows: Section II presents a review of related work. Section III explains the techniques and algorithms used throughout the project. Section IV describes the experiments conducted to evaluate system performance. Section V analyzes the results obtained. Finally, Section VI outlines the project's conclusions and suggests potential directions for future work.

II. Related Work

This section provides a review of the literature on visual topological Simultaneous Localization and Mapping (SLAM). It begins by covering the different options for perception systems, which are the sensors used to capture environmental information. Feature extraction mechanisms are analyzed afterwards, followed by node clustering and map alignment techniques. The analysis finishes with a revision of loop closure detection and map alignment algorithms. Finally, the section covers different techniques for integrating semantic information into topological maps.

A. Perception

In the perception phase, robots gather information about their environment using various sensors with which

Table I. Summary of sensors used in various sources.

Source	Camera			LiDAR	Odometry	IMU
	RGB	RGB-D	Thermal			
Sousa and Bassani [37]	✓					
Blochliger et al. [12]	✓			✓		
Zhu et al. [40]		✓			✓	
Hughes et al. [35]	✓				✓	✓
Chaplot et al. [25]	✓				✓	
Agha et al. [28]	✓		✓	✓	✓	✓
Bavle et al. [33]				✓	✓	
Luo and Chiou [14]	✓			✓	✓	
Yoshida et al. [39]	✓				✓	
Wang et al. [27]	✓					
Hou et al. [4]	✓	✓				
Piergiovanni et al. [30]	✓			✓		
Vora et al. [26]	✓			✓		
Qi et al. [15]		✓				

they are equipped. There are two primary options when selecting sensors for a robot performing SLAM: an RGB camera, which captures 2D images with one or three color channels, and LiDAR, which generates a point cloud estimating the distance to each point within its field of view. Both options can be used separately or combined to leverage the advantages of both worlds. Incorporating odometry measurements or information from an Inertial Measurement Unit (IMU) can help represent the environment in a more precise way in both approaches.

Some systems rely solely on camera information for the visual SLAM algorithm. Sousa and Bassani [37] use a CNN architecture to extract features from 2D images that are then converted to features of a topological map. Yoshida et al. [39] present a system that receives an RGB image as input of an instance segmentation algorithm which is then transformed into a graph for further processing. A similar system in presented by Wang et al. [27], where the authors perform instance segmentation of RGB images and then filter out dynamic objects. A two-camera system is proposed by Hou et al. [4], where they use the right- and left-hand sides of the same camera with different lightning conditions to train a robust model. Some techniques make use of odometry sensors or IMU information to make a more accurate representation of the environment, as shown in the Hydra architecture by Hughes et al. [35]. Finally, the use of noisy odometry sensors along with RGB panoramic images can help autonomous robots better localize themselves in indoor environments, as presented by Chaplot et al. [25].

On the other hand, there are techniques that make use of point cloud representations obtained with LiDARs. One example is the architecture proposed by Blochliger et al. [12], where they generate a voxel map with 3D landmarks and posterior processing. Odometry information is also useful when combined with point clouds, as shown by Bavle et al. [33]. The authors use LiDAR information to recognize places and localize the robot, alongside the odometry measurements.

A third option is to use both an RGB camera and a LiDAR unit to obtain environmental information. Some techniques process input data from each device separately, while others combine the data into integrated features that are then fed into subsequent deep learning architectures. The first group includes systems like NeBula, by Agha et al. [28], which participated in the DARPA Subterranean Challenge. They combine LiDAR an IMU data for accurate localization with cameras for specially complex environments. In the work by Luo and Chiou [14], the authors construct a metric map using LiDAR data and a topological map using RGB images, incorporating odometry measurements into both to achieve a more accurate representation. The second group includes architectures that leverage the intrinsic and extrinsic parameters of the sensors to ensure spatial consistency between camera and LiDAR data. An example is the system developed by Waymo, as presented by Piergiovanni et al. [30], which employs dynamic connection learning to capture relationships between LiDAR and RGB features, using attention mechanisms to focus on the most relevant features. Vora et al. [26] introduce an architecture that "paints" the LiDAR-generated point cloud with semantic information obtained from a segmentation algorithm applied to RGB images.

This semantic information is appended to the point cloud, which can then be processed using a conventional LiDAR algorithm. A primary drawback of these techniques is their increased computational cost, due to the use of 3D CNNs.

A similar option comes with the use of RGB-D cameras, whose images are known to have 2.5 dimensions (the 2D image plus the depth dimension). Zhu et al. [40] use an RGB-D camera to integrate appearance, geometric and semantic features with a cross-attention decoder so the model remains robust when one of the metrics incurs in error. The architecture presented by Qi et al. [15] performs instance segmentation with 2D images and then extend them to perform 3D image segmentation on reduced parts of the image. Table I provides a summary of the sensors used in each of the discussed works.

B. Feature Extraction

The feature extraction process transforms sensor data into numerical representations (features) suitable for use in deep learning models. Common systems include a CNN to extract image features, which are then stored in a graph structure for subsequent processing.

Sousa and Bassani [37] employ a multi-step process where visual features are extracted from images using GoogleNet. This vector represents visual characteristics and is used both for object classification and to establish spatial relationships between map nodes. The method introduced by Hou et al. [4] makes use of the image features from a layer of a CNN as image descriptors for the loop closure detection. Zhu et al. [40] leverage an RGB-D camera to combine geometric, semantic and appearance features using cross-attention to obtain a single feature for hierarchical semantic mapping. In the work shown by Wang et al. [27], the authors propose a system that performs semantic segmentation on the RGB image and uses it to extract distinctive landmarks and to create a semantic graph. The landmarks are fed into a CNN to calculate the similarity between different frames with the help of the semantic graph.

Other techniques, such as the one presented by Hughes et al. [35], employ RGB images to build a hierarchical scene graph. In this case, the authors use three different processes for different level features and computations. A similar approach is presented by Bavle et al. [33], but in this case the perception system is a LiDAR and the hierarchy is defined by fixed categories (floors, rooms...). The system used by Blochliger et al. [12] receives a sparse visual SLAM map with triangulated landmarks and builds a topological map in which the vertices are convex voxel clusters and the edges are their adjacent areas.

The architecture proposed by Yoshida et al. [39] processes the input RGB images through a semantic segmentation algorithm and converts the output to a graph by using adjacent bounding boxes and threshold-based distances. This graph is then fed to a graph convolutional network (GCN) for knowledge transfer. In the system developed by Chaplot et al. [25], four functions are combined to update the topological graph and to plan both global and local trajectories. A hybrid approach is presented by Luo and Chiou [14], where they build a graph in which nodes can have three different levels of abstraction: low abstraction (metric map) and high abstraction (topological map). This hybrid representation allows efficient semantic-based navigation.

As an alternative to CNNs, autoencoders have also been used as feature extractors in cases in which a lower dimensionality provides remarkable advantages. CodeSLAM [13] was a pioneering method that integrated an autoencoder into SLAM to learn a compact representation of dense geometry. Instead of storing full-resolution depth maps, they trained a variational autoencoder to encode a depth map into a low-dimensional code. In addition, conditioning the depth map on the image enables the code to focus only on those aspects of the geometry that cannot be directly predicted from the image.

Luo et al. [29] addressed the loop closure detection problem with a Stacked Assorted AutoEncoder (SAAE). Traditional bag-of-words (BoW) place recognition struggles under appearance change. SAAE instead learns a robust image descriptor by combining multiple autoencoders. Specifically, it stacks a denoising autoencoder (to make features robust to noise and appearance change), a convolutional autoencoder (to preserve spatial structure in the feature), and a sparse autoencoder (to enforce compactness and reduce dimensionality). The resulting loop closure feature is more robust than those produced by single-type autoencoders or BoW models.

Song et al. [42] propose a loop closure detection method based on a variational autoencoder (VAE) with an attention mechanism. The VAE network is trained to extract a low-dimensional vector representation of each input image, effectively serving as a learned global feature instead of handcrafted features. This autoencoder-derived descriptor, along with an attention module that improves the encoding of important regions, is used to match images for loop closure. This study reveals the viability of replacing CNN and hand-crafted feature extractors with low-dimensional autoencoder representations.

The Transformer architecture introduced by Vaswani et al. [11] has also been used in a variety of visual SLAM projects, with a special focus on dynamic environments. Chen et al. [41] propose VTD-SLAM, a visual SLAM system that replaces CNN-based segmentation with an improved Vision Transformer (ViT) backbone to handle dynamic scenes. Using the Transformer's global self-

attention (instead of only local convolutions), the system better captures long-range dependencies for accurate segmentation, achieving a 17% reduction in trajectory error compared to a CNN-based method.

In the work by Wang et al. [38], the authors propose a hybrid approach that combines CNNs with Transformers. The CNN component is utilized to extract initial feature maps from input images, capturing local features effectively. These feature maps are then enhanced by the Transformer module, which models contextual relationships within the image. This combination leverages the strengths of both architectures: the CNN's proficiency in local feature extraction and the Transformer's capability to understand global context through self-attention mechanisms.

Sun et al. [32] developed LoFTR, a detector-free feature matching network that uses transformers to replace the traditional keypoint detector and descriptor pipeline. LoFTR establishes dense pixel-wise correspondences between image pairs via self- and cross-attention layers, producing high-quality matches even in low-texture or motion-blurred areas. As an improvement over LoFTR in SLAM, Qu et al. [43] present MSpGLoFTR, which introduces multi-scale attention mechanisms to better handle scale variations.

C. Node Clustering and Map Matching

Once the features from the input source are extracted, they are typically organized into a graph structure composed of nodes. Each node represents a location of interest in the real world, while edges indicate the traversable space between nodes. Node clustering is a part of this process, where similar nodes are grouped together to simplify the system architecture and prevent the need for a unique node for every analyzed feature. In map matching, the system seeks to establish associations between nodes in consecutive frames or between the current frame and an existing map.

The algorithm presented by Blochliger et al. [12] clusters the recognized space starting with a unique point in the center of each free space zone (stored with its convex hull) and grows it iteratively, ensuring that no obstacles are added to the cluster. In the system presented by Sousa and Bassani [37], each node is represented by three vectors (position, visual features and average distance features vectors). The visual features vector of the nodes that enter the system gets compared to the existing ones. If the distance to the closest features vector is lower than a certain threshold, its visual features are consolidated and the graph is updated. However, if the distance is higher than the threshold, a new node is created with its new position and a moving average between the new visual features and the last vector's visual features.

Chaplot et al. [25] propose that each node keeps the panoramic image taken in its position and the relative pose between them. The Graph Update function then compares previous nodes with the new one to determine whether to create another node. A similar approach is considered by Luo and Chiou [14], where each node's features come directly from the CNN. Clustering is carried out between nodes based on spatial proximity or semantic similarity of these features.

Bavle et al. [33] present a system that constructs a hierarchical scene graph from low-level vector information, which is clustered into free space clusters. In the following layer, these clusters are connected, integrating both metric and semantic information. New connections are then established to higher layers that represent rooms and floor levels. In the system shown by Yoshida et al. [39], a GCN is used to extract a feature vector from the graph. Node clustering techniques are applied to the node embeddings produced by the GCN to decide whether each feature can be classified into an existing cluster, or it may form a new one.

In very complex environments, like the one shown by Agha et al. [28], it is preferable to store precise geometric information rather than a topological structure. In this case, semantic information is added to the map to increase its navigation capabilities. Moreover, multiple robots share their maps to build a stronger global localization system.

D. Loop Closure Detection and Map Alignment

The final part of the process involves detecting already visited zones and adjusting the generated map to ensure consistency between different localizations.

The system presented by Bavle et al. [33] performs loop closure detection with hard constraints at very low level features, representing neighboring keyframe poses, while softer constraints are applied in higher layers of the graph.

Luo and Chiou [14] propose that CNN and geometric information are both used to determine whether the robot has returned to an already visited location. Once loop closure is detected, the robot performs map alignment to correct the trajectory. The work by Wang et al. [27] introduces an advanced loop closure detection method that combines information from two different systems. The first system compares geometric similarity between graphs derived from a semantic segmentation algorithm. The second system integrates information from CNN features and Hu moments to assess appearance similarity. Each segmented image is compared to all images in the dataset that share at least one common label.

Hou et al. [4] provide an extensive comparison between hand-crafted features and CNN features (output of Con-

Table II. Comparison of Visual Topological SLAM using Deep Learning Techniques.

