ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

COMPUTER SCIENCE AND ENGINEERING

Ciclo 38

Settore Concorsuale: 09/H1 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

Settore Scientifico Disciplinare: ING-INF/05 - SISTEMI DI ELABORAZIONE DELLE INFORMAZIONI

ROBOT LOCALIZATION AND MAPPING IN COMPLEX SCENARIOS

Presentata da: Matteo Scucchia

Coordinatore Dottorato

Paola Salomoni

Supervisore

Davide Maltoni

Co-supervisore

Annalisa Franco

Esame finale anno 2026

ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

ALMA MATER STUDIORUM
UNIVERSITY OF BOLOGNA

Dottorato di Ricerca in
Computer Science and Engineering
XXXVIII Ciclo

Robot Localization and Mapping in Complex Scenarios

Presentata da
Matteo Scucchia

Supervisore
Prof. Davide Maltoni

Cosupervisore
Prof.ssa Annalisa Franco

Coordinatrice Dottorato
Prof.ssa Paola Salomoni

Settore Scientifico Disciplinare
ING-INF/05

Esame finale
2026

Abstract

Humans possess an exceptional ability to navigate and map their surroundings while continuously adapting to new environments. We effortlessly integrate prior knowledge with new sensory information, refining our spatial understanding over time. One of the long-standing challenges in robotics and Artificial Intelligence (AI) is developing a long-term *Simultaneous Localization And Mapping* (SLAM) system that can autonomously construct and update maps while adapting to dynamic and previously unseen environments.

Despite significant progress in SLAM research, current state-of-the-art SLAM systems struggle with dynamic environments. Most SLAM algorithms are designed for static or predefined environments, and exhibit a lack of robustness when exposed to evolving or previously unseen conditions. Our best algorithms are still not robust enough to handle the complexity of a dynamic world, where factors such as lighting changes, moving objects, and environmental variations constantly challenge perception and mapping. These systems struggle to adapt to such changes over time, often leading to failures in localization and mapping consistency. Moreover, scalability for long-term navigation remains a challenge, as there are no definitive strategies for managing extremely large maps generated over hours or even days of navigation. Consequently, a true lifelong SLAM framework, capable of continuously adapting and updating its knowledge in large-scale environments has yet to be realized, leaving long-term autonomy in real-world scenarios an open challenge.

In this dissertation, we focus on the integration of SLAM and continual learning to improve long-term localization and navigation in large-scale environments. Building on recent advances in machine learning and deep architectures for AI, we propose a framework that enables SLAM systems to continuously update their knowledge of the environment, adapting to dynamic and evolving scenarios, with a memory management mechanism that allows them to handle extensive navigations. Through this approach, we provide a foundation for more autonomous and reliable navigation

in real-world scenarios.

Acknowledgements

I am deeply grateful for the opportunity to pursue research in the fields of SLAM and Artificial Intelligence, areas that have become increasingly dynamic and transformative in recent years. Working at the intersection of robotics and intelligent systems has been both challenging and inspiring, offering a unique perspective on how theory and application converge in real-world scenarios. It has been a great privilege to explore these topics at a time when they are actively shaping the future of technology and scientific discovery.

I express my heartfelt gratitude to my supervisor, Professor Davide Maltoni, for his unwavering guidance, insightful feedback, and constant encouragement throughout this journey. His expertise, dedication, and support have been instrumental in not only shaping my research but also fostering my growth as a scientist.

I am also grateful to my colleagues and collaborators at Cesena BioLab — Professor Matteo Ferrara, Annalisa Franco and Guido Borghi — as well as my fellow researchers, Lorenzo Pellegrini and Nicolò Di Domenico. Their stimulating discussions, generous advice, and ongoing support made the research environment both engaging and enjoyable.

A special thanks goes to Professor Giovanni Pau for the opportunity to join the Technology Innovation Institute in Abu Dhabi, enabling me to contribute meaningfully while learning from an exceptional team. I am also grateful to my colleagues there — Michael, Ivo, Arun, Omar, Ginevra, Alberto, Andrea, Luca, Davide, Fran, Leo, Andrea, and Alex — for welcoming me so warmly, trusting me, and sharing their knowledge generously.

I am deeply grateful to my family and friends for their patience, understanding, and encouragement, which have sustained me through the highs and lows of this challenging journey.

I would also like to thank my friends from the scouts and my gym community. Their friendship, encouragement and countless shared experiences outside the lab have brought balance, laughter, and inspiration into my life, making this journey more joyful and meaningful.

I would like to take a moment to acknowledge myself and this entire experience. Beyond technical skills, this journey has taught me invaluable lessons in character, resilience, willpower, and perseverance. These qualities, developed through both challenges and achievements, have shaped me profoundly and will continue to guide me well beyond the scope of my research.

Finally, I gratefully acknowledge the funding and resources provided by the University of Bologna, without which this research would not have been possible.

Contents

Abstract **i**

Acknowledgements **iii**

1 Introduction **1**
1.1 Motivation . . . 2
1.1.1 Robustness . . . 2
1.1.2 Scalability . . . 2
1.2 Contribution . . . 4

2 Background and Related Works **7**
2.1 Deep Learning . . . 8
2.1.1 Neural Network Architectures . . . 9
2.1.2 Continual Learning . . . 12
2.2 Introduction to SLAM . . . 15
2.2.1 A brief history . . . 15
2.2.2 Applications . . . 17
2.3 Localization . . . 18
2.3.1 Odometry . . . 18
2.3.2 Place recognition . . . 20
2.3.3 Loop closure Detection and Relocalization . . . 21
2.4 Mapping . . . 23
2.4.1 Map management techniques . . . 24
2.5 Continual SLAM . . . 26
2.6 Mathematical foundation of SLAM . . . 27
2.6.1 Problem definition . . . 29
2.6.2 Proposed solutions . . . 29
2.7 Metrics and Benchmark for SLAM . . . 33
2.7.1 Traditional SLAM metrics . . . 33
2.7.2 Lifelong SLAM metrics . . . 36

2.7.3 Loop closure detection metrics . . . 37

3 Region Prediction for Robot Localization 41
3.1 The fundamental idea . . . 41
3.2 Method overview . . . 43
3.3 Dynamic map clustering . . . 45
3.4 Deep neural network . . . 47
3.5 Temporal Filter . . . 49
3.6 Integration into RTAB-Map . . . 50
3.7 Experiments design . . . 51
3.7.1 Prediction Accuracy . . . 54
3.7.2 Loop Closure Detection . . . 55
3.7.3 Pose estimation accuracy . . . 57
3.8 Conclusions . . . 58

4 Continual Learning of Regions 61
4.1 Deal with Continual Learning . . . 62
4.2 The Proposed Continual SLAM . . . 62
4.2.1 High-level Overview . . . 62
4.2.2 Region Prediction . . . 63
4.3 Experiments Design . . . 66
4.3.1 Datasets . . . 67
4.3.2 Comparisons . . . 68
4.4 E1 experiments . . . 70
4.4.1 Evaluation Workflows . . . 70
4.4.2 Results . . . 71
4.5 E2 experiments . . . 74
4.5.1 Results on RTAB-Map . . . 74
4.5.2 Results on ORB-SLAM3 . . . 78
4.5.3 Results on DROID-SLAM and DPVO . . . 80
4.6 Additional experiments . . . 81
4.6.1 Replay Memory . . . 81
4.6.2 Clustering Granularity . . . 82
4.7 Conclusions and Future Work . . . 82

5 A Synthetic Dataset for Navigation 85
5.1 Background . . . 87
5.1.1 Type of data . . . 87

5.1.2 Tasks . . . 89
5.2 Related Works . . . 90
5.2.1 Place Recognition . . . 90
5.2.2 SLAM . . . 90
5.2.3 Other GTA V datasets . . . 90
5.3 How to capture data from GTA V . . . 91
5.4 Acquired dataset . . . 94
5.5 VPR dataset . . . 95
5.6 Experiments . . . 98
5.6.1 Visual Place Recognition . . . 99
5.6.2 SLAM . . . 104
5.7 Conclusions and Future Work . . . 106

6 Contribution to the community **107**
6.1 The region package . . . 108
6.1.1 Clustering package . . . 108
6.1.2 Datasets package . . . 109
6.1.3 Losses package . . . 110
6.1.4 Models package . . . 110
6.1.5 Samplers package . . . 111
6.1.6 Storage_policy package . . . 113
6.2 RTAB-Map modifications . . . 114
6.2.1 Database modifications . . . 115
6.2.2 Main cycle modifications . . . 116
6.2.3 RTAB-Map GUI . . . 117

7 Conclusions **123**
7.1 Problems addressed and contributions . . . 123
7.2 Broader impact and future directions . . . 124

A Additional Material **125**

List of Figures

2.1 An artificial neuron receives as input a vector $[x_1, ..., x_n]$ (originating either from raw data or from the outputs of other neurons). Each input x_i is multiplied by its associated weight w_i, and the resulting products are summed to form a linear combination. A further parameter, known as the bias (denoted as $w_0(t)$ in the figure), can also be included. The obtained value is then passed through a non-linear activation function f, such as ReLU, which produces the neuron's output. 9

2.2 Structure of a multilayer perceptron (MLP), consisting of an input layer, two hidden layers of interconnected neurons, and an output layer that produces the final prediction. 10

2.3 Illustration of a Convolutional Neural Network (CNN) based on the VGG architecture [8], composed of successive convolutional and pooling layers followed by fully connected layers that produce the final output. 11

2.4 Representation of a Recurrent Neural Network (RNN). The "columnar" structure shows how the same cell is repeatedly applied across time steps. At each step, the hidden state h carries contextual information from previous inputs, allowing the network to model temporal dependencies in sequential data. 12

2.5 Overview of the original Transformer architecture, highlighting the encoder-decoder structure. The encoder processes the input sequence through stacked self-attention and feed-forward layers, while the decoder generates the output sequence by attending to both previous outputs and the encoder's representations. 13

2.6 Example of 2D map computed by a common SLAM algorithm [21]. The red line identifies the path of the robot. 16

2.7 Example loop closure diagram. The robot's estimated trajectory is represented by the black line, and a subset of robot poses are represented by triangles. The stars represent recognizable features that may produce loop closures. On the left, the loop is detected and a constraint is added to the optimization problem representing the fact that the feature (star) must be in the same place in the global frame during its original and subsequent observations. On the right, the features are not recognized as the same, and the accumulated error in location estimate persists [47]. 22
2.8 Different types of maps: 2D grid map, 2D topological map, 3D point clouds and 3D meshes [88] (from left to right, top to bottom). 24
2.9 RTAB-Map transfer-retrieval process and time complexity. As the number of nodes reaches the maximum VM size, the time complexity does not further increase. 25
2.10 The essential SLAM problem. In the image the symbol k stands for the time instant t. 28
2.11 On the left, (a) depicts the online SLAM process, in which only the current state $\mathbf{x}_t$ and the map $\mathbf{m}$ are estimated, on the right, (b) represents the full SLAM problem, which maintains a joint estimate of all past states $\mathbf{X}_{0:t}$ and the map $\mathbf{m}$, allowing for global consistency at the cost of higher computational demand. 30
2.12 Absolute trajectory error. The estimated trajectory is first aligned with the ground truth. 35
2.13 Relative pose error. In this case, it is not necessary to align the estimated trajectory with the true one. 35
2.14 Precision–Recall metric. 39

3.1 On the left, (a) a photo of the original Eiffel Tower in Paris, on the right, (b) shows a photo of a replica in Las Vegas. 42
3.2 An overview of the system we proposed in [3]. During the first exploration of the whole map, map clustering (1) is performed to assign nodes to regions. At the end of the exploration, the region labels become the prediction targets of a deep neural network (2) trained to associate the images corresponding to the nodes with their respective regions. After training, the network is frozen and used during subsequent explorations of the same environment, operating in inference mode (3) to predict regions and filter nodes (4), thus reducing LCD computational complexity. 45

3.3 Map clustering using the same algorithm, without reassignment (a) and with reassignment (b). Different colors indicate different regions. The maps are created from the exploration of the Week1 sequence from the USyd campus dataset [115]. In (a), revisiting the same road (top-right portion of the map) results in two linearly shaped, nearly parallel clusters, as node reassignment is not performed. In contrast, (b) shows a single compact cluster representing that area. 48
3.4 The architecture of the neural network used for region prediction. . . 48
3.5 Two distinct corridors (from our Campus dataset) whose visual appearance is very similar. In such a case, our model should predict both regions with high confidence. 49
3.6 Model predictions on the sequence 00 of the KITTI odometry dataset [119]. The figure highlights the top-3 predicted regions from the network when the robot (indicated by the blue dot) revisits a known location. 52
3.7 The Xaxxon autocrawler robot used to acquire the Campus dataset. . 53
3.8 On the left, (a) depicts an acquisition from the first corridor sequence of OpenLoris, captured under good lighting conditions. On the right, (b) shows the same corridor in the fourth sequence, with artificial illumination turned off. 55
3.9 Absolute trajectory error (ATE) on sequence 09 of the KITTI odometry dataset. On the left, the RTAB-Map baseline does not detect any loop closure, on the right, our method detects a loop closure, reducing the ATE visibly. 58

4.1 A functional schema of the proposed approach. f_Θ is the region prediction model that can be decomposed in the modules $c_{\Theta_c} \circ csf_{\Theta_{csf}} \circ llf_{\Theta_{llf}}$ (see Eq. 4.2) where: c_{Θ_c} is the classification head; $csf_{\Theta_{csf}}$ is the class-specific feature extraction module, and, $llf_{\Theta_{llf}}$ is the low-level feature extraction module. During the main SLAM process (left side of the figure) f_Θ is used to retrieve in the working memory (WM) nodes that could be the target of LCD; dynamic clustering is then performed and representative data are stored in the current experience E_c for the next model update. When a model update is triggered (right side of the figure) data from the E_c are mixed with replay data at the latent representation level and the modules c_{Θ_c} and $csf_{\Theta_{csf}}$ are fine-tuned. $llf_{\Theta_{llf}}$ is decorated with a snowflake in the figure to denote that its weights are kept frozen. A detailed step-by-step description of the workflow in this figure is provided in the main text of Section 4.2.1. 64

4.2 Latent replay architecture and workflow: the latent replay layer is the last of llf. A subset of the activations computed by this layer is stored in a fixed capacity replay buffer and later replayed together with data from new experiences to control forgetting. 65

4.3 Two example frames from each dataset. Images (a) from the USyd dataset illustrate an outdoor environment featuring people and vehicles. Images (b) from the St. Lucia dataset depict an urban environment under varying lighting conditions. Images (c) from the OpenLoris dataset (corridor sequences) emphasize the presence of occlusions, such as people, and differing lighting conditions. Lastly, images (d) from the KITTI dataset showcase diverse urban environments. . . 69

4.4 Map clustering results from the training sequences in E1 experiments. Each color denotes a different region. 71

4.5 Top-1 region classification accuracy in E1 experiments (full test set evaluation). The dashed vertical lines identify the boundaries of the training sequences. The faded color highlights the gap between the minimum and the maximum accuracy on the test sequences. 73

4.6 Top-3 region classification accuracy in E1 experiments (seen so far evaluation). The dashed vertical lines identify the boundaries of the training sequences. The faded color highlights the gap between the minimum and the maximum accuracy on the test sequences. 73

4.7 The map graphs created by the three versions of RTAB-Map at the end of the 12 USyd sequences. 75
4.8 WM size of RTAB-Map during exploration of 12 USyd sequences. . . 76
4.9 Comparison between the node processing time of RTAB-Map (U) and our version. The vertical dashed lines identify the sequence boundaries, while the red horizontal dashed line denotes a real-time constraint, here set to 1 second. The orange line is a rolling average of the node processing time, computed over a window of 100 data points. 76
4.10 Split-down of the node processing time in the two most computing-intensive phases: LCD and Map Optimization. 78
4.11 (a) The map created by ORB-SLAM3 on a single USyd sequence, with no loop closures detected when the robot navigates the same places in the opposite direction. (b) The map path created by ORB-SLAM3 on the 8 sequences used in the experiment. Because of false closures, loss tracking, and inaccurate map merge, the overall result is significantly worse than that obtained with RTAB-Map. 81
4.12 The effect of label updates of reassigned nodes in the replay memory. No replay denotes the case where replay memory is not used. 82
4.13 USyd and St. Lucia maps clustered using the three clustering settings C_{std}, C_{large}, and C_{small}. 83
4.14 Top-3 accuracy on USyd and St. Lucia in the seen so far evaluation under the three clustering settings C_{std}, C_{large}, and C_{small}. 83

5.1 On the left, (a) shows a sparse depth map, while on the right, (b) is the corresponding dense depth. 88
5.2 Three-quarter view of a point cloud acquired through a 3D laser from the KITTI dataset [119]. Color fades from blue to red, indicating greater proximity to the sensor at blue and greater distance at red. . 88
5.3 Image with bounding boxes from the PreSIL dataset [131], acquired from GTA V. The blue rectangle are the bounding boxes for the vehicles, while the red ones are for the pedestrians. 88
5.4 On the left, (a) shows an RGB image, while on the right (b) shows the corresponding segmented image. The images are from the GTA5 dataset [132], acquired from GTA V. Each pixel is colored according to the category of the object to which it belongs. 89
5.5 Data acquisition process. 93

5.6 The six paths acquired from GTA V. On the left, (a) highlights the scale and the legend to provide a clear visualization of the paths, while on the center, (b) reports the same paths on the GTA map. . . 96
5.7 RGB-D image extracted from the Legion Square sequence. On the left, (a) the RGB image, on the right, (b) the corresponding post-processed depth. It is worth noting the high quality of the depth image. 96
5.8 Images associated to the same place by the automatic frame selection algorithm. 99
5.9 Example of images from the three datasets, captured under different light/weather conditions. 101
5.10 Training scheme for the VPR experiments. 102
5.11 Results of the VPR experiments. On the top, (a) shows the recall at different k using day images of Tokyo 24/7 as query and night images as database, while, on the bottom, (b) reports the results of the opposite setting, using night images as query and day images as database. 103
5.12 Paths estimated by RTAB-Map compared to the ground truth. In order, (a) is LegionSquare, (b) is Stadium, (c) is VinewoodA, (d) is VinewoodB, (e) is WestEclipseBoulevard and (f) is the LongRing. The red crosses on the LongRing path highlight where the tracking is lost. 105

6.1 The `region` package, the main package added to RTAB-Map. It contains the other sub-packages and the two main classes `TrainThread` and `ContinualLearning`. 109
6.2 The `clustering` package, responsible for the region creation and the dynamic clustering. 109
6.3 The `dataset` package, responsible for the dataset types used in the training process. 110
6.4 The `losses` package, responsible for the loss functions used in the training process. 111
6.5 The `model` package, responsible for the neural network building blocks. 112
6.6 The `samplers` package, responsible for the sampling of the batches during training. 113
6.7 The `storage_policy` package, responsible for the storage policies of the replay memory. 113
6.8 The RTAB-Map classes modified to incorporate our methodology. . . 115

6.9 The RTAB-Map database tables modified to incorporate our methodology. 116
6.10 The main RTAB-Map class methods, modified to incorporate our methodology. 118
6.11 RTAB-Map GUI showing a part of the exploration of the sequence 00 of the KITTI odometry dataset. 120
6.12 RTAB-Map GUI showing a loop closure detected during the exploration of the sequence 00 of the KITTI odometry dataset. 120
6.13 RTAB-Map GUI showing a part of the exploration of the sequence Week3 of the USyd campus dataset. 121
6.14 RTAB-Map GUI showing a relocalization detected during the exploration of the sequence Week3 of the USyd campus dataset. 121

List of Tables

3.1 Top-1 and top-3 region prediction accuracy. The numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y. 54

3.2 Loop closure detection results. For Campus and OpenLoris, the numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y. 56

3.3 Relocalization results. The numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y. 57

3.4 Precision-Recall on loop closure detection. For Campus and OpenLoris, the numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y. 57

3.5 Precision-Recall on relocalization. The numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y. 57

3.6 Absolute trajectory errors results. Both the baseline and our method have to be considered with the WM size limited to 50 nodes. For the baseline, the ATE is higher because fewer loop closures are detected, while for our method it decreases because all loop closures are detected. Since the entire map cannot be corrected due to the limited WM size, results are different from those reported in the original RTAB-Map paper [5]. 58

4.1 Datasets used in the experiments. Total RGB frames denote the number of frames in the sequences used. 68

4.2 Statistics of E2 on the 12 USyd sequences. RTAB-Map is compared in the three versions: RTAB-Map (U = Unbounded), RTAB-Map (L = Time-limited) and our. 75

4.3 Loop Closure (LC) detection results. 77

4.4 Relocalization results. The numbering n-(x,y,...) at the end of each sequence name means the relocalizations in the sequence n are computed after the exploration of the sequences (x,y,...). 77
4.5 Absolute trajectory errors of RTAB-Map (U = Unbounded), RTAB-Map (L = Time-Limited) and our. For the OpenLoris sequence, the ATE is computed after the exploration of the sequences Market1, Market2, and Market3. The ATE values obtained by RTAB-Map (U) on some of the KITTI sequences are different from those reported in the original paper [5] because we used the default values for the RTAB-Map parameters. 78
4.6 ORB-SLAM3 settings and results, averaged over the five runs. For details on single metrics refer to ORB-SLAM3 paper [77]. 80

5.1 The main publicly available place recognition datasets. 91
5.2 The main publicly available SLAM datasets. 92
5.3 Number of frames across the different sequences of our dataset. 95
5.4 Absolute Trajectory Errors for the five sequences run on RTAB-Map. 104

A.1 Settings of E1: training/test sequences and hyperparameters used. . 125
A.2 Settings of E2: training/test sequences and hyperparameters used. . 126

Chapter 1

Introduction

In robotics, Simultaneous Localization and Mapping (SLAM) refers to the problem in which a mobile robot, equipped with one or more sensors, navigates through an unknown environment while incrementally building a map of that environment and simultaneously estimating its own pose within the map [1]. SLAM is widely recognized as a cornerstone of autonomous robotics, as it enables robots to operate effectively in scenarios where prior maps are unavailable, outdated, or unreliable. Accurate and consistent SLAM not only provides the foundation for safe motion and decision-making, but also underpins higher-level capabilities such as path planning, exploration, and long-term autonomous navigation. Reliable SLAM is therefore indispensable in a wide range of applications, including autonomous driving, aerial robotics, service robots, industrial automation, planetary exploration, and operations in GPS-denied environments, where robust navigation and environment understanding are critical.

From a theoretical perspective, the SLAM problem has been extensively studied and many effective solutions have been proposed. Under controlled assumptions, such as static environments, limited spatial scales, and consistent sensing conditions, current SLAM algorithms can achieve high accuracy and robustness. However, these assumptions are rarely satisfied in real-world scenarios. Environments may change over time, objects may be moved or removed, and external factors such as lighting, weather, or seasonal variations can alter the visual or geometric appearance of a place. Such dynamics increase the uncertainty in localization and challenge the consistency of the constructed map. As a result, despite the significant progress made, SLAM remains an open problem in practice, particularly in the context of lifelong or continual SLAM, where the system must operate over extended periods, adapt to environmental changes, and continuously refine its representation of the world.