Source	Approach	Input Data	Key Features	Use Case
Sousa and Bassani [37]	CNN	RGB image	Feature consolidation	Indoor robots
Blochliger et al. [12]	3D voxels generation	3D landmarks (LiDAR)	Efficient path planning	Indoor robots
Zhu et al. [40]	CNN & decoder	RGB-D image	Feature representation	Indoor robots
Sünderhauf et al. [5]	CNN	RGB image	Feature extraction	Varies
Hughes et al. [35]	CNN & 3D mesh	RGB image	3D scene graphs	Indoor robots
Chaplot et al. [25]	CNN	RGB image	Image-goal navigation	Indoor robots
Agha et al. [28]	LiDAR-based	LiDAR landmarks & IMU	Autonomy in challenging environments	All kinds of robots
Bavle et al. [33]	LiDAR-based	LiDAR landmarks	3D scene graphs	Indoor robots
Luo and Chiou [14]	CNN & metric map	RGB image & LiDAR landmarks	Semantic mapping	Service robots
Yoshida et al. [39]	CNN & GCN	RGB image	Semantic localization	Indoor robots
Wang et al. [27]	CNN	RGB image	Loop closure detection	Outdoor & indoor robots
Hou et al. [4]	CNN	RGB image	Loop closure detection	Outdoor & indoor robots
Piergiovanni et al. [30]	3D CNN	RGB video & LiDAR landmarks	3D object detection	Autonomous vehicles
Vora et al. [26]	CNN & LiDAR	RGB image & LiDAR landmarks	Input feature fusion	Outdoor robots
Qi et al. [15]	CNN	RGB-D image	3D object detection	Outdoor & indoor robots
Bloesch et al. [13]	AutoEncoder	RGB and grayscale images	Dimensionality reduction	Indoor robots
Luo et al. [29]	AutoEncoder	RGB image	Loop closure detection	Outdoor robots
Song et al. [42]	AutoEncoder + Attention	RGB image	Loop closure detection	Outdoor robots
Chen et al. [41]	Transformer	RGB image	Image segmentation	Indoor robots
Wang et al. [38]	Transformer + CNN	RGB image	Place recognition	Outdoor robots
Sun et al. [32]	Transformer	RGB and grayscale images	Feature matching	Indoor robots
Qu et al. [43]	Transformer	RGB image	Feature matching	Outdoor robots

volutional or Pooling layer) as image descriptors for loop closure detection. They reach the conclusion that CNN features achieve the same performance with good lighting condition but outperform hand-crafted features with worse conditions.

Sünderhauf et al. [5] investigate the performance of CNN features and propose two techniques to enhance the speed of image comparisons. The first technique involves approximating the cosine distance, while the second technique clusters features into semantic groups to facilitate faster comparisons. The algorithm proposed by Hughes et al. [35] constructs a hierarchical set of descriptors that represent the surroundings of each node. When conducting loop closure detection, the algorithm traverses this hierarchy from places to objects to appearance descriptors. If the similarity of the place descriptors is higher than a certain threshold, the algorithm proceeds to compare the object descriptors, and so forth.

The graph-based structure present in many of these techniques can be optimized to ensure the global consistency of graph nodes and edges, thereby mitigating potential errors in pose estimates and the inherent noise in sensor measurements. One of the most popular methods for addressing this issue is the Levenberg-Marquardt algorithm, which is employed to solve non-linear least squares problems. Table II shows a brief description of the techniques used in the mentioned literature.

E. Semantics Integration

One of the main advantages of topological maps is their compatibility with natural language navigation. Semantic information provides insights into the objects, features or places that appear in a certain image. By integrating this knowledge into the graph, nodes become richer representations of the environment that unlock a new range of possibilities. When working with a semantically-informed topological map, the user can operate the system with natural language commands, increasing both the accessibility and the ease of use. Semantic information

can also be used to enhance the system's robustness against dynamic objects or changes in lighting conditions.

During the last decade, the focus on finding optimal algorithms to extract semantic information has shifted from traditional vision algorithms to deep learning techniques that leverage CNNs. These advanced algorithms meet both main requirements for visual SLAM: high semantic extraction accuracy and real-time performance. Three specific techniques may be used for this purpose: object detection, semantic segmentation and instance segmentation.

Object detection consists on identifying distinct objects in an image, which can help the system build a better understanding of the environment. One of the most common one-stage object detectors is YOLO [8], which has been improving over the years [10] [24] and provides a consistent enough performance for object detection in SLAM. It has been diversely employed in tasks like reconstructing the 3D shape of objects from 2D object detections [21] or reducing position error in autonomous navigation systems [22]. SSD [7] is another commonly used one-stage detector that optimally balances speed and accuracy. It has been used in several works, like the system designed by Doherty et al. [20], in which they increase robustness against perceptual aliasing and odometry error, or the work presented by Zhong et al. [19], improving the performance of a SLAM system in dynamic environments.

Semantic segmentation is used to classify different parts of an image but cannot differantiate between instances of the same class. Murali et al. [9] employ this technique to combine semantic information with visual features, improving accuracy on GPS-denied navigation. Yu et al. [18] present a novel system in which they integrate semantic segmentation with local mapping and loop closing to improve dynamical environments understanding.

Finally, the most powerful image understanding technique is instance segmentation, which is an improvement over object detection by providing pixel-wise detection. However, its real-time performance is not comparable to that of an object detection system. The most used technique is Mask-RCNN, which has been used in works like the one by Runz et al. [17], where they build a RGB-D SLAM system capable of labeling objects even when they move independently from the camera.

III. Methodology

This section provides a detailed explanation of the algorithms and techniques employed during the development of the system. It begins with a description of the chosen dataset, followed by an overview of the entire process, and concludes with an emphasis on two main components: (1) the training and validation phases of the different models used for feature extraction, and (2) the simulation of a real-life scenario using ROS2 [36], a robotics framework that enables real-time communication between different nodes.

A. COLD Dataset

The COLD dataset [1] was selected to replicate the office-like environment that the wheelchair will encounter during the competition. This dataset includes data from five different trajectories recorded by robots operating in three laboratories (Freiburg, Saarbrücken, and Ljubljana). Freiburg and Saarbrücken present standard and extended trajectories, which include room types not existing in the standard ones. The dataset provides images captured by the robots at various locations, along with odometry measurements of the robots' trajectories. Each trajectory includes a variety of weather and lighting conditions (sunny, cloudy, or night), resulting in a total of 77 recorded sequences across the three labs. Although the dataset also includes LiDAR measurements, these were not utilized in this project.

Due to significant inconsistencies between the odometry data and the ground truth data (derived from image file names), the odometry information could not be used in the localization module. Consequently, the system was developed using ground truth data for localization instead of relying on odometry estimates. While this eliminates the issue of odometry drift, one of the major challenges in robotic navigation, it also means the simulation scenario does not fully demonstrate one of the key advantages of topological maps: their robustness to odometry drift. Unlike metric maps, topological maps are less dependent on precise positional accuracy, making them especially effective in scenarios where odometry is unreliable. Odometry data will be incorporated once the system is deployed on the actual wheelchair, using sensor information from the platform's wheels.

Another issue with the COLD dataset is the misalignment between the coordinates of the images and their corresponding positions on the map. To correct this, a linear transformation was applied to the recorded coordinates to accurately plot the robot's real path. For each trajectory, a set of weights was trained to fit a 5th-degree polynomial to both the x- and y-coordinate sets. The polynomial for each map was trained by selecting a minimum of 15 image coordinates and their estimated corresponding real coordinates in the map. This transformation successfully reconstructs the correct path, especially in maps with a fairly square shape. However, the Ljubljana trajectories exhibited such severe localization inconsistencies that they were considered unusable and had to be discarded. Figure 2 shows the results of this transformation: image coordinates are correctly mapped to map coordinates.

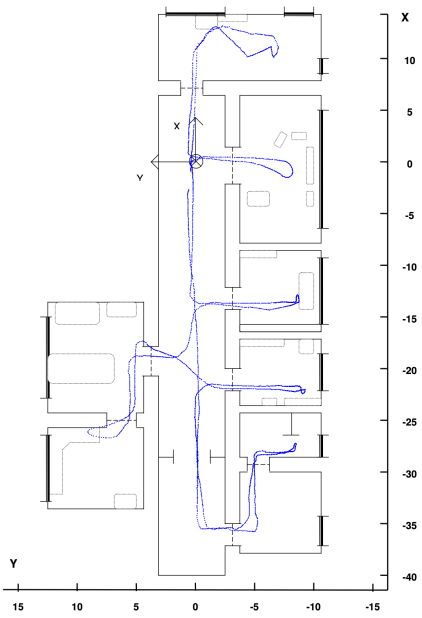


Figure 2. Example of the path followed by the robot in the Freiburg A lab.

B. General Overview

The first part of the project focuses on training a deep learning model to perform feature extraction on the provided images. CNNs and AutoEncoders have been chosen for this task due to their strong performance in image understanding and their ability to generate feature vectors that serve as image descriptors. A Transformer-based model was also considered but ultimately rejected due to its higher complexity and only marginal performance gains, as simpler models already achieved high accuracy. An Autoencoder was ultimately selected for most components of the system, as it demonstrated robust performance across a wide range of lighting conditions.

With the feature extractor trained, the next step involves simulating a real-life scenario in which data from various sensors arrives simultaneously. To achieve this, the ROS2 robotics framework is used due to its ability to integrate multiple nodes and establish real-time communication between them. The first two nodes implemented are responsible for publishing odometry data and camera images, respectively. A third node, tasked with building the graph, receives data from these two sources to construct the topological map.

Initially, this third node computed the robot's new pose based on the latest odometry readings. However, as previously discussed, the odometry data proved to be unreliable, leading to highly inaccurate localization. As a result, localization is instead derived from the positions where the incoming images had been captured, leaving the odometry node obsolete. The camera image node thus is responsible for both localization and map construction.

Once the robot's updated pose is estimated, the incoming image is processed through the feature extractor to obtain its vector representation. This vector is then added to the affinity matrix used in the node extraction process. By monitoring the eigenvalues of the Laplacian matrix, the system can detect significant changes in visual appearance, triggering the creation of a new node.

Each new node is then compared against existing nodes in terms of both pose and visual similarity. If no similar node is found, the new node is added to the graph and connected to the previous one. On the contrary, if a sufficiently similar node is found, the two nodes are merged by averaging their poses and combining their associated images. If not enough visual correspondences are detected to stitch the images, they are simply concatenated. This merging process indicates a loop closure, triggering the adjustment of all node positions within the loop. This correction distributes localization errors across all edges in the loop, rather than concentrating it solely on the last connection, an especially important feature when using real odometry data, where accumulated drift must be managed effectively.

In addition to adjusting node positions, a rewiring mechanism is used to update graph edges when new connections could improve structural coherence. This rewiring is also triggered when the robot passes near an existing node, even if the location is not identified as a new node, further enhancing the consistency of the topological map.

After adding a new node or combining the new node with a previous one, the semantic information of the node is updated. To accomplish this task, an object detector (YOLOv8 [16]) finds all the objects present in their associated image and combines them in a sentence. This sentence is then fed to a language model (CLIP [31]) that outputs a sentence embedding, which is used as the semantic information of the node.

Finally, once two complete trajectories have been processed and their corresponding maps generated, they are merged to improve the robustness of the resulting map and to correct node detection and localization errors. To ensure that as much information as possible is preserved, the combined graph includes the union of the nodes from both individual maps.

This merging process involves comparing all nodes from one map to all nodes from the other. For each node, the best match in the other graph is identified based on both positional proximity and visual feature similarity. If the overall similarity is high enough, the two nodes are merged into a single node in the final graph. If no sufficiently similar node is found, the unmatched node

is added to the final graph and connected to the current node.

After a node is added or merged, its connections are also examined to determine whether any existing edges can be updated or improved based on the newly included node.

C. Model Training and Validation

The feature extractor is the model responsible for generating a robust vector representation of an image that retains as much relevant information as possible while significantly reducing dimensionality. In this project, images were first resized to a resolution of 224×224 and then transformed into a 1024-dimensional feature vector. To ensure consistency and compatibility across the system, all trained models were designed to output vectors of the same dimensionality.

The approach used to train a robust feature extractor involved training a classification model and then using the output of its final representation layer, just before the classification layer, as the feature vector. This method proved effective, as the trained models successfully produced meaningful representations of their environments. The training and validation data consisted of images obtained from 77 trajectories in the COLD dataset, each labeled with the type of room in which it was captured. Of these, 64 trajectories were used for training and 13 for testing, corresponding to an approximate 83–17% split. Testing trajectories included only standard paths to avoid introducing unseen classes. Specifically, 5 out of 32 Saarbrücken trajectories, 5 out of 26 Freiburg trajectories, and 3 out of 19 Ljubljana trajectories were used for testing, with the remainder allocated to training.

To train these models, a pretrained network was used as a backbone, followed by fine-tuning with indoor environment images from the COLD dataset. The StepLR learning rate scheduler was employed during training to stabilize the process. It reduces the learning rate by a factor of *gamma* (0.2) every *step_size* (15) epochs.

The first model tested was a traditional CNN, known for its strong performance in image understanding by applying convolutional kernels across spatial regions of the image. ResNet101 [3] was used as the backbone. Its final two layers were removed and replaced with two fully connected layers, separated by a ReLU activation function. This modified architecture reduced the 2048-dimensional output of ResNet to an intermediate 1024-dimensional vector (used as the extracted feature vector), and then further down to 12 output classes corresponding to those in the COLD dataset. An average pooling layer was inserted before the fully connected layers to reduce spatial dimensions while preserving contextual information, and a dropout [2] unit was included to mitigate overfitting.