1.1 Motivation

Despite significant advances in SLAM, several open challenges remain that limit its use in real-world applications, especially if long-term autonomy is required to navigate complex environments. State-of-the-art solutions face two primary challenges: robustness and scalability [2].

1.1.1 Robustness

Robustness refers to the ability of the system to maintain accurate localization and map consistency despite uncertainties, environmental changes, and sensor noise, and it is strictly related to the problem of data association [2]. Most current solutions assume that the environment is static, a condition that clearly does not reflect the real world. A robust SLAM system that can handle a dynamic environment is critical for long-term autonomy. The visual appearance of a place is subject to change over time due to various factors, including day/night lighting variations, seasonal shifts, and human interactions, such as the addition, removal, or rearrangement of objects. Accurately recognizing the same location under these dynamic conditions requires a high degree of robustness, which remains an open challenge. Conversely, an opposing issue known as perceptual aliasing can arise, where distinct locations share similar visual features and are incorrectly identified as the same place. This phenomenon is particularly problematic in structured or repetitive environments, where visually similar regions can lead to incorrect localization.

1.1.2 Scalability

Scalability in SLAM refers to the ability of the system to operate efficiently and accurately as the size and complexity of the environment increase [2]. A scalable SLAM system must effectively manage extensive datasets, high-dimensional sensor inputs, and long-term mapping while maintaining real-time performance and avoiding excessive computational costs. Scalability can be examined along two orthogonal dimensions: time and memory.

Time scalability

Time scalability refers to the system's capability to maintain real-time operation regardless of the size of the mapped environment. This aspect becomes particularly challenging in long-term explorations, where the number of past observations used to recognize the same place can grow to hundreds of thousands or even millions. As the

search space expands, computational requirements increase significantly, breaking the real-time constraint and hindering the practical usability of the system.

A well-established approach to mitigating this issue is the use of keyframe-based representations, where only a subset of frames, those most representative of their surroundings, are retained for processing. Although keyframes help reduce the number of comparisons, their accumulation over extensive explorations can still lead to unbounded growth, making the system unsuitable for lifelong SLAM.

To further enhance time scalability, map management strategies have been proposed, which dynamically select only a relevant subset of the map for long-term localization. These approaches aim to discard parts of the map that are less likely to be involved in localization and loop closure detection, while preserving the critical parts. However, determining optimal strategies for selecting and maintaining such a subset remains an area of active research.

Memory scalability

Memory scalability refers to the system's ability to store and manage an extremely large map, potentially encompassing an entire world-scale environment. Traditional SLAM solutions suffer from unbounded map growth, where all observed landmarks and keyframes are stored indefinitely. As a result, increasing memory requirements eventually exceed the capacity of RAM, making extensive explorations infeasible without efficient memory management techniques.

Several strategies have been proposed to address this challenge. Map compression and pruning techniques selectively discard redundant information while preserving the integrity of the representation. Hierarchical or multi-level representations allow for storing different levels of detail, ensuring that only the necessary data is retrieved when needed.

A promising alternative to explicit map storage is implicit SLAM, where the environment is represented using neural implicit functions rather than explicitly storing point clouds or occupancy grids. This significantly reduces memory requirements by encoding map information into a compact latent space. However, when applied to large-scale environments, implicit representations introduce new challenges, such as efficiently switching between different neural networks as the robot explores new regions. Developing scalable mechanisms for handling multiple implicit maps while ensuring consistency and real-time operations remains an open research problem.

1.2 Contribution

Building upon these open challenges, this thesis introduces a novel approach to address the robustness and scalability limitations of SLAM. The main contribution is the development of a new method for lifelong loop closure detection based on continual learning, which enables reliable relocalization in dynamic and large-scale environments. By integrating continual learning into loop closure detection, the proposed system supports long-term operation on maps of unprecedented size, while adapting to environmental changes and mitigating catastrophic forgetting. This contribution directly tackles one of the most critical bottlenecks in SLAM, namely maintaining accurate localization and map consistency over extended deployments. The proposed approach is situated within the loop closure domain and with respect to the existing literature, it introduces a robust method to reduce the search space of known places, enabling scalability to large-scale maps while keeping the time complexity bounded. While other learning-based methods primarily focus on improving matching performance, typically through neural networks for place recognition, our approach addresses an orthogonal dimension, namely the avoidance of brute-force matching against all known places.

The work of my Ph.D can be summarized as follows:

- **Region-based loop closure detection:** developed a novel method validated in offline scenarios [3] and published at ROBOVIS 2024 conference. This work provided the first proof of concept for the approach, without the added complexity of continual learning (Chapter 3).

- **Continual learning for online SLAM:** extended the method to online scenarios with continual learning, employing strategies such as latent replay, class-balanced storage, and incremental layers to mitigate catastrophic forgetting [4]. The system was integrated into RTAB-Map [5], resulting in a journal paper in *IEEE Transactions on Robotics*. This contribution addresses a key bottleneck in SLAM: maintaining accurate localization and map consistency over extended deployments (Chapter 4).

- **Open-source framework:** released a reproducible research framework, including the modified RTAB-Map and full experimental pipeline, to facilitate further research in continual SLAM (Chapter 6).

- **Synthetic dataset for long-term SLAM:** collected and published a GTA V-based dataset for evaluating robotic navigation tasks, accepted at ICRCV 2025 conference. This work demonstrated that synthetic data, captured in

controlled and reproducible environments, can effectively complement or replace real data at significantly lower cost (Chapter 5).

- **Generative AI design principles:** contributed a book chapter on generative AI in the Springer volume *The Cyber-Creativity Process* (2025) [6], expanding the scope of this research to broader AI design methodologies. Part of the concepts of the book chapter are reported in Chapter 2, as an introduction to deep learning.

The thesis is organized as follows: Chapter 2 provides an in-depth discussion of SLAM and related work; the central chapters 3–6, present the main contributions introduced above; finally, Chapter 7 summarizes our findings and outlines possible directions for future research.

Chapter 2

Background and Related Works

"Encoded in the large, highly evolved sensory and motor portions of the human brain is a billion years of experience about the nature of the world and how to survive in it. The deliberate process we call reasoning is, I believe, the thinnest veneer of human thought, effective only because it is supported by this much older and much more powerful, though usually unconscious, sensorimotor knowledge. We are all prodigious olympians in perceptual and motor areas, so good that we make the difficult look easy. Abstract thought, though, is a new trick, perhaps less than 100 thousand years old. We have not yet mastered it. It is not all that intrinsically difficult; it just seems so when we do it."

– Hans Moravec, 1988

Humans have an extraordinary ability to perceive and understand their surroundings. This ability underpins our capacity to plan complex tasks for long-term goals in the real world. Indeed, not only do we have a profound knowledge of the environment in which we live, but we can also update our knowledge and adapt to changing environmental conditions, adjusting our plan to the new configuration.

In recent years, Artificial Intelligence (AI) has experienced a new boom with Large Language Models (LLMs) capable of generating human-like texts and thus automating complex tasks for human operators, even demonstrating exceptional capabilities in software programming and hard sciences such as mathematics and physics. However, our most intelligent system is far from having a deep understanding of the real world and its perception and navigation capabilities are not even comparable to those of human beings, nor to those of five-year-old children or animals. This may seem counterintuitive: how can a computer beat the world chess champion, solve complex maths and physics problems, generate human-like texts or

even images and videos of complex scenarios and not be able to do something as simple as perceive and act in the environment as humans or animals can effortlessly do? In the 1980s, computer scientist Hans Moravec noticed that, contrary to traditional assumptions, reasoning requires very little computation, but sensorimotor and perception skills require enormous computational resources. This assumption has gone down in history as Moravec's paradox. The most accredited reason for this lies in biology: all human skills arise from biological mechanisms shaped by natural selection. Over millions of years, this process has preserved and refined beneficial adaptations; the older a skill, the more optimized its design is likely to be. In contrast, abstract thinking emerged relatively recently, so its implementation may still be inefficient. This is why really complex tasks *appear* effortless and natural to us, but in reality they are not.

For more than thirty years, researchers have been trying to recreate in robots the incredible animal perception and navigation capabilities, which are the basis for long-term autonomy. In the community, this problem takes the name of Simultaneous Localization And Mapping (SLAM). Without a deep knowledge of the environment and the physical rules that govern our world, it is hard to achieve true long-term robotic autonomy, let alone human-like planning in the real world.

In this chapter, we provide the foundational background on Artificial Intelligence techniques relevant to SLAM and review more than three decades of research in the field.

2.1 Deep Learning

Deep Learning is a subfield of Artificial Intelligence that focuses on designing algorithms capable of learning complex patterns from data, particularly in tasks for which explicit, hand-crafted solutions are not available. Its foundation lies in the artificial neuron, which was first conceptualized in the 1940s by Warren McCulloch and Walter Pitts as simplified models of biological neurons. An artificial neuron receives a set of inputs, computes a weighted sum, applies a non-linear activation function, and produces an output that can be propagated to subsequent neurons (Fig. 2.1). Neurons are connected by artificial synapses, with a corresponding weight, to weigh the inputs differently. Artificial Neural Networks (ANNs) are obtained by arranging neurons into layers, so the networks can learn hierarchical representations, enabling the approximation of complex, non-linear functions [7]. To learn patterns from data,

ANNs undergo a process called *training*. Training can be categorized into three main types:

- **Supervised learning**: data are paired with labels representing the desired outputs, and the network learns to map inputs to these labels.
- **Unsupervised learning**: data have no labels, so the network must discover intrinsic patterns, structures, or representations within the input data.
- **Self-supervised learning**: data do not have explicit labels, but supervisory signals can be derived from the data itself, e.g., predicting missing parts of the input or temporal relationships in sequences.

During training, ANNs optimize a *loss function*, which quantifies the difference between the network's predictions and the expected targets (or the desired pattern in self-supervised/unsupervised settings). The network adjusts its weights iteratively, typically using gradient-based optimization methods, to minimize this loss and improve its performance. Once training is complete, ANNs can be used to make predictions on new, previously unseen data.

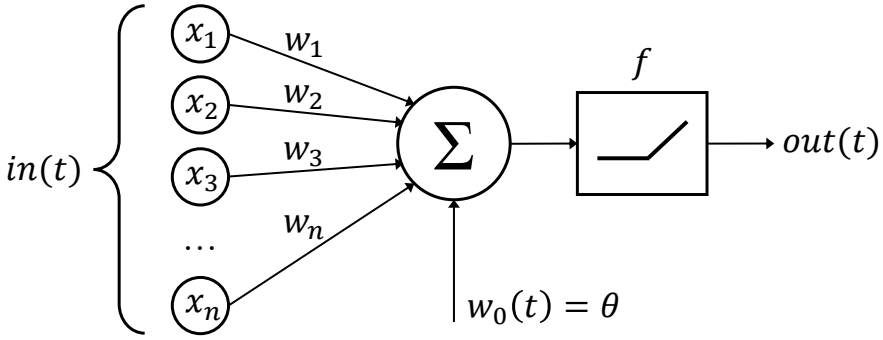


Figure 2.1: An artificial neuron receives as input a vector $[x_1, ..., x_n]$ (originating either from raw data or from the outputs of other neurons). Each input x_i is multiplied by its associated weight w_i, and the resulting products are summed to form a linear combination. A further parameter, known as the bias (denoted as $w_0(t)$ in the figure), can also be included. The obtained value is then passed through a non-linear activation function f, such as ReLU, which produces the neuron's output.

2.1.1 Neural Network Architectures

Artificial neurons can be organized into different network architectures depending on the type of data to process and the tasks to solve. Stacking many layers (potentially hundreds or even thousands) gives rise to deep architectures, which is where

the term "Deep Learning" originates. Four of the most widely used NN architectures are Multi-Layer Perceptrons (MLPs), Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers.

Multilayer Perceptrons (MLPs)

MLPs are the simplest form of feedforward neural networks and serve as the foundational building block of many other architectures [7]. They consist of an input layer, one or more fully connected hidden layers, and an output layer (Fig. 2.2). Each neuron in a layer is connected to every neuron in the subsequent layer, and the output of each neuron is computed as a weighted sum of its inputs followed by a non-linear activation function. MLPs are universal function approximators, meaning that, given sufficient neurons and layers, they can approximate any continuous function. However, they do not inherently exploit spatial or temporal structure, which limits their efficiency for high-dimensional data such as images or sequential data.

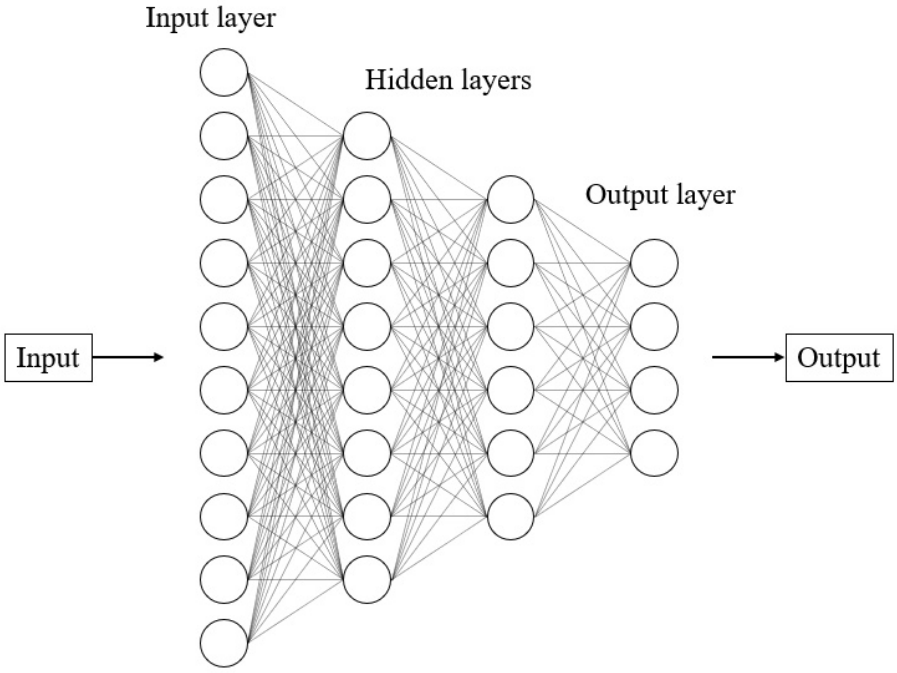


Figure 2.2: Structure of a multilayer perceptron (MLP), consisting of an input layer, two hidden layers of interconnected neurons, and an output layer that produces the final prediction.

Convolutional Neural Networks (CNNs)

CNNs are specifically designed to handle data with spatial structures, such as images [7]. They use convolutional filters to extract local features and pooling layers to reduce spatial dimensions while preserving essential information (Fig. 2.3). Early layers capture low-level features like edges and textures, while deeper layers capture higher-level semantic patterns. Weight sharing and spatial invariance make CNNs highly effective for vision tasks, including image classification, object detection, and segmentation.

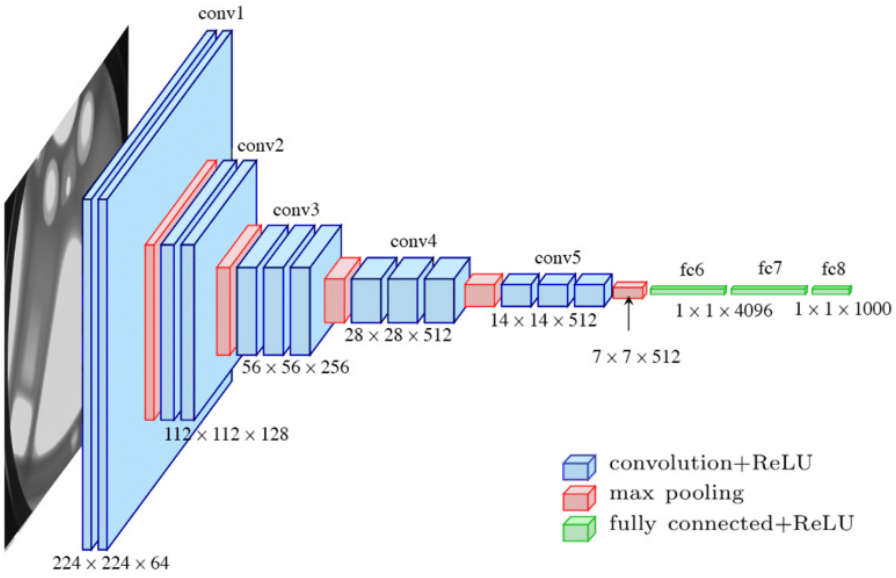


Figure 2.3: Illustration of a Convolutional Neural Network (CNN) based on the VGG architecture [8], composed of successive convolutional and pooling layers followed by fully connected layers that produce the final output.

Recurrent Neural Networks (RNNs)

RNNs are specialized for sequential data, such as time series, speech, or text [7]. They maintain a hidden state that summarizes past inputs, allowing the network to model temporal dependencies (Fig. 2.4). Standard RNNs can struggle with long-term dependencies due to vanishing or exploding gradients, which is addressed by variants such as Long Short-Term Memory (LSTM) networks [9] and Gated Recurrent Units (GRUs) [10]. These gating mechanisms regulate information flow and enable the network to capture longer-range dependencies. RNNs are widely used in applications like language modeling, machine translation, and speech recognition.

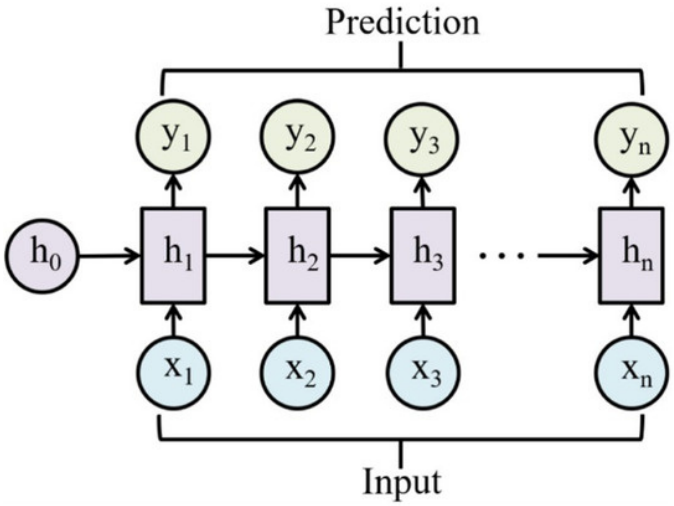


Figure 2.4: Representation of a Recurrent Neural Network (RNN). The "columnar" structure shows how the same cell is repeatedly applied across time steps. At each step, the hidden state h carries contextual information from previous inputs, allowing the network to model temporal dependencies in sequential data.

Transformers

Transformers eliminate the need for recurrence and rely on self-attention mechanisms, which allow the model to assess the relevance of all input elements simultaneously [11]. This facilitates the modeling of both short- and long-range dependencies in parallel. Transformers typically consist of encoder-decoder stacks for sequence-to-sequence tasks or stacked encoder blocks for classification tasks (Fig. 2.5). Positional encodings provide information about sequence order. Transformers underpin state-of-the-art models in natural language processing, such as BERT [12] and GPT [13], and have also been adapted for vision (Vision Transformers, ViT [14]) and multi-modal learning.

2.1.2 Continual Learning

Continual learning [15], also called lifelong or incremental learning, is a recent branch of deep learning and artificial intelligence that enables models and systems to adapt dynamically to new data while preserving previously acquired knowledge. Traditional deep learning approaches often assume a static dataset and rely on batch training, leading to inefficiencies and catastrophic forgetting when models are exposed to evolving data distributions. In contrast, continual learning allows systems to learn progressively and update their knowledge, making it particularly valuable for real-world applications where data is non-stationary but becomes available over time, such as in robotics.

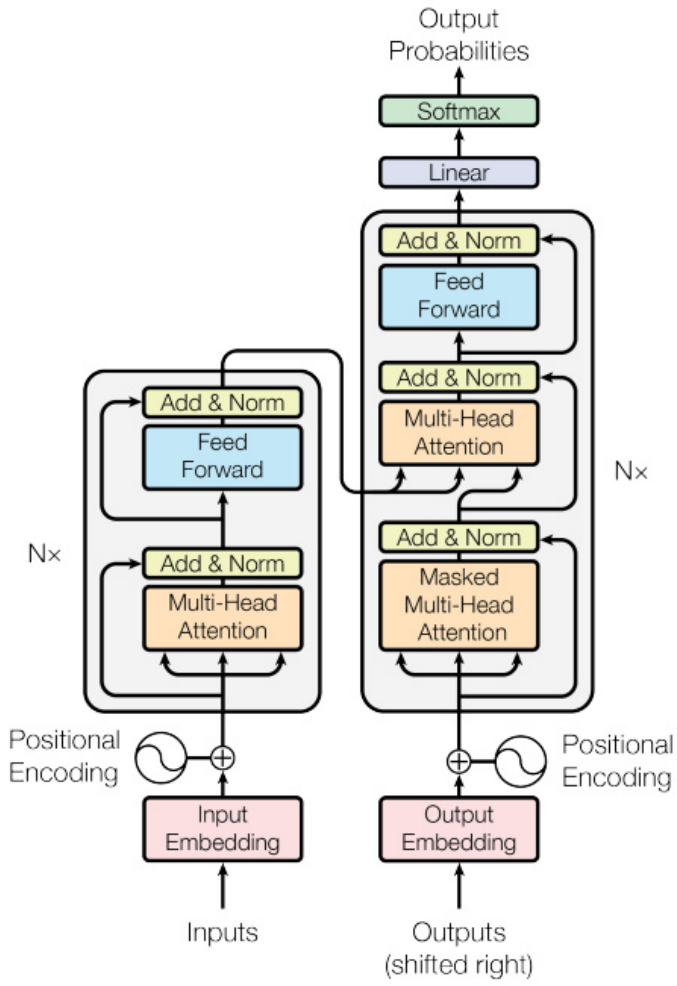


Figure 2.5: Overview of the original Transformer architecture, highlighting the encoder-decoder structure. The encoder processes the input sequence through stacked self-attention and feed-forward layers, while the decoder generates the output sequence by attending to both previous outputs and the encoder's representations.

In this context, each bunch of new data is called *experience*, and a new training stage is triggered when a new experience becomes available. In traditional CL scenarios, the model trains solely on the current experience since re-training from scratch on the cumulative data seen so far takes too long for real-world, real-time applications. Unfortunately, this easily leads to a condition known as *catastrophic forgetting*, where the model adapts to new data but loses previously acquired knowledge.

In mathematical terms, a continual learning classification problem consists of a sequence of n_e experiences, denoted as E_i, where each contains a subset of data that is accessible only during the corresponding training step. Formally, the problem can

be defined as:

$$CL = \{E_1, E_2, ..., E_{n_e}\}, \tag{2.1}$$

where each experience E_i comprises input data E_i^x and their corresponding labels E_i^y:

$$E_i = (E_i^x, E_i^y), E_i^x = \{x_1^i, x_2^i, ..., x_{n_i}^i\}, E_i^y = \{y_1^i, y_2^i, ..., y_{n_i}^i\}. \tag{2.2}$$

Here, E_i^x represents the set of input samples encountered in the i-th experience, while E_i^y denotes the corresponding labels. The number of samples in each experience is given by n_i, which can vary between experiences.