The second model explored was an AutoEncoder, a flexible architecture designed to compress an input into a lower-dimensional latent space and then reconstruct the original input from that representation. In this project, the latent space was set to 1024 dimensions, and the representation in this space was used as the extracted feature vector. Figure 3 displays the mentioned architecture. The initial attempt involved training the entire AutoEncoder from scratch; however, this approach proved unstable and difficult to optimize, achieving only around 60% accuracy.

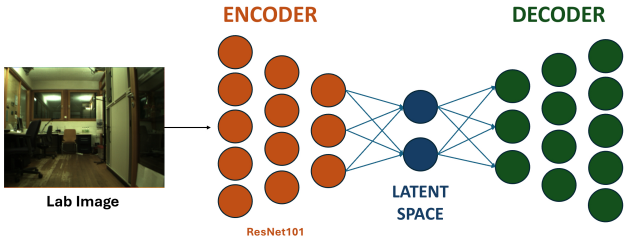


Figure 3. Architecture of the AutoEncoder with ResNet101 backbone.

To address this, the AutoEncoder was redesigned to incorporate the same ResNet101 backbone as the encoder. Its final layers were replaced with trainable fully connected layers that produced the 1024-dimensional latent vector. A decoder network was then trained from scratch to reconstruct the original image. To improve training stability and encourage generalization, batch normalization layers were added after each transposed convolution in the decoder. This revised approach was significantly more stable and even outperformed the CNN in terms of accuracy.

However, the model showed signs of overfitting, with training accuracy approaching 100%. To counteract this, a dropout unit was introduced, which successfully reduced training accuracy while improving validation performance, ultimately resulting in a more robust model when applied to unseen data.

D. ROS2 Structure

ROS2 is a robotics framework that facilitates real-time communication between different nodes using a publisher-subscriber architecture. A node acts as a publisher when it sends a message on a specific topic, and any node subscribed to that topic receives the message upon publication. Messages can carry various types of data, such as numbers, vectors, or booleans. This architecture is well-suited for simulating scenarios similar to those the wheelchair will encounter during the competition,

where multiple data sources transmit information simultaneously.

In this context, the data required to build the topological map includes odometry measurements and camera images. Each data type is handled by a separate node, responsible for extracting relevant information from the dataset, processing it into a suitable format, and publishing it on a designated topic. The *graph_builder* node receives this information, constructs the graph, and publishes a confirmation message on the *graph_alignment* topic.

This notification first triggers the publication of the second trajectory from the *camera node*. Upon the second publication, it initiates the alignment of the two maps by the *map_alignment* node. Once this process is complete, the *map_alignment* node publishes another confirmation message, enabling the *commands* node to operate using natural language commands. A visual representation of the system is shown in Figure 4.

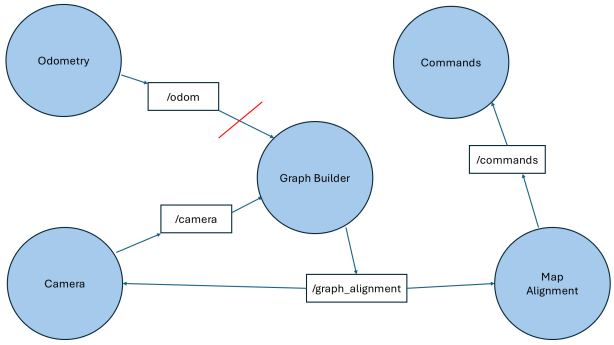


Figure 4. Diagram of the ROS2 architecture. Blue circles represent nodes, while rectangles represent topics. Arrows pointing a topic indicate that the node publishes in that topic, and arrows pointing a node indicate that the node is subscribed to that topic. The *odom* topic connection is crossed because it is not used at the end.

The *odometry* node is responsible for reading the odometry file from the dataset and publishing the readings on the *odom* topic. However, the dataset provides the robot's estimated position at each timestamp, rather than the actual wheel movements typically associated with odometry. To address this and improve the realism of the system, the wheel movements, represented by linear and angular velocities, were computed from the position and time differences between consecutive measurements. These computed velocities were then published using a *CustomOdometry* message, which included both the derived odometry data and the corresponding time interval. As previously mentioned, the original odometry data contained significant discrepancies and could not be used directly.

The *camera* node is responsible for reading images from the corresponding file and feeding them into the feature extractor model to obtain their vector representations. These image features are then published in an *ImageTensor* message on the *camera* topic. The *camera* node is also subscribed to the *graph_alignment* topic, which is used by the graph builder to signal that the initial graph construction is complete. This notification allows the *camera* node to begin publishing images from the second trajectory to be analyzed.

The next node in the architecture is the *graph_builder* node, the main component of the system responsible for constructing a topological graph based on odometry measurements and camera images. Initially, this node subscribed to both the *odom* and *camera* topics, but was finally subscribed only to the latter due to issues with the odometry data. Once the graph is built, it publishes a confirmation message through the *graph_alignment* topic.

The process begins by receiving a feature representation of an image along with its corresponding file name, which contains the pose at which the image was captured. The robot's current pose is then updated using this information. The previously described polynomial [III-A] is applied to transform the robot's position into map coordinates, allowing the entire path to be accurately plotted.

The next step is to perform the node extraction algorithm, which involves detecting when the robot enters a new area different from the one it was previously in, thereby requiring the addition of a new node to the graph. Although this task can be approached using threshold-based techniques, such methods often lack generalizability across different environments, and selecting a robust threshold is typically hard. As explained by Boal and Sánchez-Miralles [6], this task can be accomplished using only visual information and without thresholds by leveraging the Laplacian matrix. This matrix has several properties grounded in graph theory that are useful in tasks such as clustering. In particular, its second smallest eigenvalue, known as the algebraic connectivity or Fiedler value, measures how well connected a graph is. This value will be used to detect significant changes in the visual appearance of the environment, which typically indicate that the robot has entered a new room.

When working with undirected graphs, the computation of the Laplacian matrix begins with the affinity matrix $A \in \mathbb{R}^{n \times n}$, a binary matrix with ones in positions where two nodes are connected and zeros elsewhere, and the degree matrix $D \in \mathbb{R}^{n \times n}$, a diagonal matrix containing the number of connections (degree) of each node.

$$a_{ij} = \begin{cases} 1 & \text{if } i \text{ and } j \text{ are connected} \\ 0 & \text{otherwise} \end{cases}$$
$$d_{ij} = \begin{cases} \sum_{j=1}^{n} a_{ij} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

There exist several definitions of the Laplacian matrix. In this work, the symmetric normalized Laplacian is used, a positive semidefinite matrix whose eigenvalues are always real and nonnegative. This matrix is defined as follows:

$$L_{\text{sym}} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \quad (2)$$

The problem of detecting a change in appearance can be approached as a clustering problem, for which the Laplacian matrix can be effectively used. The idea is to group similar images into clusters such that, when incoming images begin to differ significantly from previous ones, a new cluster is formed, indicating the need to add a new node to the graph. By constructing the affinity matrix as a similarity matrix between previously captured images, the algebraic connectivity will decrease when the robot starts receiving visually distinct images from a new area. As more images from this new area accumulate and show increasing similarity, the algebraic connectivity will rise again. For efficiency and memory considerations, the affinity and degree matrices are limited to the most recent $n = 30$ images. Figure 5 shows the pipeline of the described method.

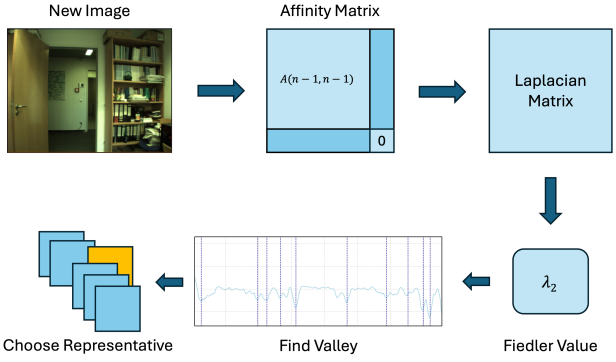


Figure 5. Node extraction pipeline. Inspired by Boal and Sánchez-Miralles [6].

The algebraic connectivity is monitored by computing the Laplacian matrix and extracting its second smallest eigenvalue (the Fiedler value). By plotting a time series of this value, displayed in Figure 6, distinct clusters of similar images can be observed. A peak and valley detection algorithm can then be applied to identify the exact moments of appearance change, signaling that the robot has entered a new location and a new node should be added to the graph.

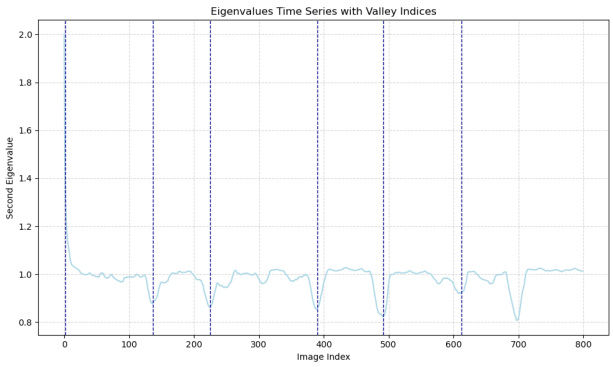


Figure 6. Algebraic connectivity segmented by valleys.

This peak and valley detection algorithm is an online method for detecting valleys in a stream of algebraic connectivity values (λ_2), using two thresholds: γ, the minimum peak value, and δ, the minimum required difference between consecutive peaks and valleys. It tracks local maxima and minima in real-time, alternating between searching for peaks and valleys. When a peak exceeding γ is followed by a drop of at least δ, a valley is anticipated. Once the signal rises by at least δ from the minimum, the valley is confirmed and its index is returned. If no valley is detected, the algorithm returns 0.

After identifying a valley, a representative image must be selected to associate with the new node. The best representative is the image most similar to the others. To avoid recomputing all similarities, the degree matrix, which contains the sum of similarities for each image, can be used: the image corresponding to the maximum value in the degree matrix is selected as the representative.

Once the representative image is selected, a new node is created with its associated ID, pose, image, and visual features, completing the node extraction process. This new node is then compared to existing nodes in the graph to determine whether it corresponds to a previously visited location or a new one. The comparison takes into account both pose and visual information to avoid over-reliance on pose data, which may be inaccurate due to odometry errors. This step is particularly challenging because nodes located at the same position may have completely different associated images if they were captured in opposite directions. Ideally, such nodes should be merged; however, relying too heavily on visual similarity might cause the system to incorrectly associate the new node with a different location.

To address this, the weight of the visual similarity in the overall comparison score is modulated based on the similarity of the angles with which the images were captured. The more similar they are, the higher the weight assigned to visual similarity, up to a maximum of 0.5. The remaining weight is assigned to pose similarity. This

ensures a balanced comparison that accounts for both visual and spatial consistency. The following equations explain how the overall similarity is computed.

Let:

- θ_i be the orientation (pose angle) of node i,
- θ be the orientation of the new pose,
- $\mathbf{v}_i$ be the visual feature vector of node i,
- $\mathbf{v}$ be the visual feature vector of the new pose,
- d_i^{pos} be the position distance of node i to the new pose,
- $d_i^{\text{vis}} = \text{cosine\_distance}(\mathbf{v}_i, \mathbf{v})$

$$\Delta\theta_i = |(\theta_i - \theta + \pi) \bmod 2\pi - \pi| \tag{3}$$

$$w_i = 0.5 \left(1 - \frac{\Delta\theta_i}{\pi}\right) \quad \text{where } w_i \in [0, 0.5] \tag{4}$$

$$s_i = (1 - w_i) \cdot d_i^{\text{pos}} + w_i \cdot d_i^{\text{vis}} \tag{5}$$

After computing the overall similarity with all existing nodes, the most similar node is identified. If the distance between this node and the new node exceeds a defined threshold, the closest node is considered too far, and the new node is added to the graph and connected to the previous node. At this point, a rewiring algorithm is executed to improve the graph structure. The rewiring algorithm performs two checks: (1) whether the new edge between the new node and the current node can be split by introducing one of the current node's neighbors and (2) whether the edges between the current node and its neighbors can be recomputed by routing through the new node.

Both checks involve projecting the new node onto the candidate edge. If the projection is sufficiently close to the new node, this suggests that incorporating the node into the edge would smooth the graph structure. In such cases, the new node is fused, both in terms of pose and visual appearance, with the projected location. Since the projection does not correspond to an existing node (and thus lacks an image), the closest associated image is used during the stitching process. Figure 7 shows a visual representation of the algorithm. The semantic vector of the node is then updated to reflect the new image.