In the CL terminology, classification problems are categorized according to the composition of experiences [16]:

- *New Instances* or *Domain Incremental* (NI): data in E_i are new instances of already known classes: $E_i^y \subseteq \bigcup_{j=0}^{i-1} E_j^y$.
- *New Classes* or *Class Incremental* (NC): data in E_i always belong to new classes: $E_k^y \cap E_i^y = \emptyset,\ i < k$.
- *New Instances and Classes* (NIC): data in E_i may belong to new classes or be instances of already known classes with no constraints on the labels.

These three settings define different levels of complexity in continual learning. While domain-incremental learning focuses on handling distribution shifts within known categories, class-incremental learning requires a model to integrate entirely new concepts while preventing catastrophic forgetting. The NIC scenario, being the most flexible, requires the model to balance both adaptation to novel categories and retention of past knowledge, making it a particularly demanding challenge in the field of lifelong learning.

Given a classification model f parametrized by Θ, train it incrementally (to minimize the loss $\mathcal{L}$) over a sequence of experiences CL:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(f_\Theta(E_i^x), E_i^y) \ \ for\ \ i = 1, ..., n_e \tag{2.3}$$

leads to a progressive forgetting of the data belonging to past experiences.

Key challenges in continual learning include overcoming catastrophic forgetting, ensuring knowledge transfer between tasks, and maintaining computational efficiency. In many classification applications, the classes of the problem are not known a priori, requiring the model to adjust its architecture as new classes are introduced. Furthermore, data within experiences can be highly unbalanced or

even belong to a single class, thereby violating the *independent and identically distributed* assumption underlying stochastic gradient descent (SGD). To address these challenges, researchers have explored various strategies, including experience replay [17, 18], regularization-based approaches [19], and dynamically expandable architectures that accommodate new knowledge while preserving past information [20]. Furthermore, biological inspirations from human learning, such as memory consolidation and synaptic plasticity, have motivated novel techniques in artificial neural networks to enhance adaptability and long-term retention. By advancing theoretical frameworks, designing novel algorithms, and implementing real-world applications, such research can bridge the gap between artificial and biological learning systems.

2.2 Introduction to SLAM

In robotics, SLAM refers to the problem in which a mobile robot equipped with a set of sensors navigates an unknown environment and incrementally builds a map of the environment in real time, simultaneously locating itself on the map under construction [1].

The problem is composed of two simultaneous tasks:

- **Mapping**, the task of building a model of the environment, perceiving it through sensors.
- **Localization**, the task in which the robot has to localize itself on the map.

Figure 2.6 illustrates the SLAM map generated following the exploration of an indoor environment.

SLAM is an inherently difficult problem because these two tasks are strictly interdependent. In fact, to build a map, the robot must know its location, and to know its location it must have a map.

SLAM is formulated as a state estimation problem because sensor measurements are inherently noisy, affecting the whole process by an error that accumulates over time, known as *drift*. Over the past three decades, SLAM has evolved from theoretical constructs to practical applications, becoming integral to technologies such as autonomous vehicles, aerial drones, and consumer robotics.

2.2.1 A brief history

SLAM [1, 22] has emerged as a cornerstone in the field of autonomous robotics, enabling machines to navigate and understand unknown environments without prior

Figure 2.6: Example of 2D map computed by a common SLAM algorithm [21]. The red line identifies the path of the robot.

knowledge. The conceptual underpinnings of SLAM can be traced back to the mid-1980s. In 1986, Smith and Cheeseman [23] introduced pioneering methods to represent and estimate spatial uncertainty, providing a probabilistic framework essential for robotic perception and navigation. Their work laid the groundwork for subsequent research in autonomous mapping and localization. In 1988, another milestone published by Durrant-Whyte [24] expanded the work of Smith and Cheeseman. The acronym "SLAM" was formally introduced during the 1995 International Symposium on Robotics Research. At this symposium, researchers presented the structure of the SLAM problem, discussing its convergence properties, and establishing it as a critical area of study within mobile robotics. The initial approaches to SLAM predominantly utilized the Extended Kalman Filter (EKF) [25]. This method extended the traditional Kalman Filter [26] to accommodate nonlinear systems, making it suitable for real-time robotic applications. The Iterated Extended Kalman Filter (IEKF) [27] extends the EKF by performing multiple iterations of the update step, effectively applying a Gauss–Newton optimization to improve convergence. However, EKF-SLAM faced challenges, particularly concerning computational scalability and inaccuracies arising from linearization, which limited its effectiveness in large-scale or highly dynamic environments [28, 2]. To overcome the limitations of EKF-SLAM, researchers have proposed alternative probabilistic filtering methods. The Unscented Kalman Filter (UKF) [29] avoids explicit linearization by employing

sigma points to more accurately approximate the propagation of uncertainty. Particle filter–based methods [30, 31], such as Monte Carlo Localization [32], further improve robustness by representing the robot's pose with a set of weighted samples, thereby handling strong nonlinearities and multimodal uncertainty more effectively. More recently, other more complex filters have been proposed to increase the accuracy of SLAM systems, such as the Multi-State Constraint Kalman Filter (MSCKF) [33], an adaptation of the EKF to the visual-inertial problem. Graph-based SLAM [34, 35] emerged as an alternative approach, becoming the state of the art. These methods model the SLAM problem as a graph-optimization task, where nodes represent robot poses or landmarks, and edges correspond to spatial constraints derived from sensor observations. Graph-based methods have shown superior scalability and accuracy, especially in large and complex environments [36]. In recent years, deep learning [7] has significantly influenced SLAM, leading to the development of learning-based SLAM systems that complement or replace traditional techniques [37]. Convolutional Neural Networks (CNNs) [7] and Recurrent Neural Networks (RNNs) [7] have been integrated into feature extraction and loop closure detection [38, 39], improving robustness in perceptually ambiguous environments. End-to-end deep learning architectures, such as DeepVO [40] and VINet [41], have been proposed to learn odometry directly from visual inputs. More recently, transformers [11] and self-supervised learning methods have been applied to SLAM to improve feature representation and long-term place recognition. Neural implicit representations, inspired by Neural Radiance Fields (NeRF) [42], have introduced implicit SLAM methods, such as iMAP [43] and NICE-SLAM [44], enabling dense, continuous, and high-fidelity scene reconstruction. These advancements address challenges like environmental dynamics, perceptual aliasing, and multi-sensor fusion, making SLAM more adaptable to real-world, large-scale applications.

2.2.2 Applications

Today, SLAM is a mature technology and therefore has a wide range of applications in various domains, playing a crucial role in enabling autonomous systems to perceive, navigate, and interact with their environment. In robotics, SLAM is fundamental for autonomous mobile robots, self-driving cars, and drones, allowing them to map unknown environments while localizing themselves using sensor data from LiDAR, cameras, and IMUs. In augmented reality (AR) and virtual reality (VR), SLAM enables accurate tracking for AR applications on smartphones and inside-out tracking in VR headsets. Medical applications leverage SLAM for surgical navigation, rehabilitation robotics, and monitoring instruments during minimally invasive

procedures. In space exploration, NASA's Mars rovers and autonomous satellites utilize SLAM for mapping and navigation in extraterrestrial environments. Military and defense applications benefit from SLAM in autonomous reconnaissance vehicles, surveillance drones, and real-time battlefield mapping. The industrial sectors employ SLAM for infrastructure inspection, automated construction mapping, and maintenance of hazardous environments. Underwater exploration relies on SLAM for deep-sea navigation in GPS-denied environments, particularly autonomous underwater vehicles (AUVs). In smart cities, SLAM facilitates urban mapping, traffic monitoring, and infrastructure management. Precision agriculture integrates SLAM into autonomous tractors and drones for efficient crop monitoring and automated harvesting. Furthermore, consumer applications such as robotic vacuum cleaners and future home assistant robots employ SLAM to intelligently navigate indoor environments. As SLAM continues to evolve, its impact extends across various scientific, industrial, and commercial fields, driving advancements in autonomy, perception, and spatial intelligence.

2.3 Localization

Localization refers to the robot's ability to locate itself in the environment. For this purpose, the robot must be able to analyze and associate data acquired from sensors with previous observations, a task known as *data association*. To associate environmental data, the robot typically identifies *landmarks*, real-world points that are easily recognizable and highly discriminating [45]. Data association can be short-term, i.e., with the last observation or observations close in time, or long-term, i.e., with observations acquired long before. The short-term data association is used to compute *odometry* [46], the estimated motion since the last acquisition, while long-term data association is related to *Loop Closure Detection* (LCD) and *Relocalization* [38, 47].

2.3.1 Odometry

The word "odometry" is made up of the Greek words "odos", which means "route" and "metron", which means "measure". It refers to the use of data association to estimate the change in position over time. Based on the sensor, there are several types of odometry [48]:

- **Wheel odometry**, if the motion is estimated by the encoders that measure the wheel revolutions.

- **Visual odometry**, when data are images, typically acquired through a camera.
- **LiDAR odometry**, when data are acquired thanks to the use of lidars.
- **Inertial odometry**, if the mobile robot has inertial sensors such as gyroscope or accelerometer.

There are also other types of odometry, and these types can be combined with each other to improve the accuracy of motion estimation, such as visual-inertial odometry (VIO). Similarly, SLAM has different names depending on the type of sensor used. Visual SLAM uses images; LiDAR SLAM uses lidars, and so on. Odometry and SLAM are not the same thing: odometry can be considered a part of the SLAM process; its purpose is to estimate the motion between two data acquisitions, while SLAM aims to construct a consistent global map, involving also other, more complex, processes.

Deep learning has significantly advanced both Visual-Inertial Odometry (VIO) and LiDAR odometry, enabling more robust, accurate, and adaptive localization systems. In the context of VIO, deep learning has improved the extraction of motion patterns from raw sensor data, surpassing traditional methods that rely on handcrafted features. PoseNet [49], a first deep learning-based approach for visual odometry, directly estimates the position and the orientation of the camera from a single image. Similarly, depth estimation networks have contributed significantly to VIO by providing dense depth maps from monocular images [50, 51]. Other approaches, based on Graph Neural Networks (GNN), are used to detect and match keypoints between images [52, 53]. Solutions such as DROID-SLAM [54] and Deep Patch Visual Odometry (DPVO) [55] use deep neural networks to jointly estimate the depth image from a monocular frame and optimize the robot trajectory.

In LiDAR odometry, deep learning has played a similar role in revolutionizing feature extraction and matching. Techniques like LO-Net [56] use deep neural networks to process consecutive LiDAR scans, learning to identify and mask moving objects, which leads to more accurate and reliable motion estimations in dynamic environments. Furthermore, methods such as R2D2 [57], which combine LiDAR data with deep learning-based keypoint extraction, improve feature matching accuracy and pose estimation across frames, providing robust localization even in challenging conditions. Furthermore, DELO [58] introduces a deep evidential approach to LiDAR odometry, using partial optimal transport to establish correspondences between LiDAR frames and incorporating uncertainty quantification into pose-graph optimization. This approach improves reliability, especially in environments with

partial or noisy data.

The integration of deep learning into SLAM has paved the way for more adaptive, precise, and robust navigation systems. These advances are crucial for autonomous vehicles operating in dynamic real-world environments, but a definitive solution to this problem has not yet been provided. One of the primary limitations is that state-of-the-art deep learning approaches often require substantial computational resources, such as high-performance GPUs, which are typically unavailable on lightweight or resource-constrained robotic platforms. Furthermore, these algorithms are trained on datasets captured under particular environmental conditions, and they are not able to generalize when facing new, unknown situations.

2.3.2 Place recognition

Place recognition is a robotic and computer vision task in which a robot must identify previously visited locations or match scenes to a known database [59, 60, 61]. This is a fundamental capability because recognizing known locations is the basis for long-term localization. Several methodologies have been developed for place recognition that utilize different types of sensor data, including visual, LiDAR, and multimodal inputs. In SLAM, the most commonly used methods for place recognition are based on visual data, as images provide a rich source of information with detailed textures, color variations, and semantic content. Such methods are also called appearance-based, as they rely on visual features extracted from images or video frames.

Traditional techniques employ handcrafted features [62, 63, 64]. More recently, deep learning-based methods using CNNs [65, 66] and transformers [67, 68] have demonstrated superior performance in learning robust feature representations for place recognition, improving generalization [39]. To increase robustness against occlusions or appearance changes, several strategies have been proposed to isolate and remove so-called "movable" objects from the scene [69, 70, 71]. By doing so, the network can focus solely on the more stable, static features of the environment, which are crucial for accurate place recognition. In this context, the emerging field of Generative AI holds great potential: generative models can be used to modify the scene's appearance by adding, moving, or removing objects and altering lighting or weather conditions. This approach allows the system to learn a wide variety of configurations in the same place, making it more adaptable and robust to real world changes. Finally, techniques such as contrastive learning [72, 73] and graph neural networks (GNNs) [74] are being explored to enhance robustness.

2.3.3 Loop closure Detection and Relocalization

Loop Closure Detection (LCD) is the process of determining whether an agent has returned to a previously visited location by analyzing its sensor data [47], thus determining whether it has closed a loop. In SLAM, it represents a fundamental part as it allows for error correction, increasing the overall localization accuracy and map consistency. Fig. 2.7 illustrates the Loop Closure Detection (LCD) problem. As the robot navigates, small localization errors accumulate over time, leading to an increasing drift between the estimated and actual positions. This drift affects both the trajectory and the constructed map, resulting in growing inconsistencies. When the robot revisits a previously mapped location and correctly associates new observations with past data, it recognizes the match, triggering a correction. This process, known as loop closure, refines the estimated trajectory, realigning it with the true path and consequently improving the accuracy of the entire map.

Relocalization works similarly, but refers to a different situation in which the robot must recover its pose within a previously built map. In relocalization, the robot typically recovers its pose after a loss of tracking due to occlusion or odometry failures, but it can also refer to a global relocalization when the robot "wakes up" on a known map and needs to determine its starting position [38]. In LCD and relocalization, the robot matches its current observation with a stored observation database. Relocalization can be seen as an LCD on a previously constructed map.

These concepts are strictly related to place recognition, as it is a fundamental subpart of long-term localization, whose aim is to identify possible candidates for loop closure detection or relocalization.

Hence, LCD and relocalization are powerful strategies for increasing the robustness of the SLAM process, but the consequences could be disastrous if not sufficiently accurate. Inaccurate data association can lead to two scenarios:

- The robot returns to a known location but does not recognize it, so the LCD is missed, and the estimated and actual pose continue to diverge.
- The robot is in a new location, but due to a wrong data association, it finds a match, and the LCD is done, so the estimated pose of the robot collapses to the pose of the wrong match. Since this process is typically irreversible, this is the worst-case scenario, as future exploration will inevitably lead to incorrect mapping and localization.

The classical and still widely used method for visual observation matching is Bag-of-Words (BoWs) [75], where feature vectors, known as descriptors, constitute the visual words used to define the similarity between observations. The BoW

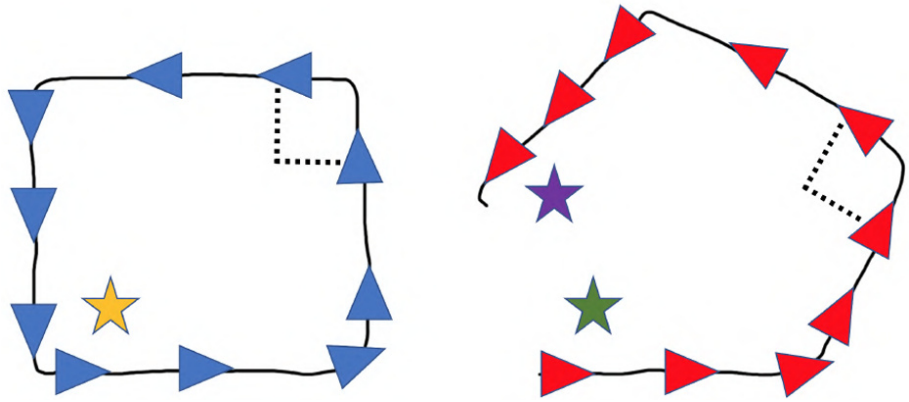


Figure 2.7: Example loop closure diagram. The robot's estimated trajectory is represented by the black line, and a subset of robot poses are represented by triangles. The stars represent recognizable features that may produce loop closures. On the left, the loop is detected and a constraint is added to the optimization problem representing the fact that the feature (star) must be in the same place in the global frame during its original and subsequent observations. On the right, the features are not recognized as the same, and the accumulated error in location estimate persists [47].

implementation choices heavily influence LCD and the entire SLAM process [38]. A key aspect of BoWs for LCD is determined by the type of vocabulary used, which can be static or dynamic.

- A static vocabulary is created offline and remains fixed during SLAM [76]. This approach simplifies the process and avoids the issue of increased computation time for vocabulary updates and searches as the map grows. However, a fixed vocabulary is not ideal for dealing with environments and conditions that differ from the initial ones. ORB-SLAM3 [77], a popular visual SLAM system, uses BoWs with a static vocabulary.

- Dynamic vocabularies are updated online by adding descriptors from incoming observations [78, 79, 80, 81, 82, 83]. This can be computationally heavier but makes the LCD more robust to real-world changing conditions. RTAB-Map [5], is a well-known SLAM system that employs dynamic vocabulary.

With the rise of deep learning, several Neural Networks based techniques have been proposed to replace BoWs by extracting more robust and invariant descriptors from images; a recent survey by Arshad et al. [39] covers the use of deep learning in different SLAM components. NetVLAD [66] and RegionVLAD [84] use a trainable layer inspired by the Vector of Locally Aggregated Descriptors (VLAD) [85], aggregating features extracted by a deep neural network to create a compact and robust representation of a single image or a group of images for image retrieval. Another

modern approach proposed by Hou et al. [86] employs deep semantic segmentation to recognize objects within an image, combining the local descriptors of these objects to create a global descriptor of the image itself.

2.4 Mapping

Mapping refers to the process of building a map that is a representation of the environment. Depending on the SLAM algorithm and the sensors used, different types of map can be obtained. Some of them are shown in Fig. 2.8. Maps can be divided into two categories [87]:

- Metric maps emphasize the exact metrical location of the objects on the map. They can be sparse or dense, depending on the construction of a sparse landmark representation of the map or a precise construction of the entire map, respectively. The dense type can be used for navigation [88].

- Topological maps emphasize the relationship among the elements of the map. These are graphs composed of nodes and edges, only considering the connectivity between nodes, so they highlight that a first node is connected to a second node, regardless of how to travel from one to the other [88].

Today, we can distinguish between two SLAM paradigms depending on the nature of the map: explicit and implicit SLAM. Explicit SLAM refers to the traditional techniques in which the map is represented explicitly, for example, using point clouds, occupancy grid maps, or meshes. In implicit SLAM, on the contrary, the map is embedded into internal continuous representations, and this is the reason for the name. Since 2020, with the advent of Neural Radiance Fields (NeRFs) [42], a number of pioneering works based on continuous neural representations have been published in this domain [43, 44]. Similarly, from 2023, Gaussian Splatting [89] began to be used in SLAM to represent the scene as a set of 3D Gaussian ellipsoids [90, 91].

Despite these advancements, several open challenges remain in the field. Implicit SLAM methods often require significant computational resources, and their real-time performance is still an area of active research. Additionally, robustness to dynamic environments and generalization across different scene types are ongoing concerns. Techniques to detect loop closures in implicit slam are currently being studied [92, 93], and there are still no definitive working solutions, especially for long-term navigation. Nevertheless, these techniques offer a promising avenue for

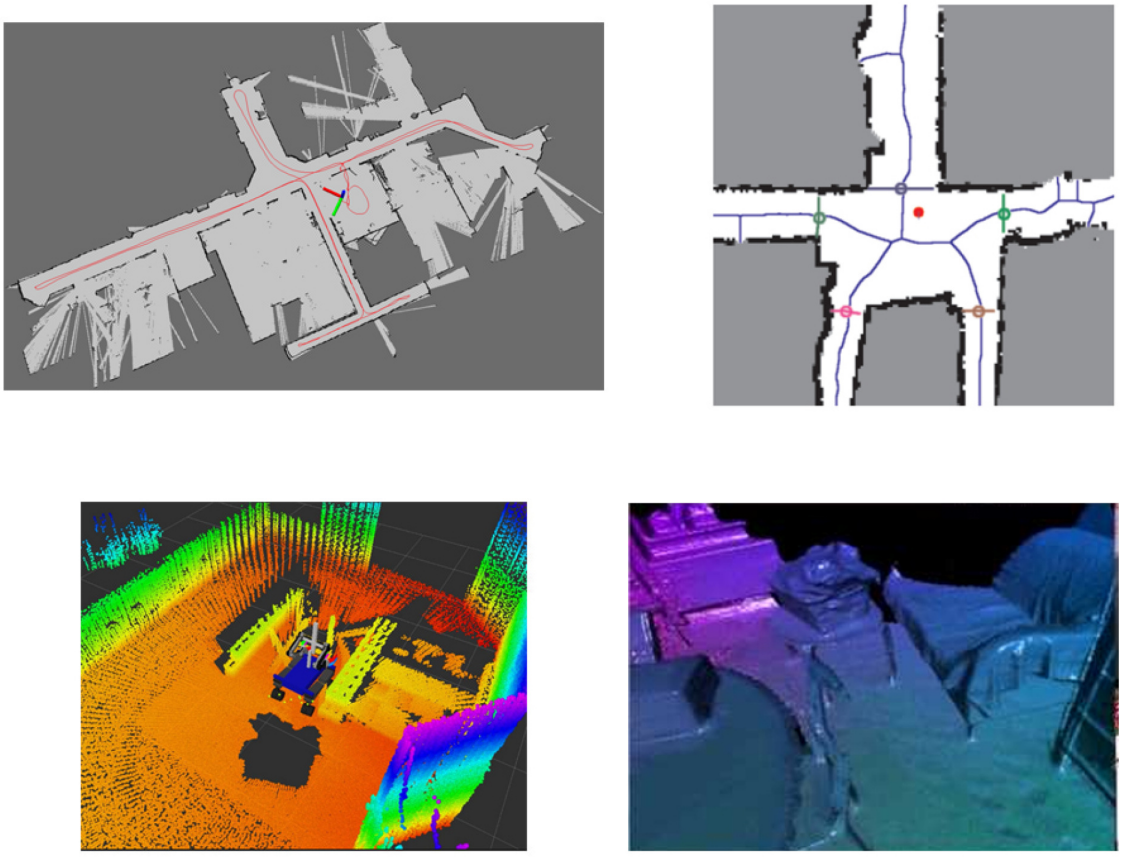


Figure 2.8: Different types of maps: 2D grid map, 2D topological map, 3D point clouds and 3D meshes [88] (from left to right, top to bottom).

reducing the memory footprint of maps by leveraging compact, continuous representations instead of storing large sets of discrete points or occupancy grids. This reduction in map size could be particularly beneficial for mobile robotics, AR/VR applications, and autonomous navigation, where efficient storage and processing are critical constraints. As research progresses, the integration of implicit representations with traditional SLAM methods may further improve scalability, efficiency, and adaptability in real-world scenarios.