Conversely, if the distance to the closest neighbor falls below the threshold, a loop closure is detected. The closest node is updated with information from the new node: their poses are averaged, and their images are stitched. If not enough matching points are found, the images are concatenated. Figure 8 presents a visual summary. Again, the semantic vector is updated to reflect the modified image. After fusing the two nodes, all nodes within the loop have their positions adjusted to correct odometry errors.

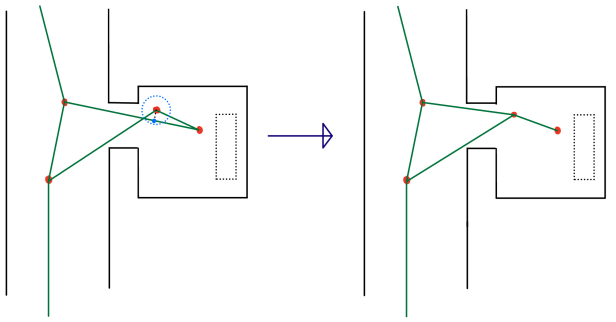


Figure 7. Visual representation of the rewiring algorithm. Blue circle indicates proximity area. If the projection is close enough to the original node, rewiring is performed.

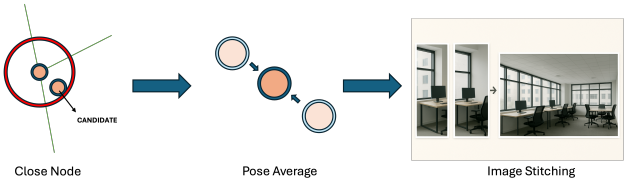


Figure 8. Node fusion process. Red circle indicates proximity area.

Graph optimization is performed using the GTSAM library [34], which distributes the error across all nodes in the loop rather than concentrating it on the final edge. This step would be more critical in systems using raw odometry, where loop closure drift is typically more significant. In this case, however, the total error is limited to the original distance between the two fused nodes. After position adjustment, the rewiring algorithm is run to explore possible new connections by projecting each node in the loop onto graph edges that do not currently include them. Again, is the projection is close enough, the loop gets rewired and a new connection is added. This process is crucial for maintaining a consistent graph structure where edges represent the optimal connections between nodes.

The final function of the *graph_builder* node is triggered when no new node is detected. If the system determines that it is passing extremely close, an order of magnitude closer than other distance thresholds, to an existing node, a loop closure is detected, and the same update procedure is applied to this node. This step is fundamental for correctly connecting nodes when no new relevant location is detected by the algorithm.

The pipeline continues with the *map_alignment* node, which is responsible for aligning the two generated maps. It subscribes to the *graph_alignment* topic and publishes to the *commands* topic. It waits for a confirmation message on the former to begin the alignment process and publishes a confirmation message on the latter to enable the NLP commands module to start operating.

Once the notification is received, the maps are loaded,

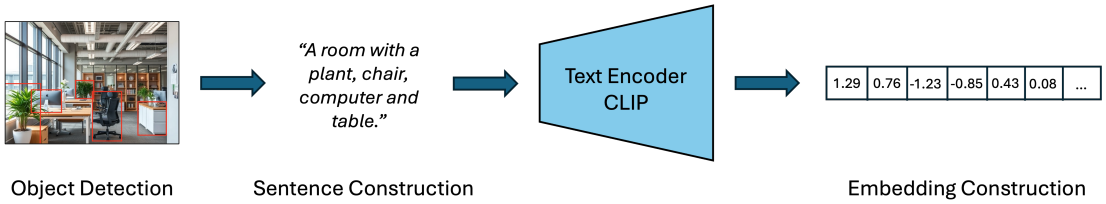


Figure 9. Semantics integration process.

and the alignment process begins. To preserve as much information as possible, the final graph includes the union of nodes from both graphs. Initially, the final graph is a clone of the first graph, and each node from the second graph is compared to the nodes in the first. The initial intention was to run both trajectories in parallel and publish their generated maps through a ROS2 topic. However, experiments revealed that this approach was too ambitious, as the laptop could not handle the memory requirements, and the graphs contained too much information to be sent as messages. In the end, the option to save and load the graphs proved to be more practical. It also enabled the use of previously generated graphs for subsequent analysis, alignment, plotting, or navigation.

For each node in the second graph, a KD-tree is used to find the closest nodes from the first graph in terms of spatial distance. The k nearest neighbors are then compared using both their relative distances and visual distances, the latter computed using cosine measure. Visual similarity is also considered with the intention to avoid over-relying on pose information, which could contain significant error in an odometry-based setting.

Once the best match is found, its overall proximity is evaluated against a threshold to decide whether to fuse the nodes or add the new node to the final graph. If the distance is below the threshold, the nodes are considered to represent the same point, and their information is merged. The fusion procedure is the same as in the previous node: their poses are averaged, and their images are stitched. Once the image is updated, a new feature vector is generated, and the semantic information gets updated.

If the distance exceeds the threshold, the nodes are considered to represent distinct locations, and the new node is added to the graph. It is connected to the previous node, and a rerouting procedure begins. All neighbors of the previous node are analyzed to determine if their edges can be split by inserting the new node. This is done by computing the cosine of the angle between the direction vector from the previous node to each neighbor and the direction to the new node. The idea is that closely aligned edges represent an opportunity to insert the new node and place a new edge in that direction. If the closest edge is sufficiently aligned, it is split to pass through the new node. Additionally, edges from the new node are evaluated to determine if the inclusion of the new node improves the graph structure, again using cosine similarity between edge directions to identify potentially replaceable edges. Figure 10 shows a visual representation of the process.

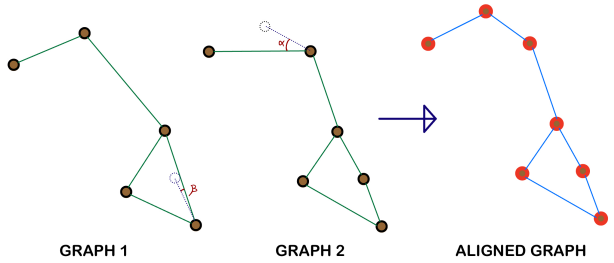


Figure 10. Map alignment process. Cosines of α and β are analyzed to check whether a rerouting should be performed.

Once the map alignment process is complete, the node saves the final graph and publishes a confirmation message, enabling the NLP commands node to begin operation.

Finally, once the final graph is obtained, the map is ready to be navigated using natural language commands. The command node allows the user to enter a query indicating where they want to go, and the system identifies the node that is closest to that location. The user's query is translated from any language into English before processing, making the system accessible to everyone.

A sentence embedding is generated using the CLIP [31] model and compared to the semantic embeddings (of the same dimensionality) associated with all nodes in the graph. Using the cosine similarity function, the system finds the closest embedding, which represents the location the user most likely intended. For example, if the user says "I want to go to the kitchen", the sentence is converted into an embedding and compared to those of the graph nodes. Nodes whose images contain common kitchen-related objects, such as chairs, fridges, pans, or cutlery, will have embeddings close in space to that of the user's sentence, and one of them will be selected as the closest match. Figure 9 shows a visual summary of the algorithm.

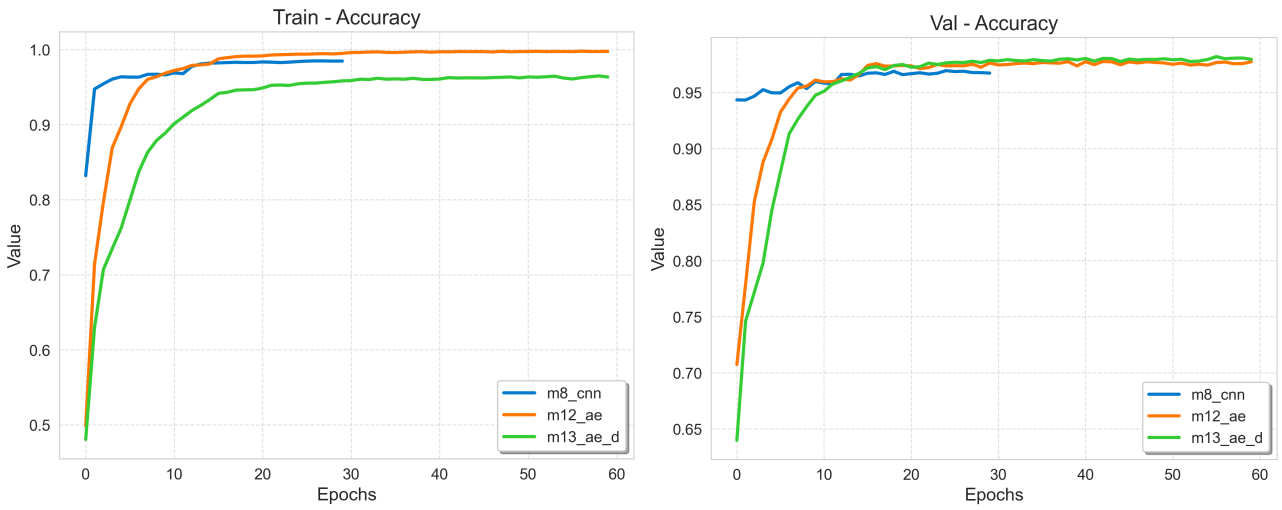


Figure 11. Training and validation accuracy of the three selected models.

IV. Experiments

The purpose of this section is to provide a guideline for potential reproducibility of the project, as well as insights into the hardware and software resources utilized and the different experiments carried out with the system.

A. Model Preparation

The preparation of the models used as feature extractors has been done with PyTorch (2.5.1+cu124) [23], a Python library focused on deep learning systems. This library supports the use of GPU units for parallel training, which has been carried out using the university's virtual classroom computers.

The training of the model starts by running the `train.py` module inside `/src`. First, it downloads the COLD dataset if it has not been downloaded yet. The data is then processed and transformed into the correct format for subsequent use. Afterwards, the dataset is prepared with an 83–17 split between training and testing, further dividing the training set in training and validation (80–20 split). Images in the dataset are resized to a resolution of 224×224 and normalized.

With the data prepared, the next step is to choose the model to train: the CNN or the AutoEncoder. Different hyperparameters can be tuned for training, which are displayed in Table III.

For efficiency, both models are implemented as classes inheriting from `torch.nn.Module`. They contain the `resnet101` backbone loaded with default weights, along with an added sequence of layers to perform their respective tasks, as explained in the previous section. The models include a `forward` function for training purposes and an `extract_features` function for feature extraction. Figure 11 shows the training and validation accuracy of the three best-performing models: a CNN (`m8_cnn`), an AutoEncoder (`m12_ae`), and an AutoEncoder with an added dropout unit to reduce overfitting (`m13_ae`). Both AutoEncoders started with significantly lower accuracy compared to the CNN but eventually surpassed it on the validation set.

Table III. Model Hyperparameters and Training Configuration.

Parameter	Value
`epochs`	60
`lr_backbone`	1×10^{-5}
`lr`	1×10^{-3}
`batch_size`	128
`dropout`	0.5
`step_size`	15
`gamma`	0.2
Loss Function	Cross Entropy
LR Scheduler	StepLR
Optimizer	Adam

B. ROS2 Environment Management

The ROS2 (Humble Hawksbill) environment is structured as follows:

```
vts_ws/
• images/
• launch/
    – project.launch.py
• src/
    – vts_camera/
      * vts_camera/
        · camera.py
        · camera_node.py
    – vts_odom/
      * vts_odom/
        · odometry.py
        · odometry_node.py
    – vts_graph_building/
```

```
    * vts_graph_building/
      · graph_builder.py
      · graph_builder_node.py
      · node.py
  – vts_commands/
    * vts_commands/
      · commands.py
      · commands_node.py
  – vts_map_alignment/
    * vts_map_alignment/
      · map_alignment.py
      · map_alignment_node.py
      · graph_class.py
  – vts_msgs/
    * msg/
      · ImageTensor.msg
      · CustomOdometry.msg
```

The system starts when the launch file is executed. This file specifies the name of the feature extractor to be used, the selected laboratory, the two specific trajectories within that laboratory, and the nodes to be deployed. It also provides the weights for the polynomial used to map image coordinates to map coordinates.

Each ROS2 node is contained within its own package, which includes a standard `.py` file responsible for the internal logic of the node, and a `_node.py` file responsible for communication with other nodes. The `graph_builder` package also includes a specific `Node` class representing the graph's nodes. This class provides several functions to enable the integration of semantic information into the graph. The `map_alignment` package additionally includes a simple `Graph` class that facilitates the construction of the final navigation structure.

When the full process is executed, the initial graphs are saved as `graph_1.pkl` and `graph_2.pkl`. Both files are then loaded by the `map_alignment` node, which generates the `final_graph.pkl` file. Simultaneously, the path followed by the robot is plotted, and the final aligned map is saved as a `.png` image.

The process highly depends on the hyperparameters selected, with the performance varying from map to map for the same set of hyperparameters. For that reason, each map has a unique set of optimal hyperparameters that produce the most robust result, shown in Table IV. These parameters include the γ and δ values of the peak and valley detection algorithm and distance thresholds for node fusion and edge rewiring.

Table IV. Hyperparameters used for each environment. `dist_thrs` represents the node fusion distance, `proj_dist` the rewiring maximum distance when a loop closure is found and `ext_proj_dist` the rewiring maximum distance when no loop closure is found. All distances measured in metres (m).

Parameter	Frei. A	Frei. Ext.	Saar. A	Saar. Ext.
gamma	0.5	0.4	0.5	0.4
delta	0.11	0.09	0.11	0.09
dist_thrs	3.5	2.0	3.0	4.0
proj_dist	3.0	3.5	1.25	4.25
ext_proj_dist	2.5	3.5	1.0	4.0

C. Description of the Experiments

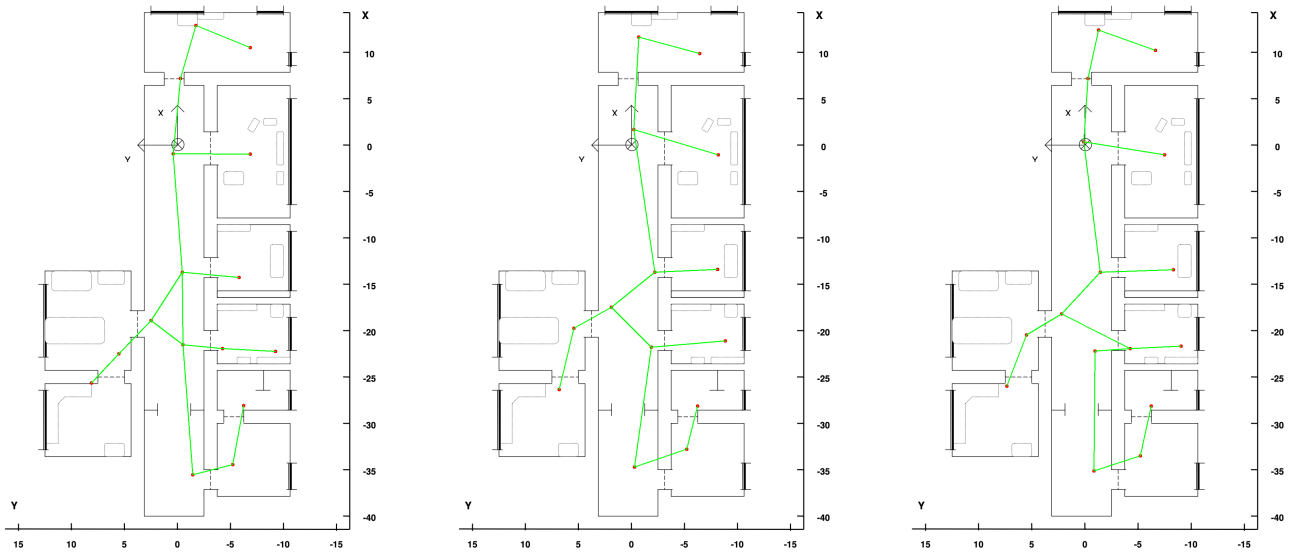
Experiments with the COLD dataset were conducted to evaluate the performance of the complete pipeline in a simulation environment. The four environments tested are the Freiburg and Saarbrücken laboratories, each evaluated using both standard and extended trajectories.

The goal of these experiments is to demonstrate that the system can build a topological map that accurately represents the environment. This will be considered successful if the nodes effectively describe key points of interest, such as rooms, doorways, and corridor centers, while avoiding redundancy or sparsity, and if the edges between nodes are optimal for navigation. The evaluation metric for this component will be primarily visual, based on the generated graphs. The commands node can be assessed by verifying whether the destination it produces matches the user's intended location, either by checking the node's position or inspecting the associated image.

The remaining of this section will focus on the different experiments carried out, starting with the creation of topological maps for the four different environments and following with the NLP commands module.

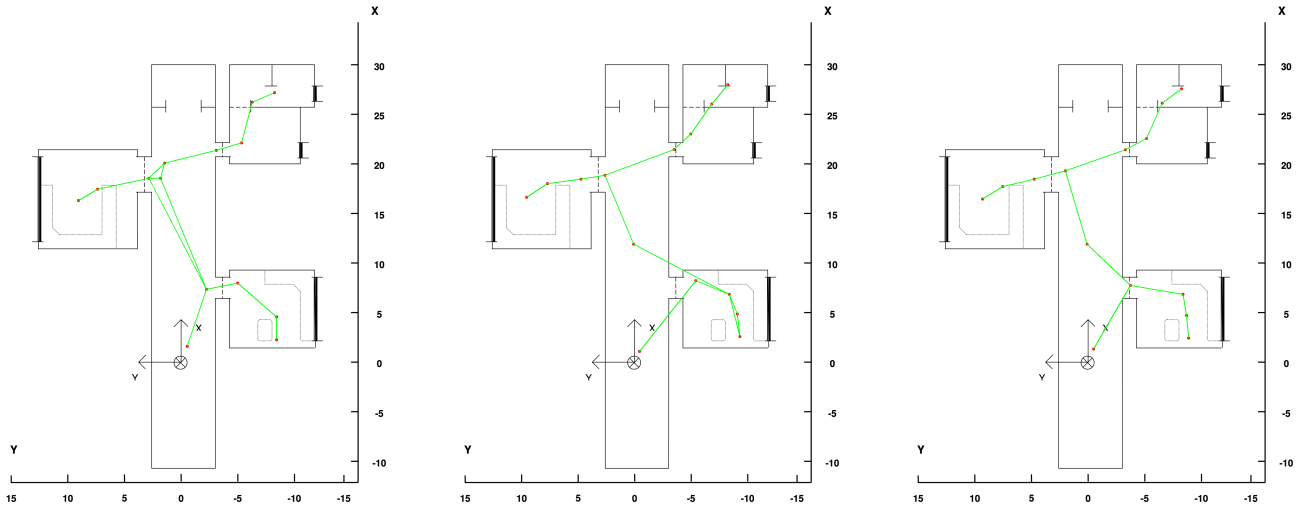
Figure 12 shows the two generated graphs from the Freiburg A lab, along with the aligned map, which represents the final output of the system. In this case, both individual trajectories produce highly consistent results, with nodes representing key locations in various rooms, doorways, and corridors. The edges are generally well-placed, although each trajectory includes one edge that passes through a wall. This situation, while not ideal, is not necessarily incorrect, as edges describe traversable paths between nodes without implying that the robot needs to follow the straight line of the edge. As mentioned at the beginning of the report, topological maps are not intended to be perfectly accurate representations of the physical environment, so some minor misplacements are to be expected. The aligned map addresses one of these issues and merges the nodes from both graphs. While this fusion introduces a sharp turn in the middle of the corridor, it also improves the connections within the rooms and enhances the overall structure.

Figure 13 displays the graphs obtained from the Freiburg Extended lab. In this case, the initial trajectories are very similar but each presents certain issues. Trajectory (a), collected under cloudy conditions, is not properly rewired and results in a graph that is suboptimal for navigation. The second graph contains two edges that run through a wall. However, this case highlights the effectiveness of the map alignment process, which successfully resolves all these issues and produces a clean, accurate graph. By combining both maps, individual errors are mitigated, resulting in a final graph that is well-suited for navigation.



(a) Single trajectory in night conditions. **(b)** Single trajectory in sunny conditions. **(c)** Aligned map.

Figure 12. Freiburg A: Comparisons of mapping results.



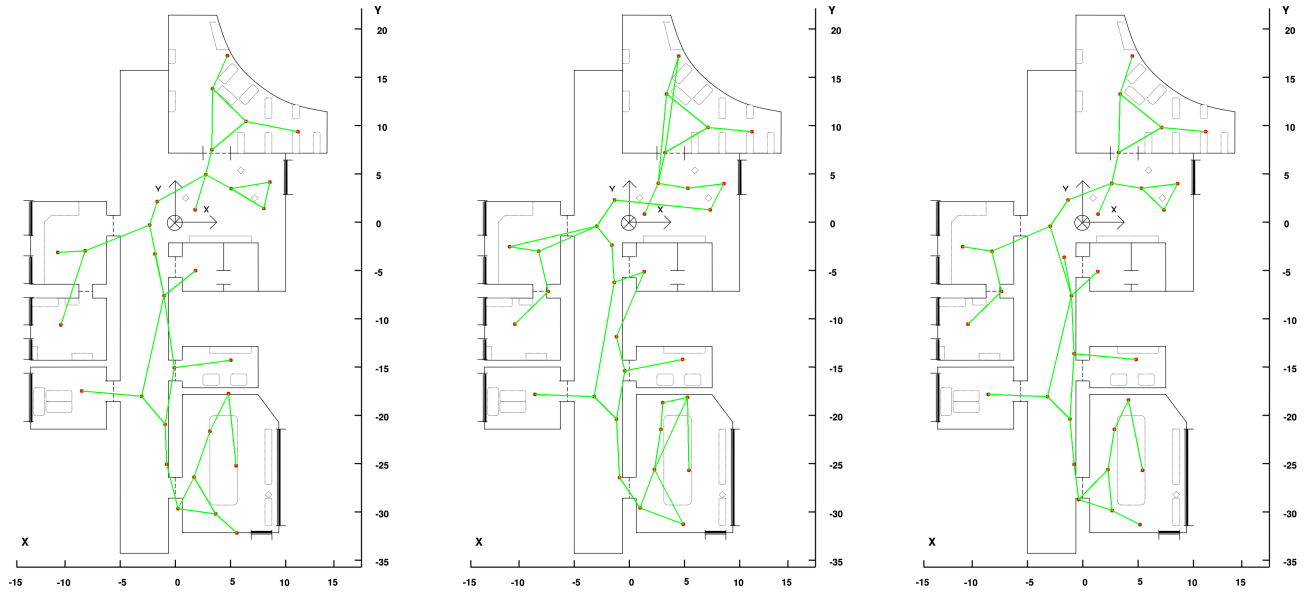
(a) Single trajectory in cloudy conditions. **(b)** Single trajectory in sunny conditions. **(c)** Aligned map.

Figure 13. Freiburg Ext: Comparisons of mapping results.

Figure 14 presents the results from the Saarbrücken A lab, the most complex environment considered, featuring numerous rooms and tight turns in the robot's path. It includes a U-shaped sub-trajectory in the lowest room, which was one of the key motivations for implementing the rewiring mechanism. Both initial trajectories perform well in mapping their surroundings, producing denser graphs due to the lab's particular layout. Despite the increased complexity, the system produces a robust result, with good node placement and connectivity, and only a few edges passing through walls. The alignment process once again improves the graph by correcting some misplacements and removing redundancies. The U-shaped area is accurately represented, as well as the small rooms located on both sides of the corridor.

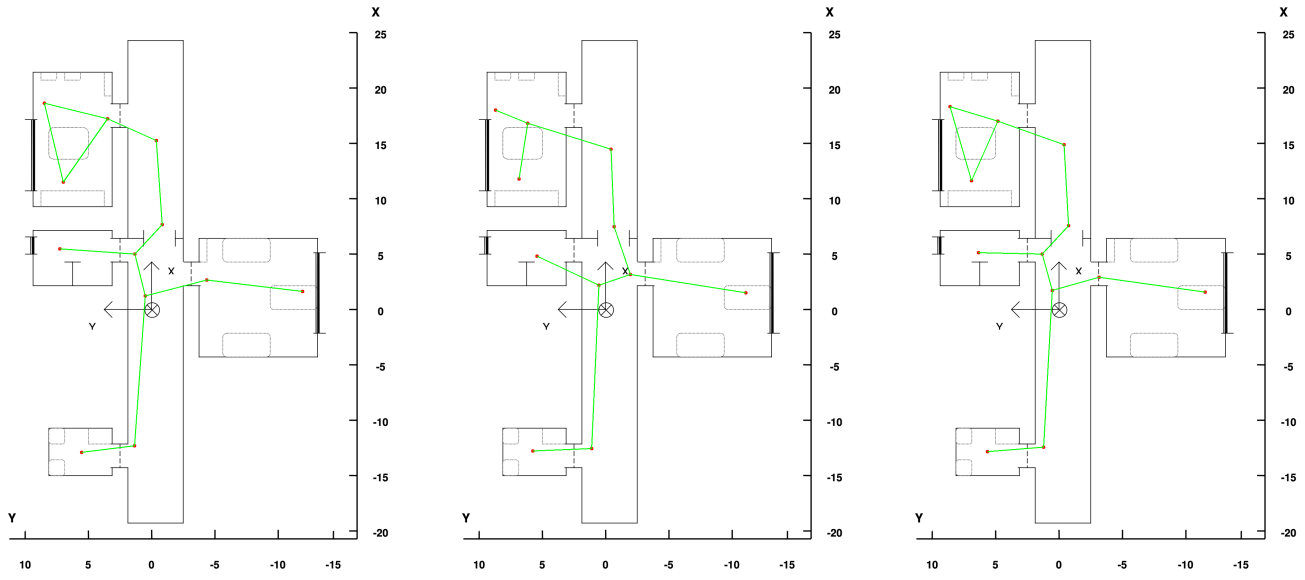
Finally, Figure 15 presents the results from the last trajectory in the Saarbrücken Extended laboratory. This case features two fairly distinct trajectories that represent the environment almost perfectly. Only the second trajectory exhibits two slightly misplaced edges that skip the doorway and run directly through a wall. However, the aligned map once again resolves the issue, producing an almost flawless graph with nodes precisely where they should be and edges forming a consistent structure suitable for navigation.