2.4.1 Map management techniques

Various techniques are proposed in the literature to manage large maps. An effective approach involves using a subset of nodes (i.e., a submap) for time-consuming tasks such as loop closure detection (LCD) and map optimization. For instance, [94] suggests dividing the map into smaller submaps, creating or activating different submaps as the robot enters new areas (e.g., rooms). However, switching between submaps can be complex in non-structured environments. The approaches in [95, 96, 97] divide the map into regions based on semantic and robust visual features.

RTAB-Map (Real-Time Appearance-Based Mapping) is a RGB-D, Stereo and Lidar Graph-Based SLAM approach based on an incremental appearance-based loop closure detector [5]. It employs a native memory management system with three types of memory [98]:

- Short-Term Memory (STM): a buffer containing the newest nodes of the map.
- Working Memory (WM): a working area (with a parameterized maximum size) used for loop closure detection and graph optimization.
- Long-Term Memory (LTM): a local database that stores all the knowledge about the map.

RTAB-Map performs LCD using a Bag-of-Words approach with a dynamic vocabulary that is periodically recreated using only the nodes in WM. This dynamic approach updates the vocabulary without increasing its size as the map grows. Map optimization is also limited to WM nodes, thus limiting the resources required. A *transfer-retrieval* process runs in the background, transferring nodes between WM and LTM (Fig. 2.9), ensuring that WM retains nodes similar to the current observation. However, a significant drawback of the native RTAB-Map implementation is that node selection is based on temporal and spatial proximity. Consequently, in large loops, the target nodes may not be in WM, leading to missed closures. This is the typical case where the robot returns to an already visited place after navigating for a long time elsewhere.

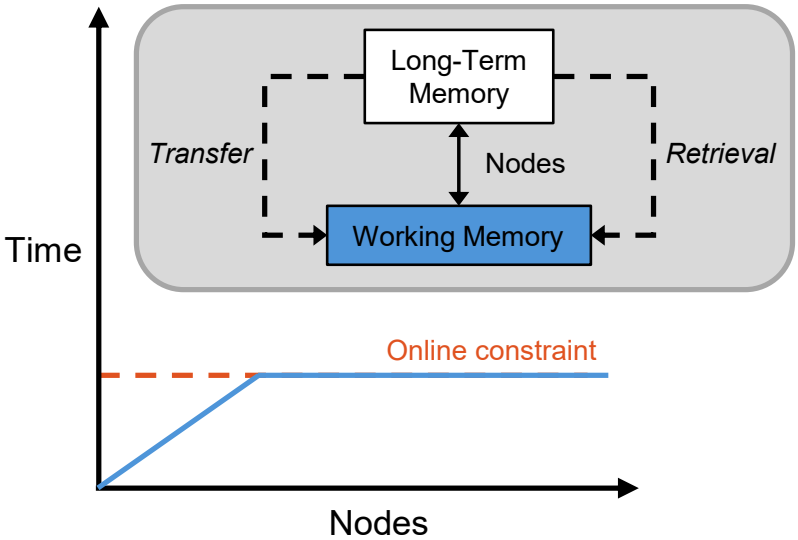


Figure 2.9: RTAB-Map transfer-retrieval process and time complexity. As the number of nodes reaches the maximum VM size, the time complexity does not further increase.

2.5 Continual SLAM

Continual SLAM, also referred to as lifelong SLAM, addresses long-term, extended SLAM systems. The term can be somewhat misleading, as it does not necessarily imply the use of continual learning techniques. In many cases, continual SLAM focuses on improving performance over extended periods by effectively handling changing environments. A common approach involves distinguishing between static and dynamic elements and modeling the map with an emphasis on features likely to change in the future [99, 100, 101, 102]. Of course, continual learning techniques are also employed within continual SLAM, and recent literature explores various ways in which continual learning can be integrated into SLAM frameworks.

In [103], visual odometry is approached as a deep learning problem (similar to previous works such as [50]) utilizing CL to adapt the network to varying environments. A replay memory mechanism is integrated to mitigate catastrophic forgetting by replaying old patterns during training.

AirLoop [104] specifically addresses loop closure detection using a convolutional neural network to extract image features and framing the problem as an image retrieval task. This study emphasizes the challenges posed by CL settings with streaming data and employs a combination of contrastive learning, replay memory, and regularization techniques to manage these challenges.

BioSLAM [95] represents a more comprehensive and sophisticated SLAM solution, leveraging deep learning techniques across all its modules. This bio-inspired approach incorporates deep visual odometry, static and dynamic memories for localization, and generative replay to prevent forgetting.

Storing and replaying part of old data is one of the most effective CL approaches. However, in real robotic applications, the size of the replay buffer cannot exceed a given limit. Sample selection techniques have been proposed to manage the amount of data retained in the replay memory. The authors of [105] formulated the sample selection problem as a constrained optimization problem. In [106] a continual visual-inertial odometry (CoVIO) is introduced where examples are added to (or retained in) the replay buffer when their cosine distance with the other samples in the buffer is high. The CL method proposed in [107], adaptively decides what to consolidate and what to forget, aiming to tolerate the environment changes determined by moving objects. An example of Continual SLAM that does not rely on continual learning is detailed in [108], where for each new place, a new experience bucket is created, where visual information for a place are collected and then used for localization. Despite these advancements, most of the aforementioned methods are not yet mature enough for integration into a fully functional SLAM system and/or lack scalability for large

maps due to their inherent complexity. In contrast, the approach proposed in this thesis is fully integrated into a state-of-the-art SLAM system, RTAB-Map, and demonstrates real-time performance on large maps.

In a broader sense, lifelong SLAM can be seen as a paradigm in which the traditional separation between mapping and localization becomes blurred, much like the distinction between training and inference in neural networks. In the context of lifelong SLAM, the environment cannot be mapped just once for subsequent use; instead, the system must continuously update the map during future navigations, rather than relying solely on localization. Here, every location encountered may either be entirely new, requiring mapping, or previously visited, requiring localization; moreover, previously mapped areas may undergo changes that necessitate simultaneous localization and map adaptation. Consequently, a lifelong SLAM system must not only perform localization and mapping in tandem but also continuously revise its internal representation of the environment. Similarly, the underlying neural networks must continually update their knowledge while simultaneously leveraging past information to make reliable predictions.

2.6 Mathematical foundation of SLAM

During navigation, at any discrete time instant t, the robot changes its state and observes an unknown number of landmarks (Fig 2.10). The key ingredients of the process can be summarized as follows [1]:

- $\mathbf{x}_t$ is the state vector, representing the current state of the robot. The current state can be the robot pose (i.e., a position and orientation), the velocity, the acceleration, etc.
- $\mathbf{u}_t$ is the control vector. Applied at time $t-1$, it evolves the state of the robot from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$.
- $\mathbf{z}_t$ is the observations vector. It corresponds to the sensors measurements. The observation of the i^{th} landmark is $\mathbf{z}_t^i$;
- $\mathbf{m}$ represents the map of the environment. The location of the i^{th} landmark on the map is $\mathbf{m}_i$.

Or in matrix format:

- $\mathbf{X}_{0:t} = \{\mathbf{x}_0, \mathbf{x}_1, ..., \mathbf{x}_t\} = \{\mathbf{X}_{0:t-1}, \mathbf{x}_t\}$ the history of robot states;
- $\mathbf{U}_{1:t} = \{\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_t\} = \{\mathbf{U}_{1:t-1}, \mathbf{u}_t\}$ the history of robot control inputs;

- $\mathbf{Z}_{0:t} = \{\mathbf{z}_1, \mathbf{z}_2, ..., \mathbf{z}_t\} = \{\mathbf{Z}_{0:t-1}, \mathbf{z}_t\}$ the history of environment observations.

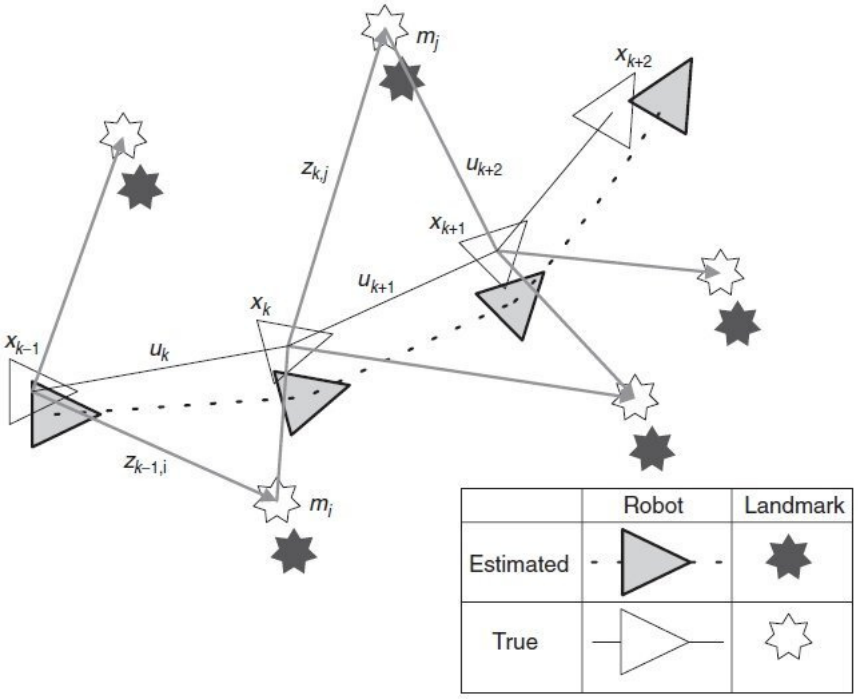


Figure 2.10: The essential SLAM problem. In the image the symbol k stands for the time instant t.

The pose of a robot is composed of a position, which is translation with respect to the origin, and an orientation, which is rotation with respect to the origin. In SLAM, poses and motions are typically represented using a transformation matrix (or equivalent notations) in the n-dimensional space. In fact, SLAM can be 2D or 3D, depending on the sensors used. Given d as the number of dimensions, the transformation matrix $\mathbf{T}$ is defined as a *Special Euclidean Group* $SE(d)$ formed by a *Special Orthogonal Group* $SO(d)$, that is a rotation matrix $\mathbf{R}$, and a translation vector $\mathbf{t}$.

$$SO(d) = \{\mathbf{R} \in \mathbb{R}^{d \times d} | \mathbf{R}\mathbf{R}^T = \mathbf{I}, det(\mathbf{R}) = 1\}. \tag{2.4}$$

$$SE(d) \ = \left\{\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \in \mathbb{R}^{(d+1)\times(d+1)} | \mathbf{R} \in SO(d), \mathbf{t} \in \mathbb{R}^d \right\}. \tag{2.5}$$

Considering a 3D SLAM, $\mathbf{R}$ is a 3×3 orthogonal matrix, $\mathbf{t}$ is a 3×1 vector, and $\mathbf{T}$ is a 4×4 matrix composed of $\mathbf{R}$ and $\mathbf{t}$.

The initial pose at $t = 0$ coincides with the origin of our reference frame and it is typically the identity matrix $\mathbf{T}_0 = \mathbf{I} \in \mathbb{R}^{d+1}$ (i.e., no rotation and no translation).

The process can be summarized as follows: $\mathbf{x}_0$ is the initial state and each control $\mathbf{u}_t$ is applied to transform the state from $\mathbf{x}_{t-1}$ to $\mathbf{x}_t$. Hence

$$\mathbf{x}_t = \mathbf{x}_0 \circ \mathbf{u}_1 \circ \mathbf{u}_2 \circ \cdots \circ \mathbf{u}_t \tag{2.6}$$

where "$\circ$" denotes group composition.

Depending on the formulation of the problem, the state $\mathbf{x}_t$ can include only the robot's pose (in which case $\mathbf{u}_t$ is simply the relative transformation $\mathbf{T}$ between two poses), or it can also encompass additional parameters, such as velocity, with $\mathbf{u}_t$ defined accordingly.

2.6.1 Problem definition

SLAM is commonly formulated as a probabilistic state estimation problem. The specific formulation depends on the scope of the estimated information, leading to two main approaches: *online SLAM* (Fig. 2.11a) and *full SLAM* (Fig. 2.11b). Online SLAM estimates the posterior probability of the robot's most recent state, as expressed in Equation 2.7. In contrast, full SLAM reconstructs the entire trajectory by incorporating all past states, as described in Equation 2.8.

$$P(\mathbf{x}_t, \mathbf{m} | \mathbf{Z}_{0:t}, \mathbf{U}_{1:t}, \mathbf{x}_0). \tag{2.7}$$

$$P(\mathbf{X}_{0:t}, \mathbf{m} | \mathbf{Z}_{0:t}, \mathbf{U}_{1:t}, \mathbf{x}_0). \tag{2.8}$$

Depending on the specific formulation, different solutions have been proposed, discussed in the following.

2.6.2 Proposed solutions

From a theoretical point of view, SLAM can be considered a solved problem because several mathematical solutions are proposed in the literature. They can be distinguished into two main categories, strictly related to the definition of the SLAM problem (online vs. full): *filtering-based SLAM* and *smoothing-based SLAM*.

Filtering-based SLAM

Filtering approaches are the oldest and are typically used to solve the online SLAM formulation. The estimate is augmented and refined by incorporating the new measurements as they become available. It is called "filtering-based" because it is solved

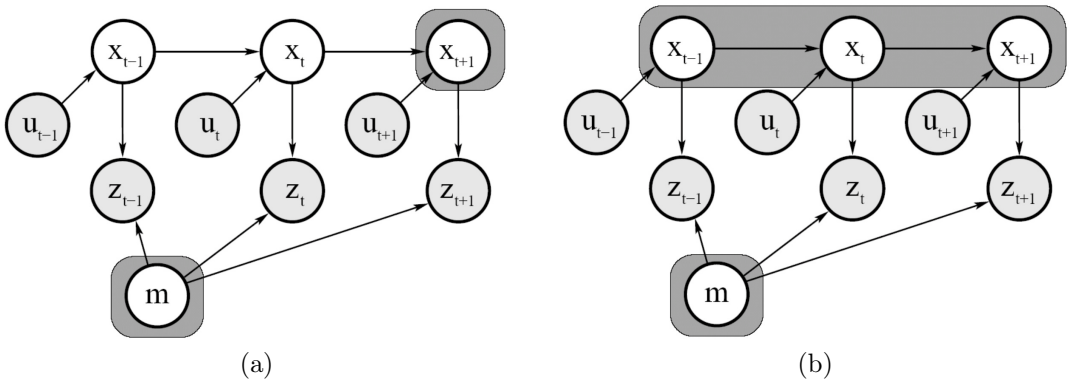


Figure 2.11: On the left, (a) depicts the online SLAM process, in which only the current state $\mathbf{x}_t$ and the map $\mathbf{m}$ are estimated, on the right, (b) represents the full SLAM problem, which maintains a joint estimate of all past states $\mathbf{X}_{0:t}$ and the map $\mathbf{m}$, allowing for global consistency at the cost of higher computational demand.

using recursive filters, such as the Kalman filter. In this approach, the goal is to compute the probability distribution reported in Equation 2.7. This is a joint posterior density of the state of the robot and the map, given the set of observations and controls and the initial state. To solve this, typically a prediction step and a correction step are done recursively. At time $t-1$, starting with an estimate for the probability distribution $P(\mathbf{x}_{t-1}, \mathbf{m}|\mathbf{Z}_{0:t-1}, \mathbf{U}_{1:t-1})$, and given the control vector $\mathbf{u}_t$ and the observations vector $\mathbf{z}_t$, the joint probability at time t can be computed applying the Bayes rule. To this purpose, an *observation model* and a *motion model* are defined.

The observation model represents the probability of making an observation given the state of the robot and the map, in the form

$$P(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m}). \tag{2.9}$$

The motion model is the probability distribution over the state transition, which is the probability of moving in a specific state given the previous state and a command vector, in the form

$$P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t). \tag{2.10}$$

In this formulation, the state at time t depends only on the state at time $t-1$, so the process has no memory of the previous states except the last one (the so-called Markov assumption), while $\mathbf{u}_t$ depends neither on observations nor on the state. The two steps of prediction and correction are computed as follow.
Prediction step:

$$P(\mathbf{x}_t, \mathbf{m}|\mathbf{Z}_{0:t-1}, \mathbf{U}_{1:t}, \mathbf{x}_0) = \int P(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t) \times P(\mathbf{x}_{t-1}, \mathbf{m}|\mathbf{Z}_{0:t-1}, \mathbf{U}_{1:t-1}, \mathbf{x}_0)\, d\mathbf{x}_{t-1}. \tag{2.11}$$

Correction step:

$$P(\mathbf{x}_t, \mathbf{m}|\mathbf{Z}_{0:t}, \mathbf{U}_{1:t}, \mathbf{x}_0) = \frac{P(\mathbf{z}_t|\mathbf{x}_t, \mathbf{m}) \times P(\mathbf{x}_t, \mathbf{m}|\mathbf{Z}_{0:t-1}, \mathbf{U}_{1:t}, \mathbf{x}_0)}{P(\mathbf{z}_t|\mathbf{Z}_{0:t-1}, \mathbf{U}_{1:t})}. \tag{2.12}$$

The prediction step tries to estimate where the robot will be after the movement, and due to the noise, this prediction is uncertain. In the correction step, the robot senses the environment and, thanks to landmark observations and an accurate data association, can correct the prediction, making the state estimation more precise.

This SLAM formulation should not be confused with pure odometry. In fact, only the current state of the system is estimated at each iteration and the process has no memory of past states, but the goal of the system is to predict also the state of the map (e.g., in landmark-based EKF), making it different from pure odometry. Since there is no memory of past states, the filtering-based approaches cannot handle loop closure detection and past error corrections, leading to drift accumulation that can never decrease.

Smoothing-based SLAM

The second approach is called smoothing-based SLAM, as it estimates the entire robot trajectory as a whole, taking into account in the posterior probability all the state vectors, as reported in Equation 2.8, and it is typically solved with least-square error minimization.

It is better known as graph-based SLAM because the map is represented as a graph. To introduce the notation used in the work of this thesis, while remaining consistent with the consolidated notation proposed in the literature, we define the concept of vertices and arcs associated with robot poses and virtual measurements, respectively.

The graph $\mathcal{G}$ is defined as a connected graph composed of n_v vertices V (or nodes) and n_a arcs A (or edges). Each node v is associated with a robot state $\mathbf{x}$, which corresponds to a pose here, and each arc a_{ij} is associated with a virtual measurement $\mathbf{z}_{ij}$, which represents a spatial constraint between the two poses $\mathbf{x}_i$ and $\mathbf{x}_j$ associated to v_i and v_j, respectively.

In the following chapters, a node in the graph represents a broader concept than a robot pose only (e.g., also the data acquired in this node); since the aim here is to

formalize the graph-based SLAM approach, these concepts are used interchangeably.

Let $\mathbf{z}_{ij}$ and $\mathbf{\Omega}_{ij}$ be the mean and the information matrix of the virtual measurement between $\mathbf{x}_i$ and $\mathbf{x}_j$, respectively. This virtual measurement is a transformation that makes the observations acquired from $\mathbf{x}_i$ maximally overlap with the observation acquired from $\mathbf{x}_j$, that is in the form of the transformation matrix $\mathbf{T} \in SE(d)$ as defined in Equation 2.5. Given a configuration of nodes $\mathbf{x}_i$ and $\mathbf{x}_j$ and a prediction of the virtual measurement $\hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$, the log-likelihood of $\mathbf{z}_{ij}$ is

$$l_{ij} \propto [\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]^T \mathbf{\Omega}_{ij} [\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]. \tag{2.13}$$

The error function that computes the difference between the real observation and the expected observation is defined as follows

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j). \tag{2.14}$$

Defining $\mathcal{C}$ as the set of pairs of indices for which a constraint exists, the goal of the maximum likelihood is to find the optimal configuration of the nodes $\mathbf{x}^*$ that minimizes the negative log-likelihood $\mathbf{F}(\mathbf{x})$ of all the observations

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{C}} \mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}, \tag{2.15}$$

that is equivalent to solve

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}} \, \mathbf{F}(\mathbf{x}). \tag{2.16}$$

This minimization problem can be solved with traditional optimization algorithms such as Gauss-Newton or Levenberg-Marquardt.

From a practical point of view, during exploration, as a new node v_i is created, odometry aims to estimate the virtual measurement $\mathbf{z}_{pi}$ between the previous node v_p and v_i, and a new arc a_{pi} is created with the associated measurement. The new pose of the robot $\mathbf{x}_i$ is then computed and associated with v_i.

This is a classical formulation known as *Pose Graph*, because only robot poses are considered as nodes in the graph. There is another formulation, in which landmarks are also included as nodes, with additional constraints connecting robot poses to landmarks, depending on the visibility of the landmarks from each node. In both cases, the graph is very sparse, because each node is connected only with a small subset of the other nodes.

The goal is to optimize the configuration of graph nodes (i.e., the robot poses and possibly the position of the landmarks) to minimize the errors introduced by

the constraints. This process takes the name of *Pose Graph Optimization* (PGO) if a pose graph is considered, while the joint optimization of poses and landmarks is known as *Bundle Adjustment* (BA).

Graph-based SLAM is a global optimization problem in which all observations are used to refine the entire map and trajectory. This formulation better handles loop closures because when the robot associates the observations taken in the current node v_i with those acquired in a known location v_q, an arc a_{iq} with an associated virtual measurement $\mathbf{z}_{iq}$ is added between the two nodes, thus closing a loop. This action triggers pose graph optimization, adjusting node poses and virtual measurements to reduce the error and maximize global map consistency.

This approach represents the state-of-the-art because it allows drift correction, which is essential to achieve long-term map consistency. However, in large explorations, optimizing the entire map can prevent the real-time constraint. In addition, optimization can get stuck in local minima, so good initialization is critical when using classical algorithms such as Gauss-Newton or Levenberg-Marquardt.

2.7 Metrics and Benchmark for SLAM

Evaluating the performance of a SLAM algorithm is a challenging task that requires formal metrics to ensure objective assessment. Typically, this evaluation relies on ground-truth data, which provides the precise pose of the robot at each instant. Standard benchmarking approaches involve aligning the timestamps of the ground-truth and estimated trajectories, followed by a comparative analysis of the estimated and actual poses. However, alternative evaluation methods exist for scenarios where ground-truth data is unavailable. To facilitate SLAM assessment, various datasets have been acquired and released to the community, and several evaluation metrics have been proposed in the literature, tailored to different types of SLAM applications. This section introduces conventional SLAM metrics as well as novel metrics specifically designed for long-term SLAM evaluation.

2.7.1 Traditional SLAM metrics

Two traditional metrics have been proposed in the literature to evaluate SLAM systems: the *Absolute Trajectory Error* (ATE), and the *Relative Pose Error* (RPE). There are useful tools for SLAM benchmarks that calculate these metrics given the estimated trajectory and the ground-truth trajectory, as for example the scripts created by Computer Vision Group of the Department of Informatics of Munich [109] for the RGB-D SLAM benchmark.