Regarding the NLP commands module, numerous ex-



(a) Single trajectory in cloudy conditions. **(b)** Single trajectory in night conditions. **(c)** Aligned map.

Figure 14. Saarbrücken A: Comparisons of mapping results.



(a) Single trajectory in cloudy conditions. **(b)** Single trajectory in sunny conditions. **(c)** Aligned map.

Figure 15. Saarbrücken Ext: Comparisons of mapping results.

periments have been conducted with consistent success. Sentences in any language can be input, and the system translates them into English before processing. The graph then outputs the node closest to the requested room, along with its associated image. To test the module's performance, the output image was analyzed and the resulting node's room was verified to match the intended destination. Figure 16 shows an example of the system in use. Note that the query sentence is in Spanish ("Ve a la sala de ordeandores") and the system perfectly handles it. The associated image also represents the desired destination. The performance of this module highly depends on the object detector and the language model. Both models can be improved by increasing the number of parameters used, with its corresponding increase in energy consumption and response time. Depending on the application, different verions of the models can be employed.

During experimentation, two issues were identified that offer room for improvement. The first involves the need for a similarity threshold that allows the system to

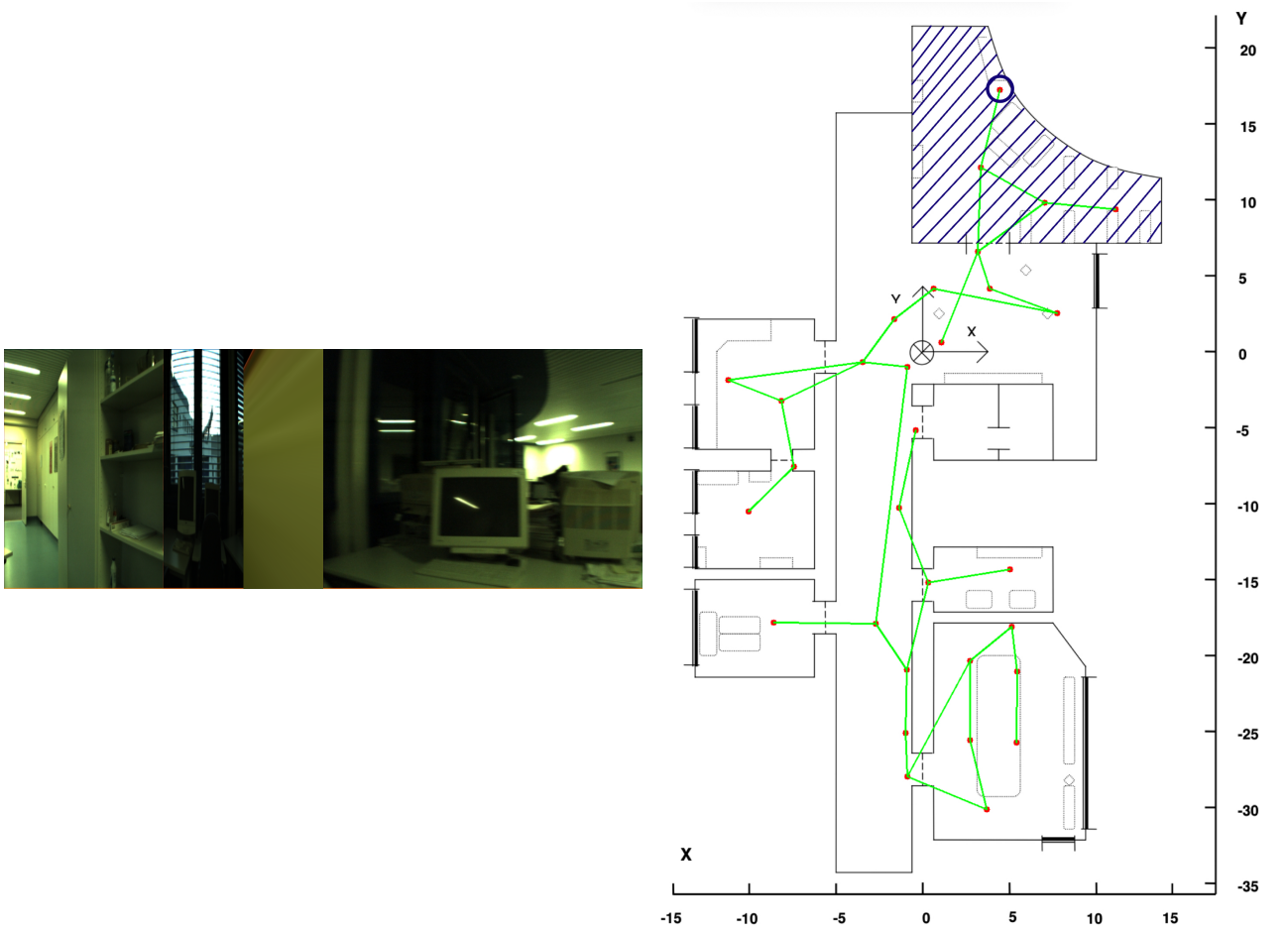


Figure 16. Node generated and associated image when asked: "Ve a la sala de ordenadores". Blue highlight indicates the selected node and the computers rooms in the map.

recognize when the requested location does not exist in the generated map. If the maximum embedding similarity does not exceed this threshold, the system should inform the user that no matching location was found. However, no threshold was found that could consistently distinguish between existing and non-existent places.

The second issue relates to how images are stitched during the graph-building process. When stitching fails, the images are simply concatenated and the pipeline continues. This can result in images being combined when they should not be, leaving the node with a corrupted associated image, which negatively affects the object detection process. An example of this situation can be found in Figure 18, in Appendix 1. The associated image corresponds to the place where the user intends to go, but it includes other concatenated images that do not represent that place. Still, the selected node is the correct one. This issue can be mitigated by either reducing the number of required keypoint matches or avoiding image concatenation if the error raises from a different cause.

Further experiments were conducted using the CNN extractor instead of the AutoEncoder. A considerable decrease in performance was observed, as fewer nodes were detected using the same set of hyperparameters. This can be attributed to a poorer feature representation produced by the final two Linear layers of the CNN extractor, in comparison to the latent space representation generated by the AutoEncoder. This outcome also supports the initial hypothesis that better classifiers yield better image representations. The results are shown in the Appendix in Figure 19, Figure 20, and Figure 21.

Additional experiments were conducted by varying the hyperparameter values to assess their influence on the system's overall performance. The most significant parameters were γ and δ, which control the peak and valley detection algorithm, as well as the distance threshold used to determine whether two nodes are close enough to be merged. As expected, making the valley detection stricter led to fewer nodes being identified, resulting in a sparser graph that represents the environment less accurately. An example of this can be found in the Appendix, in Figure 22.

V. Results

The experiments demonstrate the robust performance of the system across different trajectories and lighting conditions. While the graph generated prior to the alignment phase is already consistent and provides a reliable environment for navigation, the alignment process further refines the result, producing well-defined graphs.

The training of the feature extractor has been successful, as the features employed are highly representative of the different places. This is supported by the placement of nodes in the middle of rooms and doorways, where the features change according to the environment, specially notable in Figure 15 (c) and Figure 12 (c). As a result, the system can accurately detect appearance changes. This validates the initial approach of training a classifier model and later using its feature map as the final model.

The effectiveness of the node extraction process can be observed when analyzing the node positions. According to the logic of the algorithm, nodes should be placed at the most representative points of different locations. These points typically correspond to room centers and doorways, where appearance changes are most pronounced. This expectation aligns with the final result: nodes are generally placed at key locations visited by the robot, and edges connect them in a structure well-suited for navigation. This further confirms the success of the peak-and-valley algorithm, as valleys are correctly detected and image clustering is robust. The selection of the representative node for each cluster also appears to work reliably.

In some cases, nodes are positioned such that the edge connecting them is forced to pass through a wall. While this is not a critical issue, the results would be more consistent if this did not occur. This situation typically arises when the robot makes quick turns and the affinity matrix lacks sufficient images from the new location, causing adjacent nodes to be placed too far apart. This suggests an opportunity for improvement, possibly through the introduction of a tight-turn detection system to adjust behavior in such scenarios.

The mapping of image coordinates to map coordinates functions well overall. The fitted polynomial consistently adjusts the incoming coordinates for accurate plotting on the map. However, in some cases, especially in very rectangular maps, the plotted trajectory intersects with walls. This is likely due to higher polynomial fitting errors at extreme values.

Loop closure detection is a critical component of topological maps. The correct functioning of this module is essential to ensure location consistency and avoid redundancy. The detection system demonstrates excellent results, especially considering how frequently the robot exits and re-enters rooms within a short time span, effectively closing loops. The system appears to manage rooms very well, with node fusion functioning properly and the graph optimization module producing robust locations. Notably, there are no false positives, likely due to the strict conditions required to confirm a loop closure. There are some false negatives, particularly in the Saarbrücken A lab (Figure 14, but these are corrected during the alignment phase. The node fusion algorithm also works remarkably well, in particular when handing new connections and poses. Image stitching is correctly done in most cases, but leaves some room for improvement when not enough keypoint matches are found.

The rewiring system performs as intended, ensuring that loop nodes are connected in the most optimal way. Given the high number of rooms and frequent small loops in these environments, it is important to detect when rewiring can improve the graph's structure by preventing redundant parallel edges. Analysis of the generated graphs shows that the system performs well in the vast majority of cases. One exception is found in the second trajectory of the Saarbrücken A lab (Figure 14, where rewiring fails, likely because the projected node is too far away. However, this inconsistency is resolved during map alignment. In the Freiburg A lab (Figure 12, there is a particularly illustrative example of the system's effectiveness: in the top room, the node at the doorway is created as the robot exits rather than enters. Prior to the introduction of the rewiring system, the node at the origin of the coordinate system was directly connected to the top node, resulting in a poorly connected graph.

Graph alignment was introduced to leverage the varied lighting conditions provided by the COLD dataset, enhancing the system's robustness in changing environments. Its inclusion has proven to be a key factor in the overall success of the mapping process, correcting nearly all errors from earlier phases and ensuring that the final graphs accurately represent the environment while being optimized for navigation. It not only refines slightly misaligned nodes, but also adds missing ones and corrects erroneous edges. In effect, alignment acts as an averaging mechanism between two graphs, mitigating errors from both. As is typical with ensemble methods, combining more than two maps is expected to yield even better results.

Finally, the NLP commands module also shows robust performance. The system reliably identifies similar embeddings and outputs the correct node in nearly all test scenarios. Failures were mostly caused by corrupted images that had been concatenated with too many dissimilar ones, negatively affecting object detection performance. It is important to note that this module depends on both the object detector and the language model, both of which can be further improved if needed. This component significantly enhances the system's accessibility, allowing it to be used by people regardless of their technical knowledge

or spoken language. This is a crucial consideration, given that the system is intended for use by individuals with disabilities, for whom increased accessibility is especially valuable.

In conclusion, the system demonstrated a high degree of consistency and reliability across diverse trajectories and environmental conditions. The visual results showed topological maps that were not only structurally valid but also semantically meaningful and suitable for real-time interaction. Each component of the pipeline (feature extraction, node extraction, loop closure, map alignment, and semantic embedding) played a distinct and verifiable role in the system's performance. These results confirm the practical applicability of the developed architecture and highlight its potential for deployment in assistive robotics and other real-world scenarios.

VI. Conclusion and Future Work

This work presents a pipeline for constructing a visual topological SLAM system leveraging deep learning techniques, with the goal of enabling voice-controlled navigation in indoor environments. The system demonstrates the feasibility of generating a robust and lightweight topological map using only image data and odometry, enriched with semantic context and implemented with a ROS2 framework.

During development, several key objectives were met. A feature extractor was trained using an AutoEncoder architecture with a ResNet101 backbone, producing a 1024-dimensional latent space capable of preserving visual coherence across diverse lighting conditions. This embedding formed the basis for node extraction and loop closure detection, providing a compact yet informative representation of the visual environment.

The approach to node extraction was designed to avoid arbitrary thresholds by leveraging the algebraic connectivity of the Laplacian matrix associated with the graph. This allowed the system to detect significant changes in appearance that correspond to new locations, enabling scalable and adaptive graph construction.

Loop closure detection was implemented using a hybrid similarity measure that considered both pose and visual appearance. When a loop is identified, the GTSAM library is used to optimize the graph structure by distributing error across the connected nodes. Additionally, a rewiring mechanism is employed to refine graph connectivity by introducing more representative connections where appropriate, enhancing the coherence of the topological map.

One of the system's notable contributions is the development of a map alignment module that merges independently generated trajectories into a unified graph. This module assessed spatial and visual proximity between nodes, merging or incorporating nodes to improve the overall structure. The resulting graph exhibits greater consistency and reduced redundancy, providing a foundation for subsequent navigation tasks.

The integration of semantic information into the graph was achieved through object detection using YOLOv8 and sentence embeddings generated by CLIP. Each node was enriched with a semantic vector, enabling the interpretation of natural language commands in multiple languages. The inclusion of language understanding into the SLAM pipeline marks a step toward more intuitive and accessible assistive navigation systems.

Despite the system's positive performance, several areas remain for future improvement. A key objective is the integration of real odometry data from a wheelchair platform to evaluate the system's robustness in realistic conditions, particularly with respect to localization drift, an issue that topological maps aim to mitigate.

Other open challenges include handling invalid semantic queries. Defining a reliable mechanism to reject such inputs is essential for improving overall robustness. Additionally, the current image stitching method occasionally produces images that degrade semantic interpretation. Future iterations may benefit from more advanced stitching algorithms or different strategies when feature matching fails.

Looking ahead, the system could be extended to support multi-agent mapping, where different agents collaboratively explore and merge their respective maps. Such functionality would increase scalability, and redundancy, features particularly valuable in large or dynamic environments. Overall, the methods and architecture developed in this work provide a strong foundation for accessible, intelligent indoor navigation, integrating insights from robotics, deep learning, graph theory, and natural language processing.

References

[1] KTH. *The COLD database*. 2009. URL: https://www.cas.kth.se/COLD/.

[2] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958. ISSN: 1532-4435.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *arXiv preprint arXiv:1512.03385* (2015).

[4] Yi Hou et al. "Convolutional neural network-based image representation for visual loop closure detection". In: *2015 IEEE International Conference on Information and Automation* (2015), pp. 2238–2245. DOI: 10.48550/arXiv.1504.05241.

[5] Niko Sünderhauf et al. "On the performance of ConvNet features for place recognition". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 4297–4304. DOI: 10.1109/IROS.2015.7353986.

[6] Jaime Boal and Álvaro Sánchez-Miralles. "Online topological segmentation of visual sequences using the algebraic connectivity of graphs". In: *Robotica* 34.10 (Oct. 2016), pp. 2400–2413. DOI: 10.1017/S0263574715000053.

[7] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Cham: Springer International Publishing, 2016, pp. 21–37. ISBN: 978-3-319-46448-0.

[8] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[9] Varun Murali et al. "Utilizing semantic visual landmarks for precise vehicle navigation". In: Oct. 2017, pp. 1–8. DOI: 10.1109/ITSC.2017.8317859.

[10] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 6517–6525. DOI: 10.1109/CVPR.2017.690.

[11] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. DOI: 10.5555/3295222.3295349.

[12] Fabian Blochliger et al. "Topomap: Topological Mapping and Navigation Based on Visual SLAM Maps". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 3818–3825. DOI: 10.1109/ICRA.2018.8460641.

[13] Michael Bloesch et al. "CodeSLAM - Learning a Compact, Optimisable Representation for Dense Visual SLAM". In: June 2018, pp. 2560–2568. DOI: 10.1109/CVPR.2018.00271.

[14] Ren C. Luo and Michael Chiou. "Hierarchical Semantic Mapping Using Convolutional Neural Networks for Intelligent Service Robotics". In: *IEEE Access* 6 (2018), pp. 61287–61294. DOI: 10.1109/ACCESS.2018.2873597.

[15] Charles R. Qi et al. " Frustum PointNets for 3D Object Detection from RGB-D Data ". In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, June 2018, pp. 918–927. DOI: 10.1109/CVPR.2018.00102.

[16] Joseph Redmon and Ali Farhadi. "YOLOv3: An Incremental Improvement". In: (Apr. 2018). DOI: 10.48550/arXiv.1804.02767.

[17] Martin Runz et al. " MaskFusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects ". In: *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2018, pp. 10–20. DOI: 10.1109/ISMAR.2018.00024.

[18] Chao Yu et al. "DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 1168–1174. DOI: 10.1109/IROS.2018.8593691.

[19] Fangwei Zhong et al. "Detect-SLAM: Making Object Detection and SLAM Mutually Beneficial". In: *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2018, pp. 1001–1010. DOI: 10.1109/WACV.2018.00115.

[20] Kevin Doherty et al. "Multimodal Semantic SLAM with Probabilistic Data Association". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 2419–2425. DOI: 10.1109/ICRA.2019.8794244.

[21] Lachlan Nicholson et al. "QuadricSLAM: Dual Quadrics From Object Detections as Landmarks in Object-Oriented SLAM". In: *IEEE Robotics and Automation Letters* 4.1 (2019), pp. 1–8. DOI: 10.1109/LRA.2018.2866205.

[22] Kyel Ok et al. "Robust Object-based SLAM for High-speed Autonomous Navigation". In: *2019 International Conference on Robotics and Automation (ICRA)*. 2019, pp. 669–675. DOI: 10.1109/ICRA.2019.8794344.

[23] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[24] Alexey Bochkovskiy et al. *YOLOv4: Optimal Speed and Accuracy of Object Detection*. Apr. 2020. DOI: 10.48550/arXiv.2004.10934.

[25] Devendra Singh Chaplot et al. "Neural Topological SLAM for Visual Navigation". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 12872–12881. DOI: 10.1109/CVPR42600.2020.01289.

[26] Sourabh Vora et al. "PointPainting: Sequential Fusion for 3D Object Detection". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 4603–4611. DOI: 10.1109/CVPR42600.2020.00466.

[27] Yuwei Wang et al. "Robust Loop Closure Detection Integrating Visual–Spatial–Semantic Information via Topological Graphs and CNN Features". In: *Remote Sensing* 12.23 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12233890.

[28] Ali Agha et al. "NeBula: Quest for Robotic Autonomy in Challenging Environments; TEAM CoSTAR at the DARPA Subterranean Challenge". In: *arXiv e-prints* (Mar. 2021). DOI: 10.48550/arXiv.2103.11470.

[29] Yuan Luo et al. "Detection of loop closure in visual SLAM: a stacked assorted auto-encoder based approach". In: *Optoelectronics Letters* 17.6 (2021), pp. 354–360. ISSN: 1993-5013. DOI: 10.1007/s11801-021-0156-9.

[30] AJ Piergiovanni et al. " 4D-Net for Learned Multi-Modal Alignment ". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2021, pp. 15415–15425. DOI: 10.1109/ICCV48922.2021.01515.

[31] Alec Radford et al. "Learning Transferable Visual Models From Natural Language Supervision". In: *Proceedings of the 38th International Conference on Machine Learning*. 2021. DOI: 10.48550/arXiv.2103.00020.

[32] Jiaming Sun et al. "LoFTR: Detector-Free Local Feature Matching with Transformers". In: *CVPR* (2021).

[33] Hriday Bavle et al. "Situational Graphs for Robot Navigation in Structured Indoor Environments". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 9107–9114. DOI: 10.1109/LRA.2022.3189785.

[34] Frank Dellaert and GTSAM Contributors. *borglab/gtsam*. Version 4.2a8. May 2022. DOI: 10.5281/zenodo.5794541.

[35] Nathan Hughes et al. "Hydra: A Real-time Spatial Perception System for 3D Scene Graph Construction and Optimization". In: *Robotics: Science and Systems XVIII* (2022). DOI: 10.15607/RSS.2022.XVIII.050.

[36] Steven Macenski et al. "Robot Operating System 2: Design, architecture, and uses in the wild". In: *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074.

[37] Ygor C. N. Sousa and Hansenclever F. Bassani. "Topological Semantic Mapping by Consolidation of Deep Visual Features". In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4110–4117. DOI: 10.1109/LRA.2022.3149572.

[38] Yuwei Wang et al. "Transformer-based descriptors with fine-grained region supervisions for visual place recognition". In: *Knowledge-Based Systems* 280 (2023), p. 110993. ISSN: 0950-7051. DOI: 10.1016/j.knosys.2023.110993.

[39] Mitsuki Yoshida et al. "Active Semantic Localization withnbsp;Graph Neural Embedding". In: *Pattern Recognition: 7th Asian Conference, ACPR 2023, Kitakyushu, Japan, November 5–8, 2023, Proceedings, Part I*. Kitakyushu, Japan: Springer-Verlag, 2023, pp. 216–230. ISBN: 978-3-031-47633-4. DOI: 10.1007/978-3-031-47634-1_17.

[40] Siting Zhu et al. "SNI-SLAM: Semantic Neural Implicit SLAM". In: *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2023), pp. 21167–21177. DOI: 10.1109/CVPR52733.2024.02000.

[41] M. Chen et al. "Visual simultaneous localization and mapping (vSLAM) algorithm based on improved Vision Transformer semantic segmentation in dynamic scenes". In: *Mechanical Sciences* 15.1 (2024), pp. 1–16. DOI: 10.5194/ms-15-1-2024.

[42] Shibin Song et al. "Loop closure detection of visual SLAM based on variational autoencoder". In: *Frontiers in Neurorobotics* Volume 17 - 2023 (2024). ISSN: 1662-5218. DOI: 10.3389/fnbot.2023.1301785.

[43] Hangzhou Qu et al. "Multi-scale parallel gated local feature transformer". In: *Scientific Reports* 15.1 (2025), p. 7684. ISSN: 2045-2322. DOI: 10.1038/s41598-025-91857-5.

APPENDIX

APPENDIX 1: ADDITIONAL FIGURES

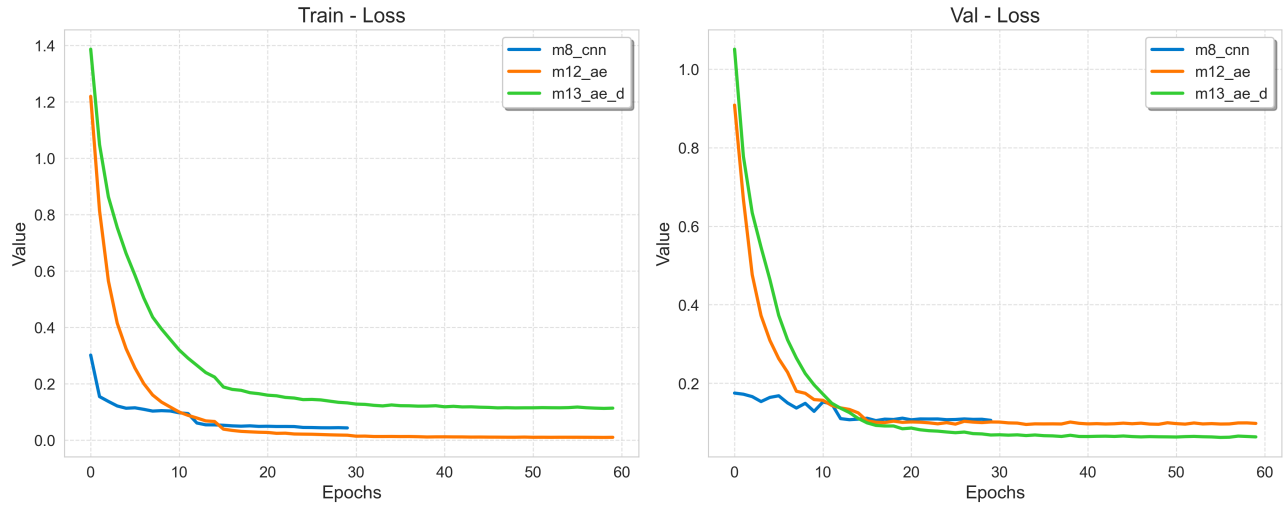


Figure 17. Training and validation loss of the three selected models.