Two sequences of poses are provided for the evaluation, the estimated trajectory $\mathbf{P}_1, ..., \mathbf{P}_n \in SE(3)$ and the ground-truth trajectory $\mathbf{Q}_1, ..., \mathbf{Q}_n \in SE(3)$.

Absolute Trajectory Error

The absolute trajectory error (ATE) measures the global consistency of the estimated trajectory of the robot [110]. It is evaluated by comparing the absolute distances between the estimated and the ground-truth trajectory (Fig. 2.12). Since both trajectories can be defined in arbitrary coordinate frames, alignment is necessary before comparison. This alignment is typically achieved using Horn's method [111], which computes the rigid-body transformation $\mathbf{S}$ that provides the least-squares solution for mapping the estimated trajectory $\mathbf{P}_{1:n}$ onto the ground truth trajectory $\mathbf{Q}_{1:n}$. The absolute trajectory error at time step i is then calculated as:

$$\mathbf{F}_i := \mathbf{Q}_i^{-1}\mathbf{S}\mathbf{P}_i. \tag{2.17}$$

In visual SLAM, the absolute trajectory error on the entire path is given by the RMSE over all the time indices of the translational components as

$$RMSE(\mathbf{F}_{1:n}) = \left(\frac{1}{n}\sum_{i=1}^{n} ||trans(\mathbf{F}_i)||^2\right)^{1/2}. \tag{2.18}$$

In the most recent metrics, presented in 2.7.2, also the *Absolute Orientation Error* (AOE) is considered, and it can be calculated in the same way as the ATE, considering the orientational components, as shown in Equation 2.19.

$$RMSE(\mathbf{F}_{1:n}) = \left(\frac{1}{n}\sum_{i=1}^{n} ||orientation(\mathbf{F}_i)||^2\right)^{1/2}. \tag{2.19}$$

Relative Pose Error

The relative pose error (RPE) measures the local accuracy of the trajectory over a fixed time interval Δ [110]. Thus, the relative pose error corresponds to the drift of the trajectory (Fig. 2.13). Contrary to the ATE, it does not require the global rigid alignment, because the discrepancy between corresponding poses in the estimated trajectory and the ground truth is the error itself. The RPE at time step i is defined as

$$\mathbf{E}_i := \left(\mathbf{Q}_i^{-1}\mathbf{Q}_{i+\Delta}\right)^{-1}\left(\mathbf{P}_i^{-1}\mathbf{P}_{i+\Delta}\right). \tag{2.20}$$

If the ground truth part and the estimated part are identical, $\mathbf{E}_i$ is the identity

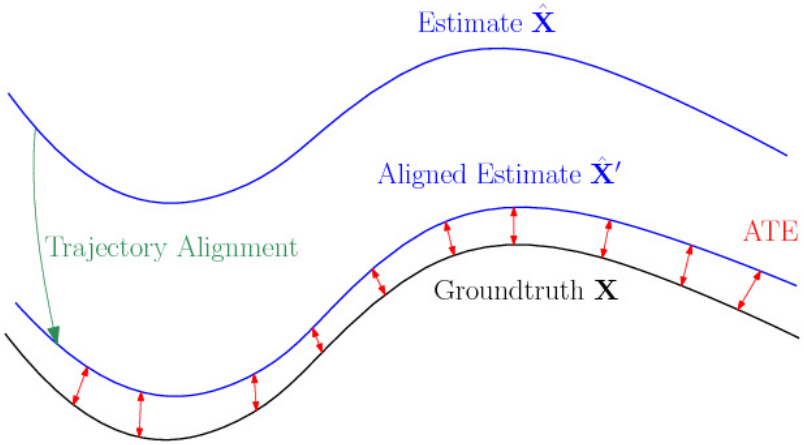


Figure 2.12: Absolute trajectory error. The estimated trajectory is first aligned with the ground truth.

matrix. So, given a sequence of n poses, $m = n - \Delta$ are the individual relative pose errors along the sequence and as for the ATE, the RMSE is computed on the translational components as follows

$$RMSE(\mathbf{E}_{1:n}, \Delta) = \left(\frac{1}{m} \sum_{i=1}^{m} ||trans(\mathbf{E}_i)||^2 \right)^{1/2}. \tag{2.21}$$

Changing the Δ value leads to compute the drift over a different number of poses. For example with $\Delta = 1$ the error measures the drift per frame.

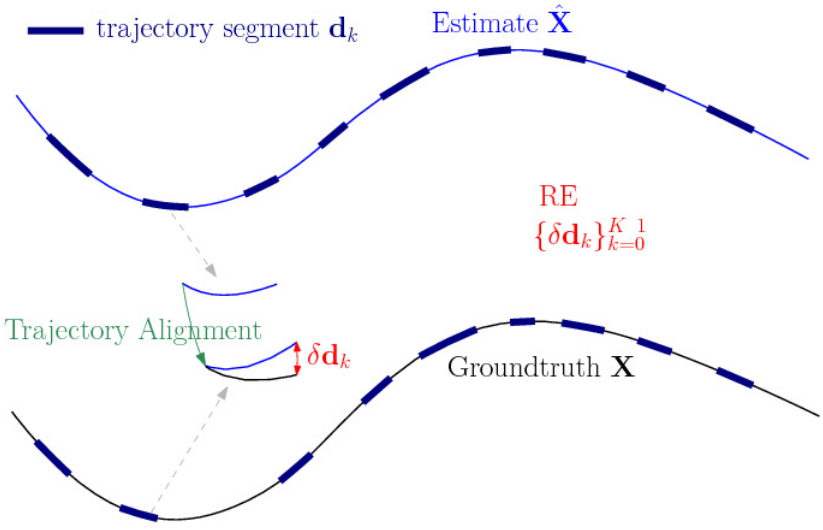


Figure 2.13: Relative pose error. In this case, it is not necessary to align the estimated trajectory with the true one.

2.7.2 Lifelong SLAM metrics

To achieve lifelong SLAM, two fundamental properties are needed: robustness and scalability. Scalability is not strictly related to the classic SLAM metrics, in fact it is more related to memory occupation and time complexity than on the accuracy of the SLAM process. Two types of long-term SLAM metrics have recently been proposed in the literature: robustness metrics and accuracy metrics [112].

Robustness metrics

To evaluate the robustness, several measures are defined.

Correctness

For each estimated pose $\mathbf{p}_i$ at time $\mathbf{t}_i$, given the ground-truth pose at that time, the correctness of the estimate is computed from its ATE and AOE as follow

$$c^{\varepsilon,\phi} = \begin{cases} 1, & \text{if ATE}(\mathbf{p}_i) \leq \varepsilon \text{ and AOE}(\mathbf{p}_i) \leq \phi \\ 0, & \text{otherwise} \end{cases} \tag{2.22}$$

where ε is the ATE threshold and ϕ is the AOE threshold. They should be chosen according to the area of the scene and the expected drift of the algorithm. Correctness is an on-off indicator that evaluates the precision of a single pose, comparing the errors with the two thresholds.

Correct Rate (CR) and Correct Rate of Tracking (CR-T)

These measures evaluate the overall robustness over one or more data sequences, that is the correct rate over the whole time span of data. Given a sequence from $\mathbf{t}_{min}$ to $\mathbf{t}_{max}$ and the estimated-true trajectory $\{\mathbf{t}_i, \mathbf{p}_i\}_{i=0,\dots,N}$, it is possible to define

$$\text{CR}^{\varepsilon,\phi} = \frac{\sum_{i=0}^{N} (min(\mathbf{t}_{i+1} - \mathbf{t}_i, \delta) \cdot c^{\varepsilon,\phi}(\mathbf{p}_i))}{\mathbf{t}_{max} - \mathbf{t}_{min}}, \tag{2.23}$$

$$\text{CR}^{\varepsilon,\phi}\text{-T} = \frac{\sum_{i=0}^{N} (min(\mathbf{t}_{i+1} - \mathbf{t}_i, \delta) \cdot c^{\varepsilon,\phi}(\mathbf{p}_i))}{\mathbf{t}_{max} - \mathbf{t}_0}, \tag{2.24}$$

where $\mathbf{t}_{N+1} \doteq \mathbf{t}_{max}$ while δ is a parameter to determine how long a correct pose estimation is valid for and it should be set larger than the pose estimation time, but much smaller than the time span of the sequence. Typically, it is set to one second. The insight here is that robustness is calculated as the sum of the correctness of

each pose multiplied by the time the pose $\mathbf{p}_i$ is correct in the sequence, that is $min(\mathbf{t}_{i+1} - \mathbf{t}_i, \delta)$, normalized by the total time span of the sequence. In $\text{CR}^{\varepsilon,\phi}$-T the time for re-localization and the initialization $(\mathbf{t}_0 - \mathbf{t}_{min})$ is excluded, because tracking does not work during that time.

Correctness Score of Re-localization (CS-R)
This measure evaluates the re-localization score of the robot. Re-localization is the task in which the robot must recognize its location on the map. This metric also takes into account how much time the re-localization takes and it's defined as follow

$$\text{C}^{\varepsilon,\phi}\text{S}^{\tau}\text{-R} = e^{-(\mathbf{t}_0 - \mathbf{t}_{min})/\tau} \cdot c^{\varepsilon,\phi}(\mathbf{p}_0) \tag{2.25}$$

where τ is a scaling factor, typically set to 60 seconds. It can be noticed that if the re-localization is instantaneous, so that $\mathbf{t}_0 = \mathbf{t}_{min}$, CS-R = 1.

Accuracy metrics

Regarding accuracy, the proposal is to use ATE and RPE statistics on one or more sequences taking into account only correct estimations of the pose. For example, $\text{C}^{0.1}$-RPE RMSE is the root mean square error of RPE of correct pose estimates selected from an ATE threshold of 0.1 meter.

2.7.3 Loop closure detection metrics

Accuracy alone is not a reliable metric for evaluating loop closure detection, as it is highly sensitive to both the trajectory and the characteristics of the environment. For example, an algorithm may completely miss valid loop closures in one scenario, while in another it may incorrectly match visually similar but distinct locations. To overcome these limitations, *Precision–Recall* is typically adopted as the primary evaluation metric (Fig. 2.14). Precision–Recall is especially appropriate in settings with highly imbalanced classes [113], where the distribution of true loop closures can vary greatly across environments. Some trajectories may contain many valid closures, while others may include only false candidates arising from perceptual aliasing. Unlike accuracy, which only reflects overall correctness, Precision–Recall captures the trade-off between true positives, false positives, and false negatives, offering a more robust and informative measure of loop closure detection performance.

Precision

Precision quantifies the relevance of the detected results. It represents the fraction of correctly identified loop closures among all detections made by the algorithm.

Formally, precision (P) is defined as the ratio between the number of true positives (T_p) and the sum of true positives and false positives (F_p):

$$P = \frac{T_p}{T_p + F_p}.$$

In the context of loop closure detection, precision corresponds to the proportion of detected loop closures that are actually correct.

Recall

Recall measures the completeness of the detections. It indicates the fraction of true loop closures that the algorithm successfully identifies out of all loop closures that exist in the environment.

Formally, recall (R) is defined as the ratio between the number of true positives (T_p) and the sum of true positives and false negatives (F_n):

$$R = \frac{T_p}{T_p + F_n}.$$

For loop closure detection, recall reflects how many of the actual loop closures present in the explored environment are correctly detected.

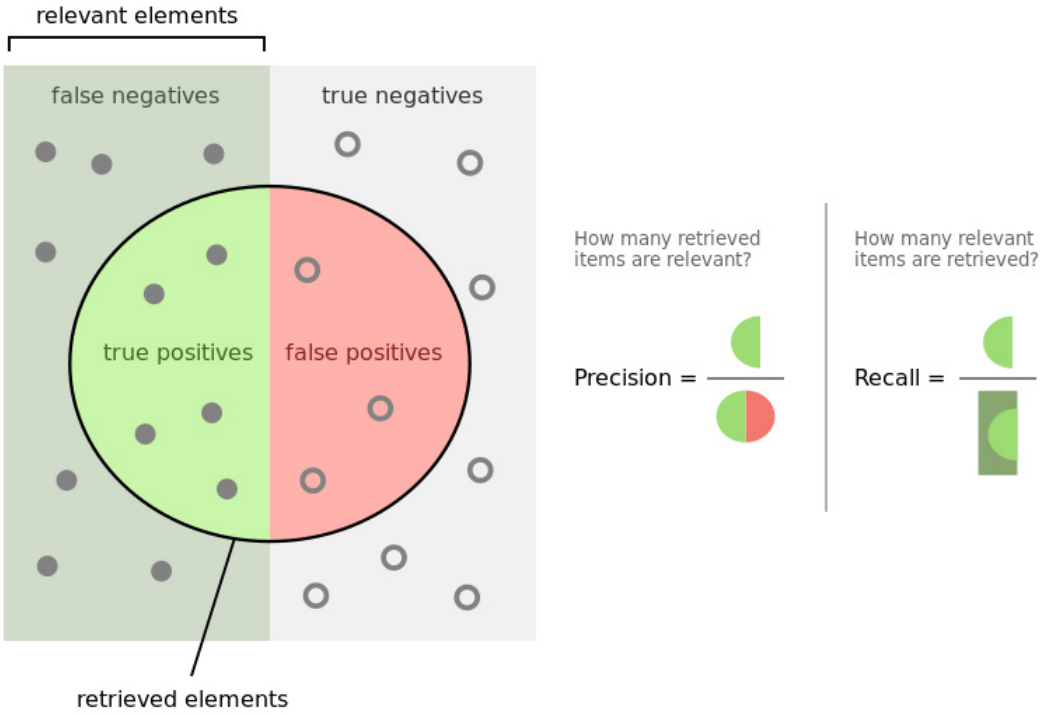


Figure 2.14: Precision–Recall metric.

Chapter 3

Region Prediction for Efficient Robot Localization on Large Maps

"Simplicity is the supreme sophistication."

– Leonardo Da Vinci

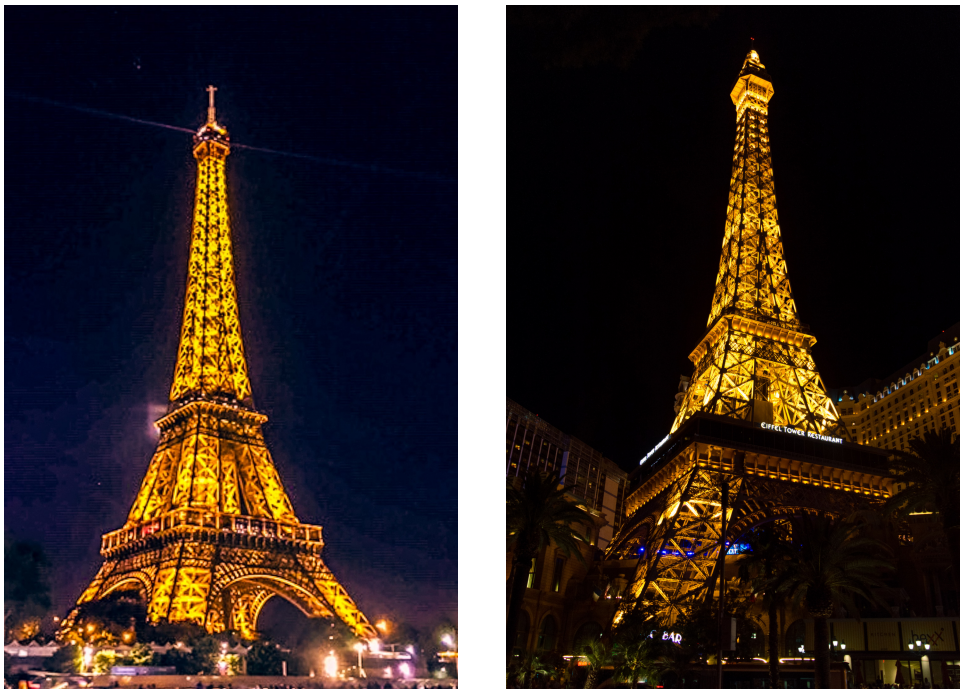
In this chapter, we present a novel methodology for long-term localization, very simple but effective, and in some sense inspired by human localization capabilities. A first preliminary work that aims to demonstrate the effectiveness of this methodology is then discussed.

3.1 The fundamental idea

To illustrate the fundamental idea behind our approach, consider the following analogy: imagine being blindfolded and transferred to an unknown location. When the blindfold is removed, you find yourself in front of a familiar place, but one that you cannot precisely identify. To understand where you are, you start exploring, moving in the environment, and searching for new information to disambiguate this familiar place. In these few seconds, your brain is making thousands of hypotheses and associations with the experience in your memory, and each new piece of information is useful to discard some hypotheses and to consolidate others. In such a situation, our brain does not precisely know where we are, but it can qualitatively estimate our location in the world until a single hypothesis about our true location is confirmed. Often, a single view of a place is not enough to correctly understand where we are, we need more information. Looking at Fig. 3.1 below, you could think of being in Paris, facing the Eiffel Tower from two different viewpoints. But if you were there, it would probably only take you a few seconds of exploration to realize that you

are completely wrong. In fact, Fig. 3.1a depicts the original Eiffel Tower in Paris, while Fig. 3.1b is a fake copy of the Eiffel Tower situated in Las Vegas (you can see the buildings behind the tower and also the sign "Eiffel Tower Restaurant" on the tower). This shows that more information than a single image may be needed to disambiguate our location. Moreover, our brains have a strong notion of time, and our experience in the physical world makes us aware that we cannot teleport instantaneously from one point to another, so our location cannot change abruptly in a few moments.

Despite these simple and quite obvious observations, most LCD and relocalization methods used today do not take into account multiple images to perform long-term localization, nor the concept of temporal consistency. Building on the observations above, our method aims to implement a similar mechanism to very quickly discard unlikely hypotheses and consolidate others until only one can be considered correct and localization can occur. We explain the approach below, starting with a very general overview. We take special care to explain the approach in a very general way to show that it is not constrained to our particular implementation, but can be easily adapted to other implementations as well, since the underlying idea remains general and effective.



(a) (b)

Figure 3.1: On the left, (a) a photo of the original Eiffel Tower in Paris, on the right, (b) shows a photo of a replica in Las Vegas.

3.2 Method overview

The method proposed in this work builds upon existing SLAM frameworks without aiming to redefine their fundamental principles. In particular, it does not replace the standard components of a SLAM pipeline: odometry estimation can still be obtained through any conventional approach, the map remains represented as a graph, and loop closures may continue to be detected using Bag-of-Words techniques. Instead, the focus here is complementary: we address the problem of long-term localization, that is, the challenge of efficiently identifying and maintaining the relevant parts of the map during operation.

The core idea is to guide the SLAM process by restricting attention to a smaller subset of the map that is most consistent with the current observations. In this way, the method acts as a pre-filter that flanks the standard SLAM modules, determining which regions of the map are worth considering and which can be safely ignored. This design makes the approach compatible with any graph-based SLAM system, as it provides an additional layer of selection without interfering with the underlying estimation machinery.

To achieve this, the method is structured around three main components:

1. A dynamic clustering procedure to partition the map into spatial areas, hereafter referred to as *regions*.

2. A deep neural network that formulates and evaluates hypotheses about region membership.

3. A temporal filtering mechanism that ensures consistency and robustness over time.

The process unfolds as follows. During the navigation, the environment is dynamically clustered, which provides a high-level partitioning of the map into distinct regions. Every time a new bunch of nodes paired with their corresponding region labels is available, the neural network starts training to associate incoming observations with their most likely region. This step introduces the ability to discard a large number of implausible hypotheses early on and to progressively refine the plausible ones as new data arrives. At the same time, the predictions generated by the network are passed through a temporal filter and used to discard nodes from the detection process. The role of this filter is to smooth the sequence of region assignments, reducing the likelihood of abrupt changes and enforcing temporal coherence in the localization process. In this way, only the preselected set of nodes

will be involved in loop closure detection and eventually in the successive graph optimization, making the overall process more efficient, allowing at the same time for the reduction of false positive loop closures that could arise when there are strong autosimilarities between different environments.

It is worth emphasizing that each of these three components can be instantiated in different ways depending on the application scenario. The clustering algorithm may vary, different types of observations can be exploited, the architecture of the neural network can be adapted, and several choices exist for the temporal filtering strategy. The strength of the proposed approach lies in this generality: this method provides a flexible framework where the specific ingredients may change, but the overall principle, the region-based preselection to support long-term localization, remains the same.

While in the general case the entire process takes place online and also the neural network used for region prediction should be trained online and in a continual learning fashion (i.e., without a clear distinction between training and inference), we developed a preliminary work aimed at demonstrating the underlying idea, in which we adopt some simplifications and identify four phases:

1. **Exploration**, in which the robot explores the environment, builds the map and creates the regions through dynamic clustering. This phase outputs the dataset, which consists of images acquired during the exploration and the associated region labels.

2. **Training**, in which the dataset is used to train the neural network offline and in a static setting. The region labels become the prediction target of a classification process to associate images with regions. The output of this phase is the trained neural network.

3. **Inference**, in which the trained neural network is used to help the robot localization in successive navigations of the same environment, applying the temporal filter.

4. **Nodes filtering**, in which the output of the neural network passes through the temporal filter and is used to discard unlikely regions.

Fig. 3.2 shows the skeleton of the simplified process. Here, we separate between two clearly distinct stages: the first exploration of the environment, which comprises phases 1 and 2, and the successive explorations of the same environment, which comprises phases 3, and 4. Here, in step 2 the neural network is statically trained offline, avoiding all the complications of incremental learning. The neural network

training is based on traditional offline batch training on the entire dataset acquired in step 1. This further simplifies the training phase, because the region labels of the nodes cannot change at training time here, while it can happen if a node is reassigned to a different region in a continual scenario, as discussed in the next chapter. Once the neural network is trained, it can be used in successive explorations of the same environment as in step 3, to help the robot in long-term localization, allowing in step 4 to preselect the set of nodes to consider for relocalization.

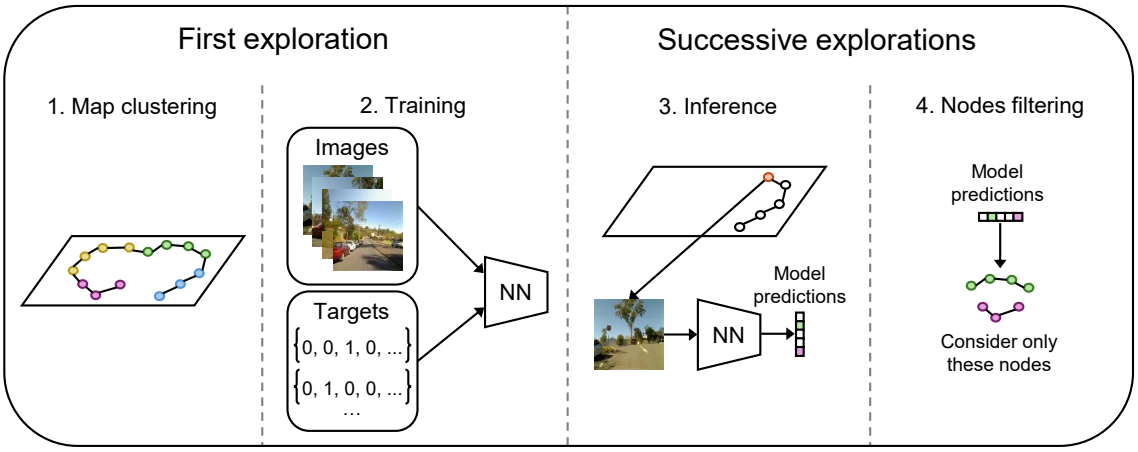


Figure 3.2: An overview of the system we proposed in [3]. During the first exploration of the whole map, map clustering (1) is performed to assign nodes to regions. At the end of the exploration, the region labels become the prediction targets of a deep neural network (2) trained to associate the images corresponding to the nodes with their respective regions. After training, the network is frozen and used during subsequent explorations of the same environment, operating in inference mode (3) to predict regions and filter nodes (4), thus reducing LCD computational complexity.

3.3 Dynamic map clustering

Map clustering plays a crucial role in the proposed approach: if the prediction targets were limited to individual nodes, the LCD process would lack robustness against the pose and view changes that occur when the robot revisits the same location. Furthermore, when the robot revisits the same place, new observations are associated with the corresponding region, thus further increasing tolerance to environmental changes.

As described in 2.6.2, in graph-based SLAM the map is represented as a connected graph $\mathcal{G}$ of n_v vertices V (or nodes) and n_a arcs A. Each node v is a robot pose (i.e., position and orientation) and each arc a_{ij} is a constraint (e.g., an odometry

measurement) between two nodes v_i and v_j:

$$V = \{v_i : 0 \leq i < n_v\}. \tag{3.1}$$

$$A = \{a_{ij} : 0 \leq i < n_v \wedge 0 \leq j < n_v\}. \tag{3.2}$$

Our method involves online incremental clustering of the map during exploration. We define a cluster (or region) $C_i = (V_i, A_i)$ as a connected subgraph of $\mathcal{G}$, where:

$$V_i \subseteq V, A_i = \{a_{jk} \in A : v_j \in V_i \wedge v_k \in V_i\}. \tag{3.3}$$

The clustering is exclusive, meaning that each node v_j belongs solely to one cluster C_i:

$$\forall C_i, C_j \in \mathcal{G}, V_i \cap V_j = \emptyset. \tag{3.4}$$

Inter-cluster arcs (i.e., $a_{ij} \in A : v_i \in V_r \wedge v_j \in V_s \wedge r \neq s$) do not belong to any cluster.

In our implementation, we adopted an algorithm first proposed in [114], which dynamically creates clusters during exploration by reducing the spatial scattering of the nodes within clusters. The scattering $s(\cdot)$ reflects the extent to which the nodes of a cluster are dispersed compared to the equivalent radius, which describes a cluster regardless of its actual shape, comparing it to an ideal high-density topology.

$$s(C_i) = s' + s''(C_i). \tag{3.5}$$

where s' is a threshold and s'' expresses to what extent the nodes of C_i are scattered with respect to the equivalent circle of C_i.

The algorithm relies on three parameters: (i) the desired cardinality η of the clusters, (ii) a shape factor ϕ, and (iii) the radius upper bound ρ of the equivalent circle. These parameters can be easily tuned to avoid too small or too large regions and to penalize irregular shapes. The algorithm is dynamic not only because it incrementally builds clusters during the exploration but also because it permits the reassignment of nodes between different clusters. A node v from C_i can be reassigned to a different cluster C_j if (i) reassignment does not violate the connectivity constraint, so that the removal of v from C_i does not break the connectivity of C_i and v is directly connected to a node of C_j and (ii) it reduces the total scattering, so that the reduction of the scattering of C_i due to its removal is greater than the increase of the scattering of C_j due to its inclusion. For example, if the robot is positioned at node v, which belongs to C_i, and it closes a loop with an already existing

node of C_j, then v is at the border between C_i and C_j. Therefore, the clustering algorithm checks if reassigning v to C_j reduces the total scattering.

Node reassignment facilitates the creation of compact and spatially coherent clusters as the robot navigates within the same region or returns to previously explored areas. Without such reassignment, cluster shapes would tend to be mostly linear, leading to potential overlap between different regions. Fig. 3.3 shows the benefit of node reassignment in the clustering obtained from the exploration of a real map. Although node reassignment improves cluster quality, it introduces challenges to the continual learning of the region prediction model. Changing a node's region assignment during subsequent explorations results in label changes for past data, which the training algorithm must tolerate. While label updates are feasible for future training, eliminating the impact of previous labels on the model's weights remains challenging. This is a peculiar case where a partial forgetting is not only unavoidable but even welcome.

Finally, observing the clustering results in Fig. 3.3 and Fig. 4.4 it is quite evident no "meaningful" region ending or borders are considered to create regions. Nodes are spatially grouped and, in unstructured maps, clusters form at arbitrary positions. In principle, this can lead to ambiguities close to the region borders because multiple views of the same place (e.g. a building) could be assigned to different neighboring clusters. However, this is not a limitation in practice because more than one region can be considered for LCD and this set often includes the most probable region and the neighboring ones (an example is provided in Fig. 3.6). To demonstrate the method robustness with respect to the region formation, in Section 4.6.2, we report results with different cluster granularities.

Here we described our particular implementation, but also totally different implementations of the clustering algorithm can be adopted; for example, in [95] and [96] the map is clustered depending on the visual feature similarities rather than the spatial location of the nodes.

3.4 Deep neural network

The ultimate goal of the neural network is to mimic the human ability to discard and consolidate hypotheses about the current location. For this purpose, it must learn to associate new observations (e.g., images) with the region they belong to and to infer the regions in which the robot is most likely to be given the current observation.

Our method is independent of the type of observations: it can be implemented

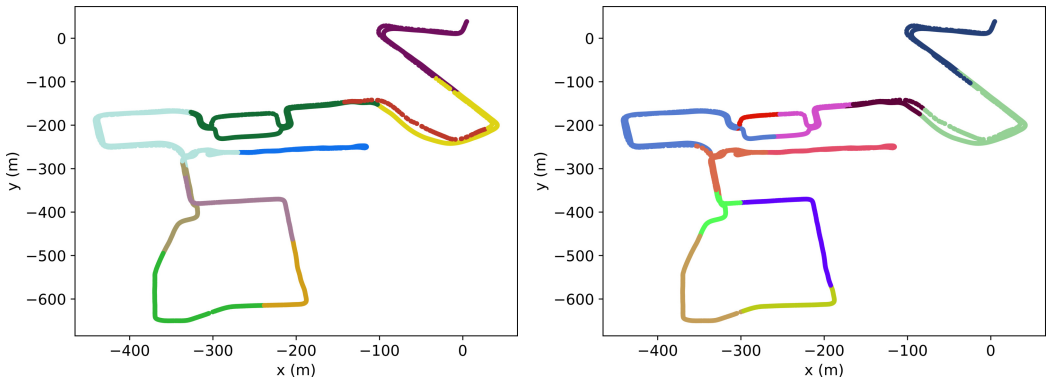


Figure 3.3: Map clustering using the same algorithm, without reassignment (a) and with reassignment (b). Different colors indicate different regions. The maps are created from the exploration of the Week1 sequence from the USyd campus dataset [115]. In (a), revisiting the same road (top-right portion of the map) results in two linearly shaped, nearly parallel clusters, as node reassignment is not performed. In contrast, (b) shows a single compact cluster representing that area.

using images, depth data or other observations of the robot. However, long-term localization is typically achieved through images, because they are richer in semantic meaning than depth data and therefore they are more effective in recognizing known places. For this reason, we use monocular RGB images as robot observations. To deal with images, we use a lightweight Convolutional Neural Network (CNN) as model f_Θ, which can be easily replaced with more complex models tailored to place recognition, such as NetVLAD [66] and RegionVLAD [84] or ResNet models pretrained on Places365 [116].

In our first work [3], a lightweight MobileNetV2 [117] pretrained on ImageNet and with frozen weights is used as a convolutional backbone to extract features from images, while a simple multilayer perceptron (MLP) is trained on top of the backbone as a classifier to predict region probabilities (see Fig. 3.4).

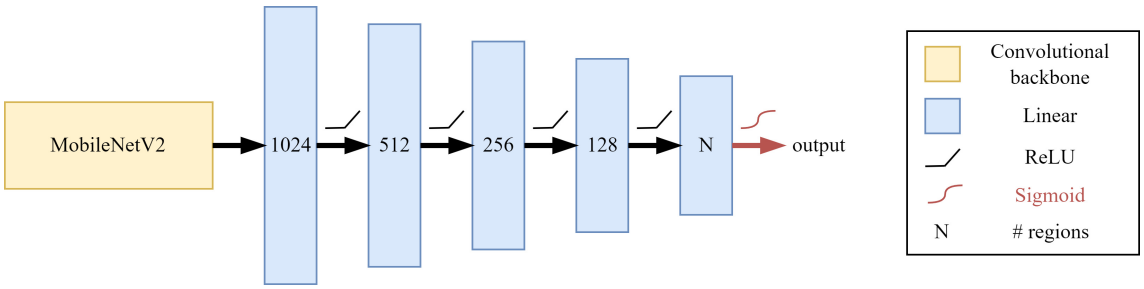


Figure 3.4: The architecture of the neural network used for region prediction.

Some additional precautions are implemented to adapt the learning process to

the specific application requirements:

- In the classifier's final layer, we replace the classical softmax function (which ensures class probabilities sum to 1) with a logistic function to enable high-confidence predictions for multiple regions. This adjustment is necessary for environments with sparse textures, such as corridors (see Fig. 3.5), where accurate predictions should reflect multiple regions.
- A focal loss function is used instead of the classical cross-entropy, as it better addresses highly unbalanced datasets [118].

(a) (b)

Figure 3.5: Two distinct corridors (from our Campus dataset) whose visual appearance is very similar. In such a case, our model should predict both regions with high confidence.

3.5 Temporal Filter

The temporal filter in our method stabilizes the probability estimates of the region, preventing abrupt changes. This mechanism mitigates the risk of the robot erroneously perceiving sudden, unrealistic shifts in its location, as if it had teleported, a condition that can be caused by inaccurate network predictions or when similar but distinct places are present. However, the temporal filter cannot be too strict, otherwise a correct relocalization would never occur; it must balance the impossibility of teleporting with the possibility of reobserving a known place after a long time. The output of the neural network f_Θ at time t is a vector of probabilities for the clusters, $P(C)_t$, in which each entry i represents the probability for the i^{th} region. This vector passes through the temporal filter, becoming a vector of weighted probabilities $P_w(C)_t$, used to select the top-k regions to consider, that are the k regions with the highest weighted probabilities, where k is a parameter.

In our implementation, an exponential moving average is applied, enhancing temporal consistency. The weighted probability $P_w(C_i)_t$ for the region i at time t depends on the current observation probability $P(C_i)_t$ and the probability of the previous step $P(C_i)_{t-1}$, weighted by a parameter α (Equation 3.6).

$$P_w(C)_t = \alpha P(C)_t + (1 - \alpha)P(C)_{t-1}. \tag{3.6}$$

When $\alpha = 1$, no temporal filtering is applied. In this case, the robot can relocalize more quickly, but this also increases the likelihood of frequent switches between different regions across consecutive iterations, making the transfer–retrieval process more computationally demanding, even if not needed in many cases. Empirically, we found that setting $\alpha = 0.8$ provides a good balance between relocalization speed and stability, effectively reducing excessive region switching.

Future works involve the adoption of a Bayes filter instead of the exponential moving average to further improve the effectiveness of the temporal filter.

3.6 Integration into RTAB-Map

In essence, our system allows us to sharply distinguish between two sets of nodes in the graph: the nodes of the map to consider for long-term localization from those to not consider. This fits very well with the memory management system already implemented into RTAB-Map, where the Working Memory (WM) is a buffer for the nodes involved in loop closure detection and graph optimization, while Long-Term Memory (LTM) is a database for the other nodes in the graph (see 2.4.1). For this reason, we decided to embed our method into RTAB-Map, but it can be embedded in any graph-based SLAM system as long as a similar memory mechanism is also implemented. In fact, the native RTAB-Map components responsible for odometry estimation, loop closure detection, and map optimization, which are very common among graph SLAM solutions, remain unchanged. Beyond the memory management aspects, we chose RTAB-Map also because it is a complete SLAM framework with support for a wide range of sensors, including inertial, GPS, stereo, depth, and LiDAR-only configurations. It integrates multiple solutions within a unified architecture, ranging from single-camera setups to multi-camera systems with up to six cameras, and relies on different backend optimization libraries. As a result, it represents a highly versatile research tool for exploring and identifying the most suitable solution or configuration for a given scenario, adapting effectively to diverse operational requirements.

Only the RTAB-Map transfer-retrieval process was extended to include region

prediction, as detailed in Algorithm 1. In the original version, at each iteration, the highest loop closure hypothesis v_h is selected from the nodes in WM (line 4). Then, two disjoint sets of nodes are retrieved (lines 5-8): N_1, comprising n_{k_1} nodes in LTM that are spatial or temporal neighbors of v_h, and N_2, consisting of n_{k_2} nodes already in WM that are neighbors of v_h. Here, n_{k_1} and n_{k_2} are parameters, and $N_1 \cap N_2 = \emptyset$. Our approach introduces an additional set, N_3, of retrieved nodes. Thanks to the neural network and the temporal filter, nodes in N_3, not already in $N_1 \cap N_2$, are retrieved into WM if they belong to the top-k most probable regions according to $P_w(C)_t$. At the end of the process, nodes in WM $\notin N_1 \cup N_2 \cup N_3$ are transferred back to LTM (lines 15-19). Depending on the platform's computational capabilities, different numbers (k) of regions can be moved to WM. In our experiments, retrieving the top-3 most likely regions provides a good balance between loop closure detection and RTAB-Map processing time. An example in Fig. 3.6 shows the top-3 regions predicted during the retracing of a KITTI [119] sequence.

The proposed transfer-retrieval approach provides three significant improvements to RTAB-Map:

1. The time required for LCD and graph optimization remains constant regardless the map size, as only a subset of nodes is kept in WM.

2. The recall of LCD increases, especially in cases where places are reached by unknown long paths, a scenario where both spatial and temporal closeness would fail to keep the target nodes in WM.

3. The precision of LCD potentially increases because the least likely regions are transferred back to LTM and not considered for detection, thereby reducing the probability of false positives.

Actually, (1) and (3) are also achievable by simply limiting the size of the WM in the original version of RTAB-Map, but at the cost of reducing the loop closure detection recall. In contrast, our method even improves the recall compared to the original RTAB-Map (2).

3.7 Experiments design

Two types of experiments have been designed to assess the validity of the proposed approach. The former is aimed at estimating the ability of the neural network to predict the correct region when the robot revisits known places. The latter investigates the accuracy of loop closure detection by comparing the original version

Algorithm 1 Modified RTAB-Map Transfer-Retrieval

procedure WM_UPDATE(v)
  $f_\Theta \leftarrow$ trained model
  $x \leftarrow$ observation of v // RGB image
  $v_h \leftarrow$ highest loop closure hypothesis in WM
  $N_1 \leftarrow n_{k_1}$ neighboring nodes (in time and space) of
    v_h in LTM
  $N_2 \leftarrow n_{k_2}$ neighboring nodes of v_h in WM
  retrieve($N_1 \cup N_2$)
  $P(C)_t \leftarrow f_\Theta(x)$ // region prediction
  $P(C)_{t-1} \leftarrow$ region probabilities at previous iteration
  $P_w(C)_t \leftarrow \alpha P(C)_t + (1-\alpha)P(C)_{t-1}$ // Eq. 3.6
  $R_t \leftarrow$ top-k most likely regions according to $P_w(C)_t$
  $N_3 \leftarrow$ nodes in LTM whose region $\in R_t$
  retrieve(N_3)
  for each v_i in WM **do**
    if $v_i \notin N_1 \cup N_2 \cup N_3$ **then**
      transfer(v_i)
    end if
  end for
end procedure

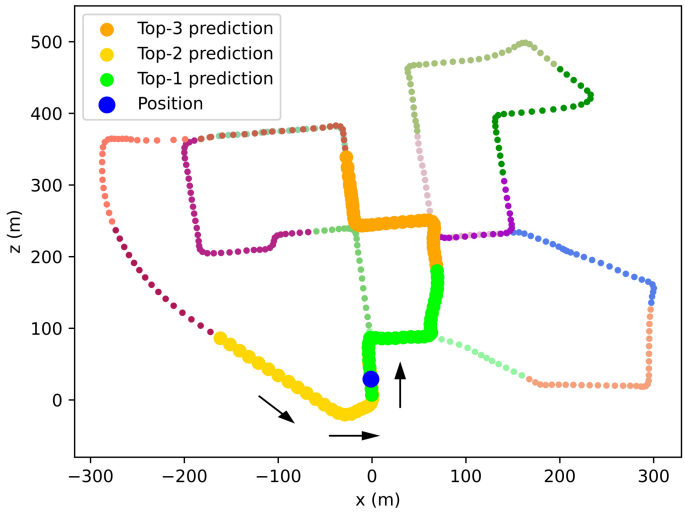


Figure 3.6: Model predictions on the sequence 00 of the KITTI odometry dataset [119]. The figure highlights the top-3 predicted regions from the network when the robot (indicated by the blue dot) revisits a known location.

of RTAB-Map with the modified one. To this purpose, we report both the number of loop closures detected and the corresponding increase in overall pose accuracy.

For the experiments, we used three datasets. The first two are well known and widely used in SLAM:

- **OpenLoris**: the OpenLoris-Scene dataset [112], specifically created for life-long SLAM, with different indoor sequences with variations in environmental conditions and with the presence of moving people introducing further variability and occlusions;

- **KITTI**: the KITTI odometry dataset [119], an outdoor dataset including 22 sequences (11 with ground truth to tune/train the algorithms and 11 without ground truth to test them) acquired with stereo cameras and a 3D velodyne laser in urban and rural settings.

A third dataset (denoted as **Campus**) was set up using sequences captured with the Xaxxon autocrawler robot [120] equipped with a 2D rotating lidar and an Intel D435 depth camera (see Fig. 3.7), navigating in three corridors of our university campus. Other datasets are available for visual place recognition (e.g. St Lucia Multiple Times of Day [121] and Nordland Dataset [122]). However, we cannot use them because of the lack of stereo or RGB-D data, which are necessary to perform visual-odometry estimation in a complete SLAM system.

For the experiments performed in our work, we only used RGB data for region prediction.

Figure 3.7: The Xaxxon autocrawler robot used to acquire the Campus dataset.

Sequences	Top-1 accuracy	Top-3 accuracy
OpenLoris/Market 1-2	0.81	0.92
OpenLoris/Market 1-3	0.79	0.93
OpenLoris/Corridor 1-2	0.27	0.45
OpenLoris/Corridor 1-3	0.06	0.14
OpenLoris/Corridor 1-4	0.17	0.44
OpenLoris/Corridor 1-5	0.65	0.90
Campus/Corridor-Biolab 1-2	0.53	0.79
Campus/Corridor-DEI 1-2	0.47	0.77
Campus/Corridor-Arc 1-2	0.33	0.76

Table 3.1: Top-1 and top-3 region prediction accuracy. The numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y.

3.7.1 Prediction Accuracy

For the first type of experiments, we used Campus and two OpenLoris packages, where the robot navigates the same locations in different sequences:

- the **Market** package, comprising three sequences captured inside a large marketplace, with relatively consistent lighting conditions across sequences;
- the **Corridor** package, comprising five sequences captured along a single corridor with large windows, with strongly varying lighting conditions (including artificial illumination turned off).

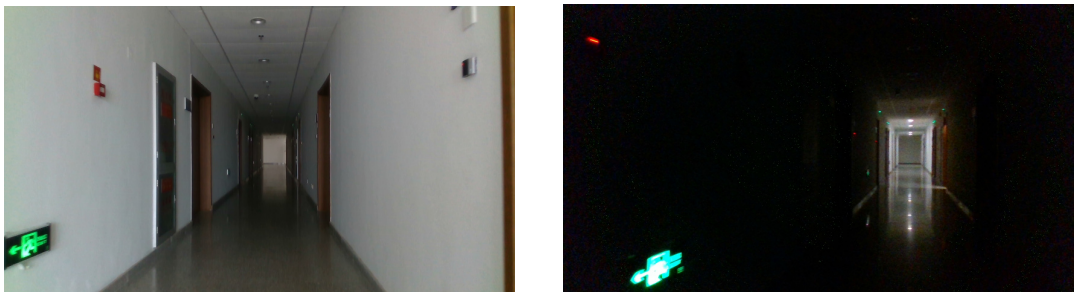
KITTI was not used here because of the lack of sequences revisiting the same locations. For OpenLoris and Campus, the first sequence is used to train the neural network and the others for testing. The pairing of the regions between the training and test sequences was manually identified, by visually matching the images and assigning consistent region labels.

The top-1 and the top-3 accuracy metrics are used for these experiments because they highlight the network's ability to predict the correct region and to capture similarities between the different regions.

Table 3.1 shows the results. In the OpenLoris Market and Campus Corridors sequences, the top-3 accuracy is good, indicating that the correct region is almost always among the three most likely predictions and therefore its nodes have a high chance of being retrieved in WM. Sometimes incorrect regions may be predicted as top-1, however, this behavior is even desirable in some cases, to deal with region similarities.

The prediction accuracy is not as good on the Corridor sequences of OpenLoris, due to the strong variation in the lighting conditions (see the example in Fig. 3.8).

To deal with such variation a continual learning of region prediction would be necessary as discussed in Section 3.8.



(a) (b)

Figure 3.8: On the left, (a) depicts an acquisition from the first corridor sequence of OpenLoris, captured under good lighting conditions. On the right, (b) shows the same corridor in the fourth sequence, with artificial illumination turned off.

3.7.2 Loop Closure Detection

In the experiments of loop closure detection, in addition to the OpenLoris Market package and Campus sequences, we also used KITTI sequences which contain loop closures, namely sequences 00, 02, 05, 06, 07 and 09, (loop closures are not present in the other sequences). Since the ground truth robot pose is known for KITTI and OpenLoris, we implemented a simple algorithm to match poses across the sequences, in order to identify correspondences (and then loop closure candidates) in an automatic way. In particular, if the distance between the (x, y) location in the 2D plane and the angle (also in the 2D plane) between the two poses are less than two given thresholds, then the two poses are considered as a match.

In these experiments we made a fundamental assumption: in explorations of very large environments (potentially the whole world) only a very small subset of map nodes can be considered for loop closure detection and map optimization to avoid violating the real-time constraint. Since the datasets here do not provide very large exploration, we simulated it by limiting the WM size of RTAB-Map to a very low fixed number of nodes (instead of fixing the number k of regions retrieved); the total WM size was limited to $n_{WM} = 50$ nodes (approximately 15% of the average size of the sequences considered). Referring to the Algorithm 1, the parameters n_{k_1} and n_{k_2} were set to default RTAB-Map values of 2 and 0.25 (that is, $25\% \times n_{WM} = 12$ neighboring nodes in WM cannot be transferred to LTM, avoiding the transfer of too recent nodes), respectively. According to the constraints, at each iteration, only $n_{WM} - n_{k_1} - n_{k_2}$ nodes $\in N_3$, which corresponds to 36 nodes, a very low number.