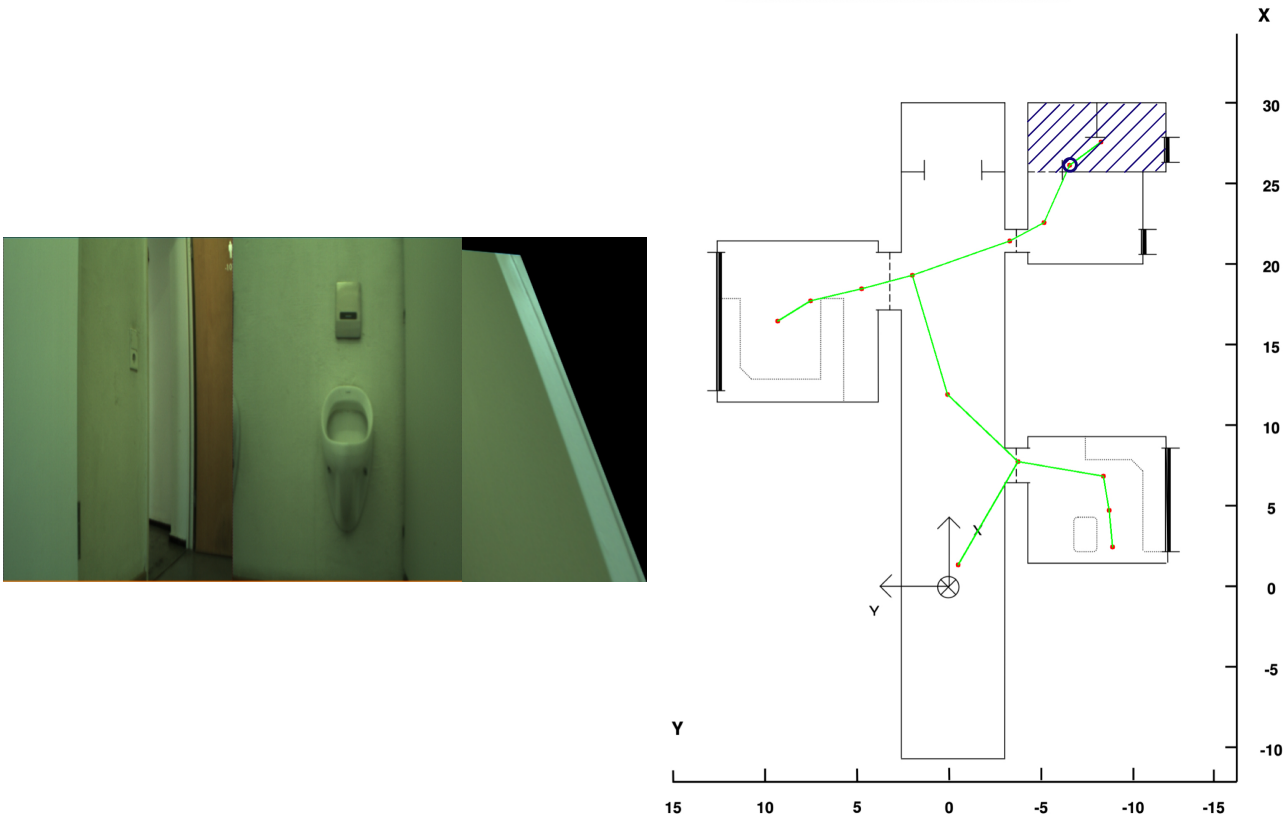
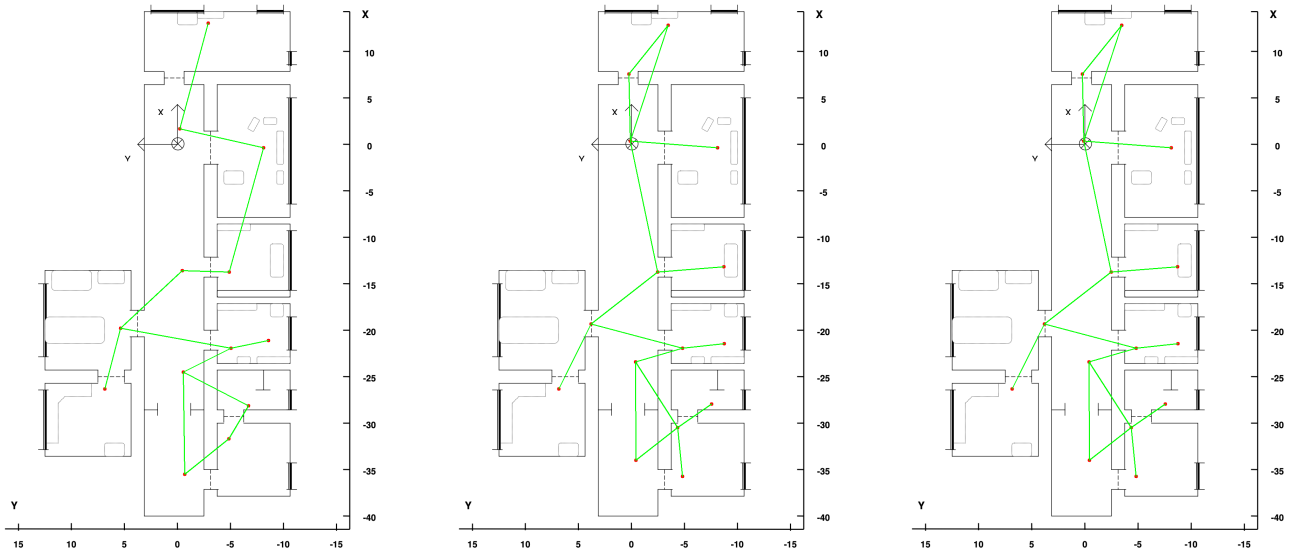
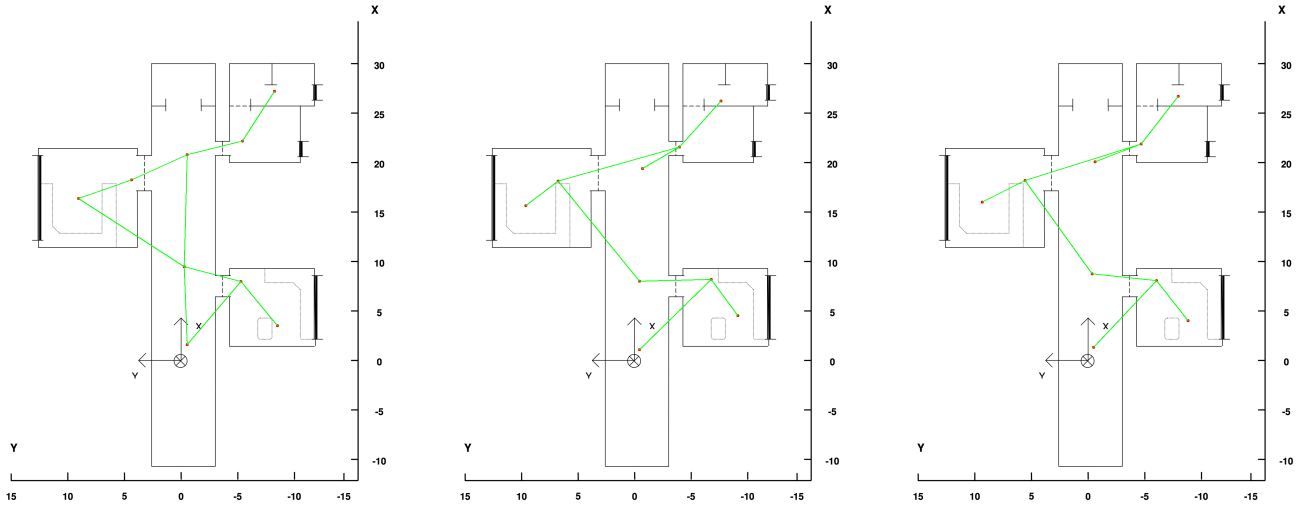


Figure 18. Node generated and associated image when asked: "Go to the restrooms". Blue highlight indicates the selected node and the restrooms position in the map.



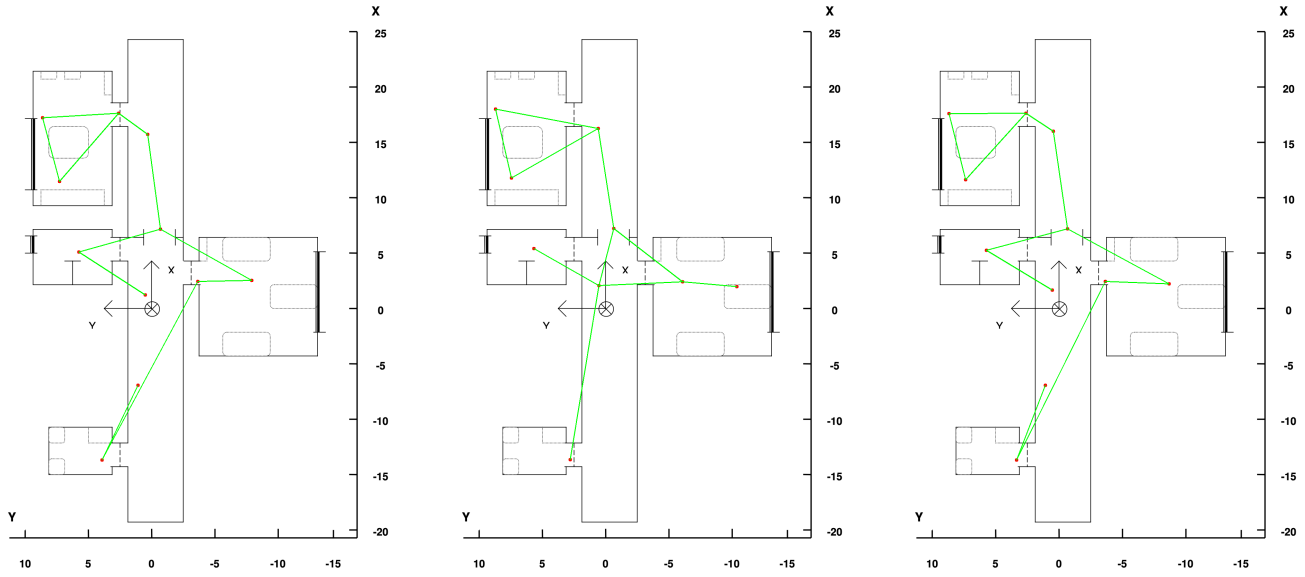
(a) Single trajectory in night conditions. **(b)** Single trajectory in sunny conditions. **(c)** Aligned map.

Figure 19. Freiburg A. Mapping results with CNN extractor.



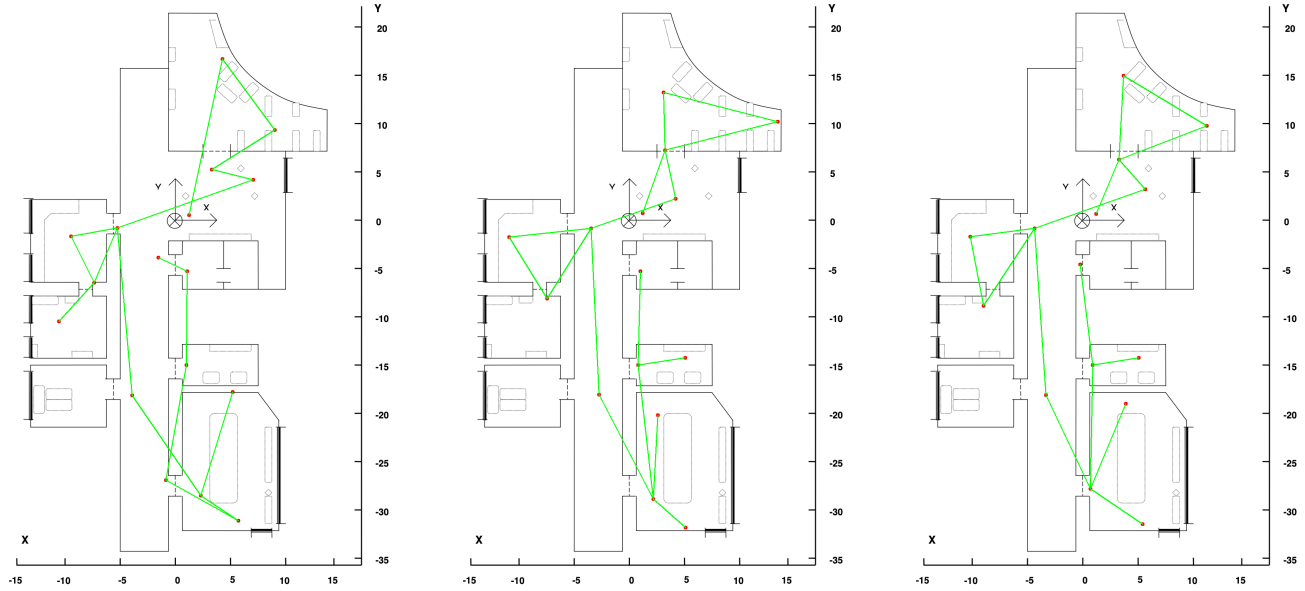
(a) Single trajectory in cloudy conditions. **(b)** Single trajectory in sunny conditions. **(c)** Aligned map.

Figure 20. Freiburg Ext. Mapping results with CNN extractor.



(a) Single trajectory in cloudy conditions. **(b)** Single trajectory in sunny conditions. **(c)** Aligned map.

Figure 21. Saarbrücken Ext. Mapping results with CNN extractor.



(a) Single trajectory in cloudy conditions. **(b)** Single trajectory in night conditions. **(c)** Aligned map.

Figure 22. Saarbrücken A. Mapping results with stricter valley detection.