As shown in Table 3.2, despite the strong constraints applied, our approach was able to recognize all the loop closure in all the sequences considered (except one) totaling 95% of loop closure detection, even with a very limited WM. On the contrary, the default RTAB-Map memory management was able to detect loop closure only 15% of times because the spatial and temporal closeness criteria can be unsatisfactory when navigating large maps.

OpenLoris sequences often have no loop closures within the single sequence, but in different sequences the robot returns to the same places and relocalization can be performed (see Table 3.3). To deal with these cases, RTAB-Map allows reloading all memory in the WM when a new sequence is started or reloading only the WM as it was at the end of the previous exploration. To keep the WM limited to 50 nodes, the latter option was chosen.

The common metric used to evaluate loop closure detection is the Precision-Recall metric [123]. Tables 3.4 and 3.5 report Precision-Recall for loop closure detection and relocalization, respectively. The proposed method, compared to the RTAB-Map baseline, greatly increases the recall in the large maps scenario. Furthermore, the precision does not worsen, as the loop closure detection process is not modified. On the contrary, precision could increase because only nodes belonging to regions predicted with high probability are considered, discarding other observations that could lead to false positives.

Sequences	**N° Loop closures**	**Loops detected**	
		RTAB-Map baseline	**Our**
KITTI/00	4	1	**4**
KITTI/02	2	0	**2**
KITTI/05	3	0	**3**
KITTI/06	1	0	**1**
KITTI/07	1	1	1
KITTI/09	1	0	**1**
Campus/Corridor-Biolab 1-2	2	0	**2**
Campus/Corridor-DEI 1-2	2	0	**2**
Campus/Corridor-Arc 1-2	3	1	**2**
OpenLoris/Market 1-3	1	0	**1**
Total	20	3 (15%)	**19 (95%)**

Table 3.2: Loop closure detection results. For Campus and OpenLoris, the numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y.

Finally, it is worth noting that even if our node preselection was perfect (i.e., ensuring that the right nodes are always in WM), the native loop closure algorithm of the SLAM used could miss some closures because of insufficient similarity of the

Sequences	N° Relocalization	Relocalizations performed	
		RTAB-Map baseline	Our
OpenLoris/Market 1-2	3	1	**3**
OpenLoris/Market 1-3	3	1	**3**
Total	6	2 (33%)	**6 (100%)**

Table 3.3: Relocalization results. The numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y.

Sequences	RTAB-Map baseline		Our	
	Precision	Recall	Precision	Recall
KITTI/00	1	0.25	1	**1**
KITTI/02	-	0	**1**	**1**
KITTI/05	-	0	**1**	**1**
KITTI/06	-	0	**1**	**1**
KITTI/07	1	1	1	1
KITTI/09	-	0	**1**	**1**
Campus/Corridor-Biolab 1-2	-	0	**1**	**1**
Campus/Corridor-DEI 1-2	-	0	**1**	**1**
Campus/Corridor-Arc 1-2	1	0.33	1	**0.66**
OpenLoris/Market 1-3	-	0	**1**	**1**

Table 3.4: Precision-Recall on loop closure detection. For Campus and OpenLoris, the numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y.

Sequences	RTAB-Map baseline		Our	
	Precision	Recall	Precision	Recall
OpenLoris/Market 1-2	1	0.33	1	**1**
OpenLoris/Market 1-3	1	0.33	1	**1**

Table 3.5: Precision-Recall on relocalization. The numbering x-y at the end of each sequence name means that the predictor was trained on sequence x and tested on y.

corresponding signatures. A posterior analysis of the results showed that the only loop missed by our approach in Table 3.2 falls into this category.

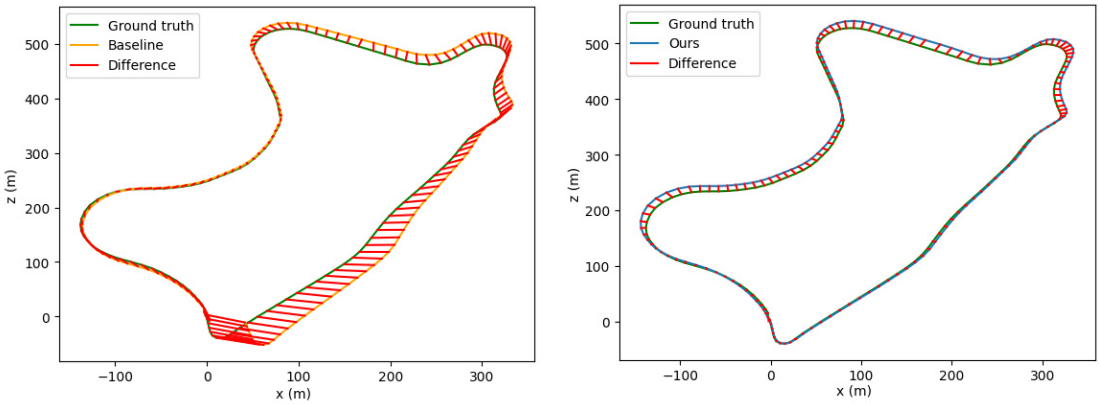
3.7.3 Pose estimation accuracy

Since loop closure detection is critical for optimizing the map and reducing localization error, we also report the accuracy of pose estimation on different sequences of the KITTI odometry dataset. The absolute trajectory error (ATE) reported in Table 3.6 highlights that even on small maps (such as the KITTI sequences) where the drift is not so large, the error correction achieved through loop closure detection has a strong impact on pose accuracy, especially in presence of large loops (e.g.,

sequence 09, see Fig. 3.9). Here, ATE is computed on the graph at the latest WM state, no global optimization is performed a posteriori.

Sequences	**ATE(m)**	
	RTAB-Map baseline	**Our**
KITTI/00	4.754	**3.107**
KITTI/02	6.891	**6.080**
KITTI/05	2.188	**1.946**
KITTI/06	3.791	**1.811**
KITTI/07	1.116	**1.079**
KITTI/09	13.222	**5.623**

Table 3.6: Absolute trajectory errors results. Both the baseline and our method have to be considered with the WM size limited to 50 nodes. For the baseline, the ATE is higher because fewer loop closures are detected, while for our method it decreases because all loop closures are detected. Since the entire map cannot be corrected due to the limited WM size, results are different from those reported in the original RTAB-Map paper [5].



(a) (b)

Figure 3.9: Absolute trajectory error (ATE) on sequence 09 of the KITTI odometry dataset. On the left, the RTAB-Map baseline does not detect any loop closure, on the right, our method detects a loop closure, reducing the ATE visibly.

3.8 Conclusions

In this chapter, we introduced a region prediction technique, where map clustering is used to partition nodes into regions and a deep model is trained to predict the region that the robot is currently navigating. This approach enables an efficient

implementation of place recognition for real-time loop closure detection. Our experiments validate the effectiveness of the idea but leave some issues open. In fact, in this preliminary work, the proposed technique cannot be entirely performed in real-time, because the predictor is trained offline on the whole accumulated map knowledge.

To overcome this bottleneck, the natural evolution is to move toward an online continual learning formulation, where the model incrementally acquires knowledge of new regions and new instances of previously seen regions (e.g., day/night variations), without forgetting past knowledge.

Addressing this challenge requires effective strategies to avoid catastrophic forgetting while ensuring scalability. In the next chapter, we present a continual learning extension of the proposed method, based on latent replay [124] and complementary mechanisms, and demonstrate how it enables real-time, lifelong operation when integrated into a full SLAM system.

Chapter 4

Continual Learning of Regions for Efficient Robot Localization on Large Maps

"Birds have an uncanny ability to find their way across thousands of miles. They do it without maps, but not without memory."

In this chapter, we address the considerably more complex problem of training a region-prediction model in a continual learning fashion as the map is explored: (i) discovering new regions, and (ii) revisiting and updating known regions due to environmental changes [4]. This is the natural evolution of our previous work, and it represents the complete concretization of our methodology. Here, there is no more a clear distinction between exploration and inference phase: the system must recognize at any time if it is facing a known or a new place. The proposed approach is particularly suited to scenarios requiring truly long-term navigation over extended periods of time (hours or days), where the system must gradually adapt to varying illumination conditions and changes in environmental configurations. Outdoor scenarios, such as large-scale urban mapping, are expected to benefit the most from this approach. Large-scale dynamic indoor environments may also be considered, provided that they span substantial areas (e.g., navigation across multiple buildings); otherwise, alternative techniques may already sufficiently address these requirements.

4.1 Deal with Continual Learning

The problem addressed by our method falls into the NIC category (see Section 2.1.2), since new regions are discovered incrementally during exploration, and new nodes of existing regions may also be identified. However, a specific peculiarity of our implementation is that the label of a data point can change over time, because a node can be reassigned to a different cluster during exploration (as explained in Section 3.3).

Recalling Equation 2.3, given a classification model f parametrized by Θ, train it incrementally (to minimize the loss $\mathcal{L}$) over a sequence of experiences CL:

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(f_\Theta(E_i^x), E_i^y) \;\; for \;\; i = 1, ..., n_e$$

leads to a progressive forgetting of the data belonging to past experiences. To mitigate this problem, in our implementation a replay memory $\mathcal{R}$ of fixed capacity n_{replay} is used, where a subset of past data is stored and replayed jointly with the samples in the current experience.

$$\Theta^* = \arg\min_{\Theta} \mathcal{L}(f_\Theta(E_i^x \cup \mathcal{R}_i^x), E_i^y \cup \mathcal{R}_i^y) \;\; for \;\; i = 1, ..., n_e \tag{4.1}$$

where $\mathcal{R}_i^x$ and $\mathcal{R}_i^y$ are data and labels, respectively, contained in the replay memory during the training on the experience E_i. Actually, to improve efficiency and save storage, our approach leverages latent replay (details are provided in Section 4.2.2) where latent representations of the replay data are stored in $\mathcal{R}$ rather than the corresponding raw inputs. The replay memory updates at the end of each experience by adding a subset of samples from the current experience.

4.2 The Proposed Continual SLAM

This Section provides a top-down explanation of the proposed approach, beginning with an overview of the whole system and then detailing the individual components.

4.2.1 High-level Overview

Fig. 4.1 illustrates a functional schema of the method: the main loop of the SLAM process is shown on the left, while an incremental training step of the region prediction network is depicted on the right. As the robot moves to a new position (1), new data is acquired from the sensors (including an RGB image), the odometry is computed, and both a new node v_i (white in the figure) and a new arc $a_{p,i}$ are added

to the graph. The RGB image is then processed by the current region prediction model (2), which outputs a vector indicating the candidate regions where the robot might be located. Nodes belonging to the top-k regions are then moved from LTM to WM. Then, nodes with a low probability of being the target of the next LCD are transferred back into LTM to keep WM size bounded (see Algorithm 1 in Section 3.6). During step (3), LCD is executed on WM nodes to determine whether v_i represents a new place on the map or whether the robot returned to a known location. If a loop is detected, a further arc $a_{i,q}$ (red in the picture) is added to the map, and graph optimization is launched. In step (4), incremental clustering assigns v_i to one of the existing regions or a new region, and neighboring nodes may be reassigned if necessary (see Section 3.3). Note that clustering occurs after graph optimization because node positions can change during the optimization process, affecting clustering results. In step (5), the RGB image x and the corresponding region label y are added to the current experience E_c. The labels of other data in E_c are updated if the clustering algorithm reassigns the corresponding nodes to different regions. When the experience buffer reaches its maximum capacity (a predefined parameter), an incremental training step of the region prediction model is launched concurrently to avoid delaying real-time navigation.

When a new training step is triggered, the model architecture may need to be adapted (6) if the current experience includes data from new classes. In this case, new neurons (one for each new class) are added to the network's last layer. Training then begins (7): latent representations of the data in the current experience E_c are extracted and combined with data from the replay memory. These are then fed into the model for a fine-tuning session. Finally (8), the replay memory is updated and the new model is deployed for future predictions. More details are provided in Section 4.2.2.

4.2.2 Region Prediction

Region prediction is performed through a deep neural network classifier trained to map images to region labels. For this purpose, one may employ an ad hoc architecture like NetVLAD [66] or RegionVLAD [84] that are designed for robust place recognition. However, to optimize efficiency, in our work we adopt a lightweight Convolutional Neural Network (CNN) model, f_Θ. Specifically, we select ResNet18, as larger ResNet models (e.g., ResNet50) do not support real-time training and inference on real-world robotic platforms where a GPU is not always available. The ResNet18 architecture can be virtually partitioned into three distinct sets of layers,

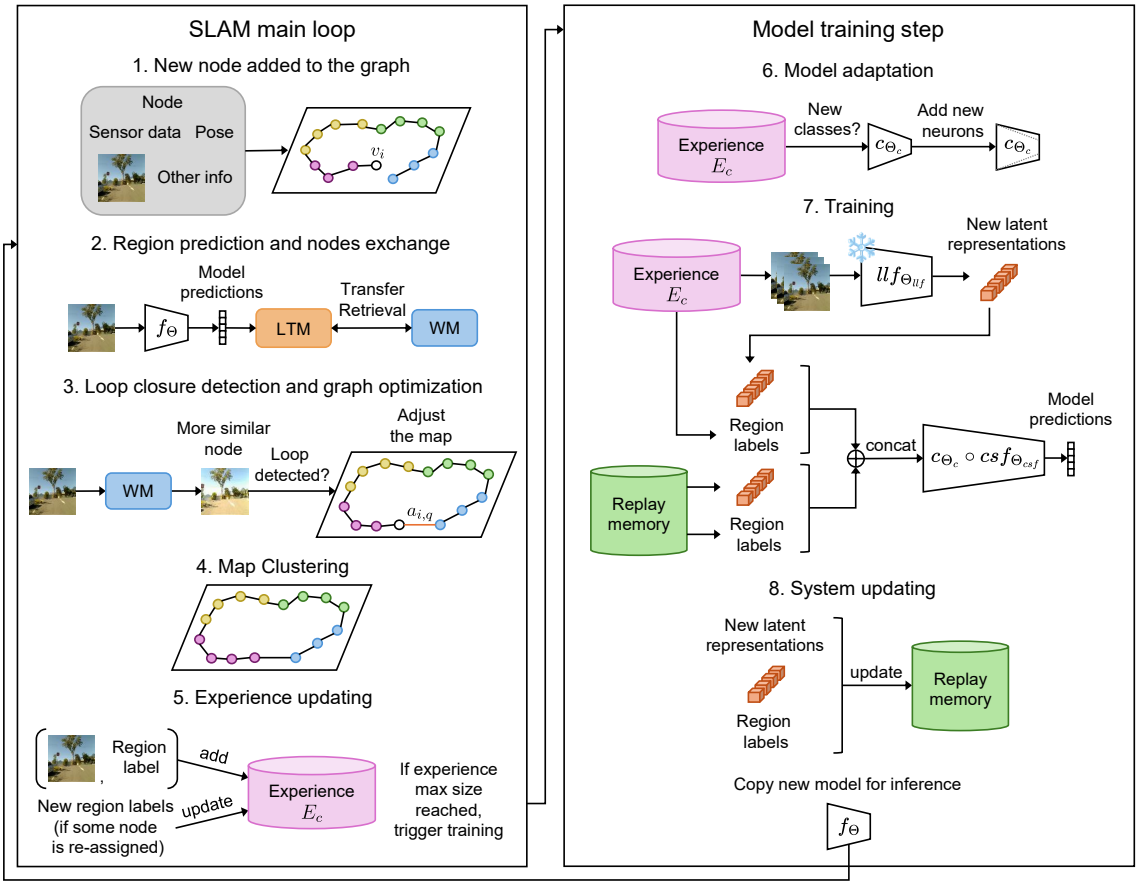


Figure 4.1: A functional schema of the proposed approach. f_Θ is the region prediction model that can be decomposed in the modules $c_{\Theta_c} \circ csf_{\Theta_{csf}} \circ llf_{\Theta_{llf}}$ (see Eq. 4.2) where: c_{Θ_c} is the classification head; $csf_{\Theta_{csf}}$ is the class-specific feature extraction module, and, $llf_{\Theta_{llf}}$ is the low-level feature extraction module. During the main SLAM process (left side of the figure) f_Θ is used to retrieve in the working memory (WM) nodes that could be the target of LCD; dynamic clustering is then performed and representative data are stored in the current experience E_c for the next model update. When a model update is triggered (right side of the figure) data from the E_c are mixed with replay data at the latent representation level and the modules c_{Θ_c} and $csf_{\Theta_{csf}}$ are fine-tuned. $llf_{\Theta_{llf}}$ is decorated with a snowflake in the figure to denote that its weights are kept frozen. A detailed step-by-step description of the workflow in this figure is provided in the main text of Section 4.2.1.

which can be composed in functional notation as:

$$f_\Theta = c_{\Theta_c} \circ csf_{\Theta_{csf}} \circ llf_{\Theta_{llf}}, \tag{4.2}$$

where:

- llf denotes the set of layers parametrized by Θ_{llf}, which extract low-level features and include the first 4 ResNet blocks, consisting of 12 convolutional layers.

- csf denotes the set of layers working on top of llf, extracting class-specific features parametrized by Θ_{csf} and includes the last ResNet block, consisting of 4 convolutional layers.

- c denotes the final classification layer parameterized by Θ_c.

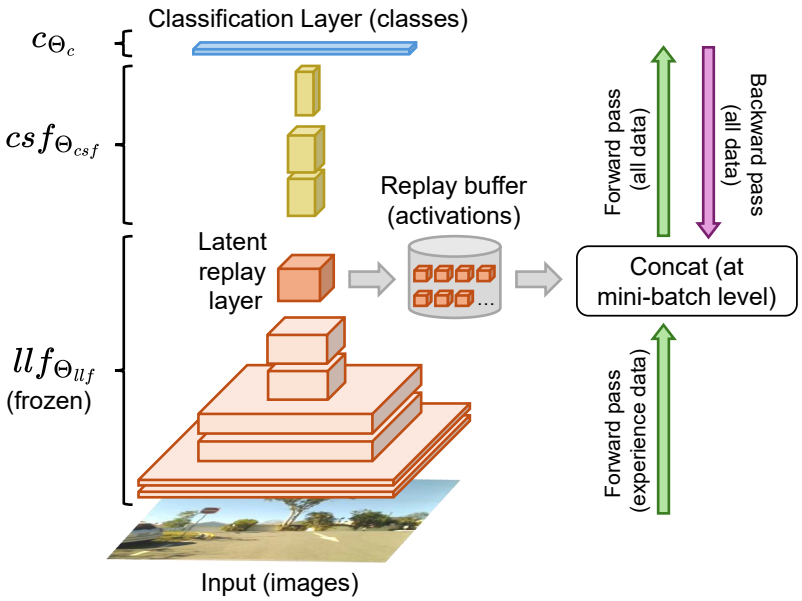


Figure 4.2: Latent replay architecture and workflow: the latent replay layer is the last of llf. A subset of the activations computed by this layer is stored in a fixed capacity replay buffer and later replayed together with data from new experiences to control forgetting.

ResNet18 is pre-trained on Places365 [116], a dataset for scene recognition with ∼1.8 million images of places. Consequently, the feature extractor weights Θ_{llf} are reasonably well-tuned for our task, and during continual learning, these weights remain frozen while Θ_{csf} and Θ_c are fine-tuned. This architectural partitioning, where the latent replay level is defined as the last layer of a frozen backbone, is crucial for latent replay (see Fig. 4.2). With Θ_{llf} fixed, the activations stored in the replay buffer remain valid for subsequent updates. Algorithm 2 details the continual learning process for a single experience E_i. The replay memory $\mathcal{R}$ is populated (line 17) using a class-balanced reservoir sampling policy [125, 18], maintaining an equal number of examples for each class seen so far and ensuring a proper balance of past and new examples in the buffer.Reservoir sampling is a randomized algorithm designed to select a fixed-size sample of k elements from a data stream of unknown or potentially unbounded length, ensuring that each element has equal probability of being included. The algorithm maintains a reservoir of k samples: the first k elements are inserted directly, while each subsequent element (at position $i > k$)

Algorithm 2 Continual learning step

procedure TRAIN(E_i)
  $llf_{\Theta_{llf}} \leftarrow$ low level feature extractor
  $csf_{\Theta_{csf}} \leftarrow$ class specific feature extractor
  $c_{\Theta_c} \leftarrow$ classification head
  $f_\Theta \leftarrow$ model // $c_{\Theta_c} \circ csf_{\Theta_{csf}} \circ llf_{\Theta_{llf}}$
  $n \leftarrow$ number of epochs
  $\mathcal{R} \leftarrow$ latent replay buffer
  $\mathcal{L} \leftarrow$ loss function
  $f_\Theta \leftarrow$ **adaptation**(f_Θ, E_i) // add new neurons in
      the last layer if E_i contains new classes
  $F_i^x \leftarrow llf_{\Theta_{llf}}(E_i^x)$ // extract features until latent
      replay layer
  $(\mathcal{R}_x, \mathcal{R}_y) \leftarrow$ sample from $\mathcal{R}$
  $(mb_x, mb_y) \leftarrow$ concat $(F_i^x, \mathcal{R}_x), (E_i^y, \mathcal{R}_y)$
  train $c_{\Theta_c} \circ csf_{\Theta_{csf}}$ on (mb_x, mb_y) for n epochs
      with loss function $\mathcal{L}$
  $\mathcal{R} \leftarrow$ **update replay memory**(F_i^x, E_i^y)
end procedure

replaces an existing element in the reservoir with probability k/i. This approach guarantees uniform sampling without requiring storage of the entire stream, making it particularly suitable for streaming and large-scale data processing scenarios.

4.3 Experiments Design

To validate the proposed method, two types of experiments have been designed:

- **E1 (Continual learning of regions)** is aimed at demonstrating the effectiveness of the continual learning strategy in incrementally learning visual information about places and associating them with regions. It requires datasets where the same (or part of the same) path is retraced multiple times, and a specific evaluation workflow is necessary, as explained in 4.4.1.

- **E2 (Continual SLAM)** evaluates the efficacy and efficiency of the proposed approach when embedded into a complete SLAM solution like RTAB-Map. For this purpose, we run RTAB-Map (modified as described in Section 4.2) on several sequences of the datasets detailed in 4.3.1. The focus here is on the classical LCD precision-recall metric, the accuracy of the generated map and time efficiency.

The details of the experimental setting in E1 and E2, including the split between the train/test sequences, the order of the sequence, and the value of hyperparameters are reported in the Appendix A.

In general, we tried to keep the clustering parameters fixed across different environments; this was possible for outdoor experiments, but not for indoor datasets which cover smaller maps, and the number of resulting clusters would be too small. The same principle was used for training parameters (learning rate, mini-batch size, number of epochs, etc.) which, in both joint and incremental training, were fixed across experiments. We empirically observed that ∼25 samples per region in the replay memory are sufficient to control forgetting. Considering that we are storing latent representations whose size is about 200KB each, the total space required by the replay buffer is 200KB × ∼25 samples × number of classes; in our largest experiments (St. Lucia) this amounts to ∼400 MB.

All experiments are conducted on a workstation equipped with an Intel I9-11900K CPU, an NVIDIA Quadro RTX A4000 GPU, and 64 GB of RAM. The continual learning component was initially developed in Python and validated using Avalanche [126], a popular continual learning library for PyTorch [127]. For efficient integration into RTAB-Map, the implementation was then migrated to C++ using LibTorch, the C++ porting of PyTorch.

4.3.1 Datasets

We employ four different datasets in our experiments, including three well-known and widely used datasets for SLAM:

- USyd campus [115], an outdoor dataset, originally proposed for semantic segmentation, where the same path around the Sydney campus is retraced 52 times, once for every week of the year, capturing the same trajectory under different weather and lighting conditions, with the presence of moving vehicles and people introducing further variability and occlusions. No position ground truth is available.

- St. Lucia Multiple Times of Day [121], an outdoor dataset of monocular images with a GPS ground truth where a very large urban path is retraced 10 times at different times of the day.

- OpenLoris-Scene [112], a dataset specifically designed for lifelong SLAM, including different indoor sequences with variations in environmental conditions and with the presence of moving people introducing additional obstructions

and variability. In particular, for E1 experiments we focus on *corridor* sequences that are particularly challenging since they also include sequences in the dark (with artificial illumination turned off), while for E2 we use the *market* sequences because they contain real (intra-sequences) loop closures. The frames have an associated position ground truth computed using laser odometry.

- KITTI odometry [119], an outdoor dataset including 22 sequences (11 with ground truth to tune/train the algorithms and 11 without ground truth to test them) acquired with stereo cameras and a 3D velodyne laser in urban and rural settings.

We used USyd and OpenLoris-Scene datasets for both E1 and E2 experiments. St. Lucia was used only in E1 due to the lack of stereo or RGB-D images, which precludes its use with RTAB-Map. On the contrary, KITTI has been tested only in E2 because the lack of retraced paths makes it unsuitable for E1 experiments. Table 4.1 summarizes the dataset-experiments mapping, while Fig. 4.3 provides examples of images from the four datasets.

We believe that the four selected datasets are sufficiently dynamic for this study, and we are not aware of any other (more dynamic) dataset that offer multiple retracing of the same path, which is a crucial requirement for our experiments. Evidence of dynamism can be observed by looking at the accuracy trend over experiences in Fig. 4.5: the learning curves clearly show that the complementary information (or variation) learned at each retracing constantly improves place recognition on kept-out sequences.

Dataset	E1	E2	Position ground truth	Data available	Total RGB frames
USyd Campus	X	X		RGB, 3D laser point clouds	1,007,500 (26 seq. used in E1) 465,496 (12 seq. used in E2)
St. Lucia	X		X	RGB	203,536
OpenLoris-Scene	X	X	X	RGB-D, 2D laser point clouds	20,357 (corridors seq. used in E1) 22,618 (market seq. used in E2)
KITTI		X	X	Stereo RGB, 3D laser point clouds	16,027 (7 seq. used in E2)

Table 4.1: Datasets used in the experiments. Total RGB frames denote the number of frames in the sequences used.

4.3.2 Comparisons

We compare the proposed approach with other existing and baseline methods:

- For E1, we compare the incremental strategy with a joint training baseline. This baseline assumes that all the training data are available for a single global

(a) USyd (b) St. Lucia

(c) OpenLoris (d) KITTI

Figure 4.3: Two example frames from each dataset. Images (a) from the USyd dataset illustrate an outdoor environment featuring people and vehicles. Images (b) from the St. Lucia dataset depict an urban environment under varying lighting conditions. Images (c) from the OpenLoris dataset (corridor sequences) emphasize the presence of occlusions, such as people, and differing lighting conditions. Lastly, images (d) from the KITTI dataset showcase diverse urban environments.

batch training session (that can be conceived as a single large experience) and whose performance is usually considered a sort of upper bound for achievable CL accuracy.

- For E2, we compare the proposed approach with the unmodified RTAB-Map, considering both unbounded and limited WM and our previous approach based on offline learning [3] (see Chapter 3). Furthermore, we compared our method with ORB-SLAM3 [77], a state-of-the-art graph-based SLAM. We also considered DROID-SLAM [54] and DPVO [55], which are modern deep learning-based solutions.

It is worth noting that place recognition approaches such as NetVLAD [66] and RegionVLAD [84] are not competing methods and can be easily integrated into our approach by simply replacing the NN architecture (ResNet18) used for region prediction. We are confident that the complete release of the proposed approach and the associated experimental setup will enable other researchers to more easily compare future developments in continual SLAM.

4.4 E1 experiments

4.4.1 Evaluation Workflows

In these experiments, we concentrate exclusively on the CL accuracy of the region prediction network. Consequently, we isolate this component from the rest of the SLAM framework as much as possible to mitigate any errors or inaccuracies introduced by visual odometry, LCD, or map optimization, which could otherwise affect the results. During incremental training, the robot retraces the same path multiple times (training sequences), creating a map graph that is clustered into regions. During evaluation, the robot retraces the path again (test sequences), and we compare the predicted node region with the ground truth region. The experimental setup requires addressing two main issues:

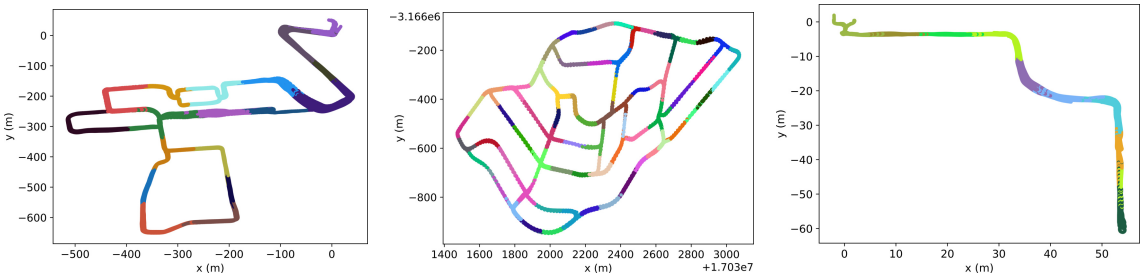
- *Map clustering (without LCD or hybrid)*: as detailed in 2.6.2, an extra arc is added to the graph when the robot located in v_i observes (and recognizes) an already existent node v_q, thereby closing a loop. However this capability depends on the specific loop closure detector used and is not error-free, so in E1 experiments we replaced it with a more precise approach. If the ground truth of node positions is available (this is the case for St. Lucia and OpenLoris) loop-closure arcs are added with all the nodes v_q which are closer to v_i than a given threshold. If the ground truth of node positions is not available (USyd) we use and hybrid approach where: (i) intra-sequence loop-closure arcs are created by RTAB-Map LCD; (ii) inter-sequence loop-closure arcs in successive training sequences are created with nodes closest than a threshold after ICP-based global alignment of the sequences with the map created so far. This preprocessing step reduced the number of USyd input frames from ∼45K to ∼8K per single sequence, totaling ∼153K input images.

  Fig. 4.4 illustrates the clustering results for the datasets used in E1. The number of regions created is 18 for USyd, 68 for St. Lucia, and 10 for OpenLoris corridor. It is worth noting that the regions created have regular shapes and nodes of different sequences are clustered together (thanks to the reassignment mechanism).

- *Computing ground truth regions for nodes in the test sequences.* We use the ICP algorithm to globally align each test sequence with the graph obtained from the clustering of training sequences, and each node in the test sequences is labeled as the closest node in the graph.

To evaluate the accuracy of a CL algorithm, two different metrics are commonly used in the literature: *full test set* and *seen so far* (referred to as fixed and growing test set in [16]). Both track the classification accuracy after each experience, but the former always uses a fixed complete test set (including also classes not yet encountered) while the latter includes in the test set only the classes seen so far. *Full test set* provides insight into the learning trend of an algorithm that, experience after experience, should ideally approach the accuracy of joint training. In *seen so far* the complexity of classification grows with the experiences (since the number of classes increases) and often a (misleading) decreasing accuracy is plotted; however, *seen so far* provides a clear indication of the actual system accuracy at each stage.

Algorithm 3 outlines the entire E1 workflow with *full test set* metric. Since training is incremental, the temporal order of the sequences is relevant (see Table A.1 of Appendix A for details). As the train sequences are visited (lines 7-17), their nodes are dynamically clustered and the node data x together with the corresponding region label y (returned by clustering) are added to fixed-size (n_{exp}) experiences which will be used for successive incremental training (lines 20-24). Before testing, the nodes in the test sequences are aligned and labeled (lines 18-19) as discussed before. The evaluation workflow can be easily modified to cope with the *seen so far* evaluation: in this case, (i) the test set is aligned and labeled at each experience using the graph of the explored portion of the map; (ii) nodes in the test set corresponding to unexplored nodes in the training sequences are excluded from the accuracy evaluation.



(a) USyd (b) St. Lucia (c) OpenLoris - Corridor

Figure 4.4: Map clustering results from the training sequences in E1 experiments. Each color denotes a different region.

4.4.2 Results

Fig. 4.5 presents the top-1 classification accuracy achieved by the proposed CL approach in E1 experiments (with *full test set* evaluation). Across all datasets,

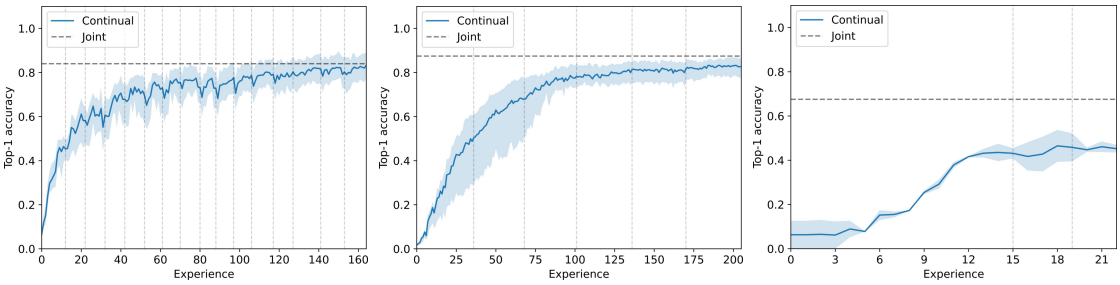
Algorithm 3 E1 workflow - full test set

```
1: procedure E1 - FULL TEST SET
2:     S_train ← train sequences
3:     S_test ← test sequences
4:     n_exp ← experience size
5:     E_c ← ∅ // current experience
6:     E ← ∅ // set of experiences
7:     // Dynamic Clustering (without LCD or hybrid)
8:     for each {V_i, A_i} in S_train do
9:         for j = 0; j < size(V_i); j++ do
10:            x, y ← dynamic clustering(v_j)
11:            E_c ← E_c ∪ {x, y}
12:            if size(E_c) == n_exp or j == size(V_i)−1 then
13:                E ← E ∪ {E_c}
14:                E_c ← ∅
15:            end if
16:        end for
17:    end for
18:    // Align and label S_test
19:    S_test ← align and label(S_test, S_train)
20:    // Incremental training
21:    for each E_i in E do
22:        continual learning(E_i) // Alg. 2
23:        evaluate on full test set(S_test)
24:    end for
25: end procedure
```

the accuracy shows a steady increase during exploration, and for the USyd and St. Lucia datasets, it approaches the accuracy achieved by joint training. This result indicates minimal forgetting of previous knowledge, which is a notable achievement for a CL approach. The improvement in accuracy demonstrates the benefit of revisiting previously explored locations, as this allows for the addition of new poses and the acquisition of more environmental variations. This advantage underscores the superiority of continual SLAM over static SLAM. On the OpenLoris corridor dataset, the top-1 accuracy is lower due to the increased difficulty of the dataset, which includes fewer sequences and sequences captured in the dark (no artificial lighting inside the corridor).

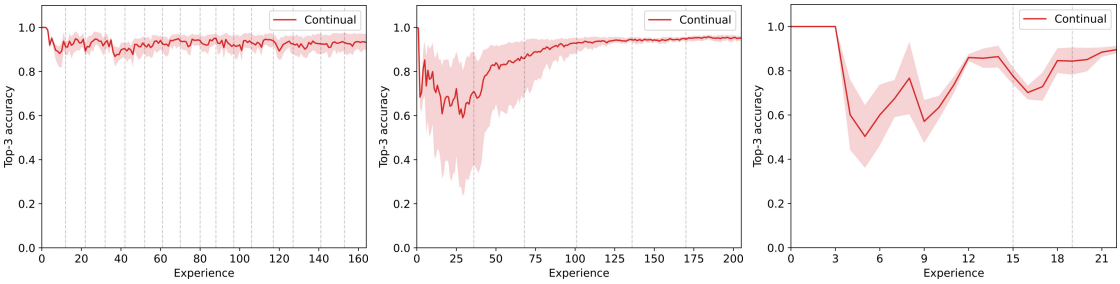
Complementary results appear in Fig. 4.6 where we examine the classes *seen so far* and consider the classification correct if the ground truth region is among the top-3 regions predicted by the model (in line with the number top-k of regions moved

to WM in our RTAB-map integration). This provides a better idea of the practical effectiveness of the proposed region selection mechanism. Observations include: (i) the accuracy is very high at the beginning of the first sequence because the number of classes is relatively small; (ii) in the second part of the first sequence, the number of classes steadily increases (up to the total number of classes), but only a limited amount of place variation can be learned, leading to a slight decrease of accuracy; (iii) during successive training sequences the top-3 accuracy grows and stabilizes to very high values by the end of exploration.



(a) USyd (b) St. Lucia (c) OpenLoris corridor

Figure 4.5: Top-1 region classification accuracy in E1 experiments (full test set evaluation). The dashed vertical lines identify the boundaries of the training sequences. The faded color highlights the gap between the minimum and the maximum accuracy on the test sequences.



(a) USyd (b) St. Lucia (c) OpenLoris corridor

Figure 4.6: Top-3 region classification accuracy in E1 experiments (seen so far evaluation). The dashed vertical lines identify the boundaries of the training sequences. The faded color highlights the gap between the minimum and the maximum accuracy on the test sequences.

4.5 E2 experiments

4.5.1 Results on RTAB-Map

We first present E2 results on USyd, as its extended exploration period provides valuable insights. We compare our modified RTAB-Map with the baseline version in two configurations: (i) unbounded (U), where the WM can grow indefinitely, and (ii) time-limited (L), where nodes are transferred to LTM when the node processing time exceeds 1 second. We evaluate these versions across 12 USyd sequences, simulating extensive navigation in both time (266 minutes) and space, and accumulating a substantial set of nodes in memory.

The final map graphs produced by all three methods are visually similar (see Fig. 4.7), but the exploration statistics (Table 4.2) reveal notable differences. Although the absence of ground truth for node positions prevents precise computation of LCD metrics, we observe that our approach detects approximately twice as many loop closures and relocalizations compared to the baseline versions. We attribute this performance to two factors:

- Since the network predictions are accurate, the target nodes for LCD have high chance to be in WM (high recall).
- Since the node processing time is significantly lower in our approach (about 300ms vs more than 700ms) more than twice as many nodes are created in the map and the increased granularity allows a finer-grained node matching.

While we cannot guarantee that all detected loop closures are true positives (i.e., precision = 100%), we argue that RTAB-Map's stringent acceptance criteria for loop closure hypotheses likely result in a negligible number of false positives.

To better understand the role of WM, in Fig. 4.8 we plot its size throughout the exploration. RTAB-Map (U) shows a logarithmic growth in WM size, which becomes less than linear due to increasing node processing times (see Fig. 4.9a), leading to fewer nodes being created as the map expands. In RTAB-Map (L) WM grows until reaching a certain point, after which it stabilizes. In contrast, our approach maintains a consistently small WM size (less than 1/10 on average, according to Table 4.2) because only the top-3 regions and some neighbors of the highest loop closure hypothesis are retained. Results in Table 4.2 reflect an online execution of RTAB-Map. Under this setup, node creation follows a best-effort strategy and depends on the computational cost of each iteration. For this reason, there is a different number of generated nodes.

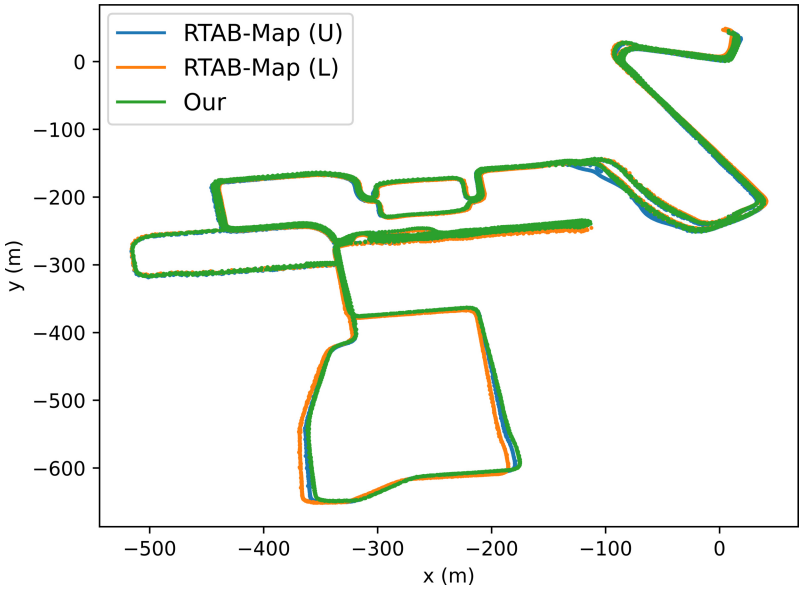


Figure 4.7: The map graphs created by the three versions of RTAB-Map at the end of the 12 USyd sequences.

	RTAB-Map		**Our**
	(U)	**(L)**	
Nodes created	19290	18800	45519
Average time (ms)	759	777	301
Max WM size	14370	7905	1520
Average WM size	6866	4813	426
Loop closures and relocs detected	851	862	1590

Table 4.2: Statistics of E2 on the 12 USyd sequences. RTAB-Map is compared in the three versions: RTAB-Map (U = Unbounded), RTAB-Map (L = Time-limited) and our.

Fig. 4.9 illustrates the processing time of each node during exploration. RTAB-Map (U) exhibits an increasing average processing time, requiring more than 6 seconds by the end of the exploration, which is very far from the real-time constraint (assumed to be 1 second per node). On the contrary, our approach keeps the average node processing time constantly below 1 second. Individual time measures reveal occasional peaks, higher than the average, during phases of intensive computation such as massive node transfer-retrieval or incremental model training. These peaks could be reduced with a more optimized implementation utilizing asynchronous parallel threads. Fig. 4.10 provides a further breakdown of the node processing time, distinguishing between LCD and map optimization time. It is evident that,

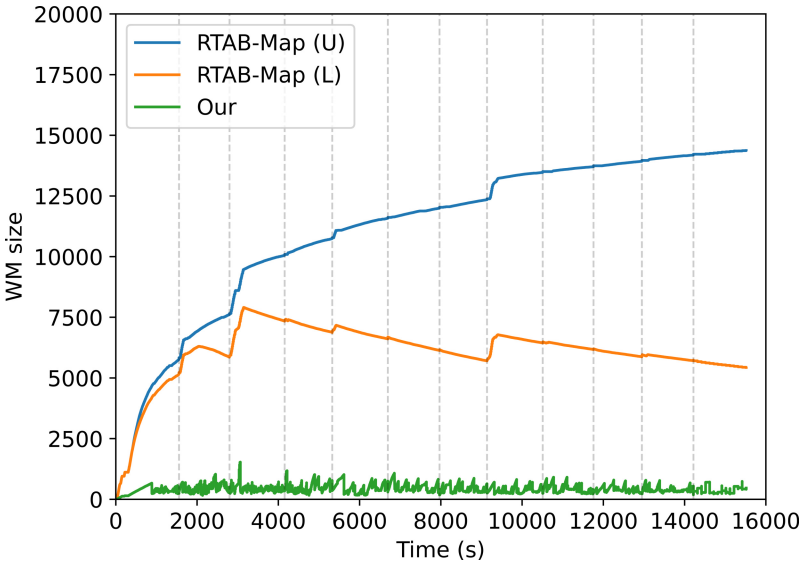
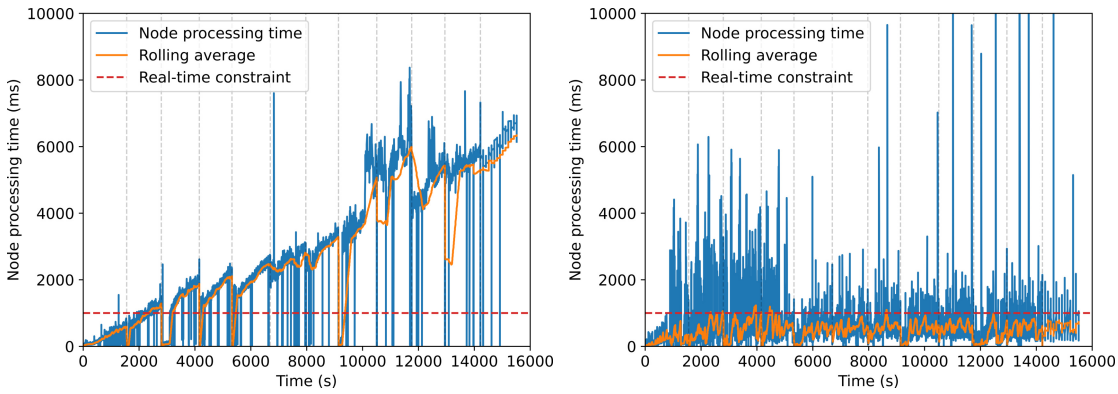


Figure 4.8: WM size of RTAB-Map during exploration of 12 USyd sequences.

in this experiment, map optimization time dominates by a large margin LCD time. This point will be further elaborated in the next subsection.



(a) RTAB-Map (U)

(b) RTAB-Map our

Figure 4.9: Comparison between the node processing time of RTAB-Map (U) and our version. The vertical dashed lines identify the sequence boundaries, while the red horizontal dashed line denotes a real-time constraint, here set to 1 second. The orange line is a rolling average of the node processing time, computed over a window of 100 data points.

Hereafter we present E2 results on KITTI and OpenLoris datasets. Despite their smaller size, lack of real path retracing, and limited loop closures and relocalizations, the availability of ground truth node positions allows for additional insights. Tables 4.3 and 4.4 report the number of loop closures and relocalizations detected by our RTAB-Map version. Both tables indicate very high precision-recall metrics, with nearly all loop closures and relocalizations detected at precision 1, since no

Sequence	Real LC	Detected LC	
		Incremental (Our)	Offline [3]
KITTI/00	4	4	4
KITTI/02	2	2	2
KITTI/05	3	3	3
KITTI/06	1	1	1
KITTI/07	1	1	1
KITTI/09	1	1	1
OpenLoris/Market3	1	1	1
Total	13	13 (100%)	13 (100%)

Table 4.3: Loop Closure (LC) detection results.

Sequence	Real Relocs	Detected Relocs	
		Incremental (Our)	Offline [3]
OpenLoris/Market2-(1)	3	3	3
OpenLoris/Market3-(1,2)	3	2	3
Total	6	5 (83%)	6 (100%)

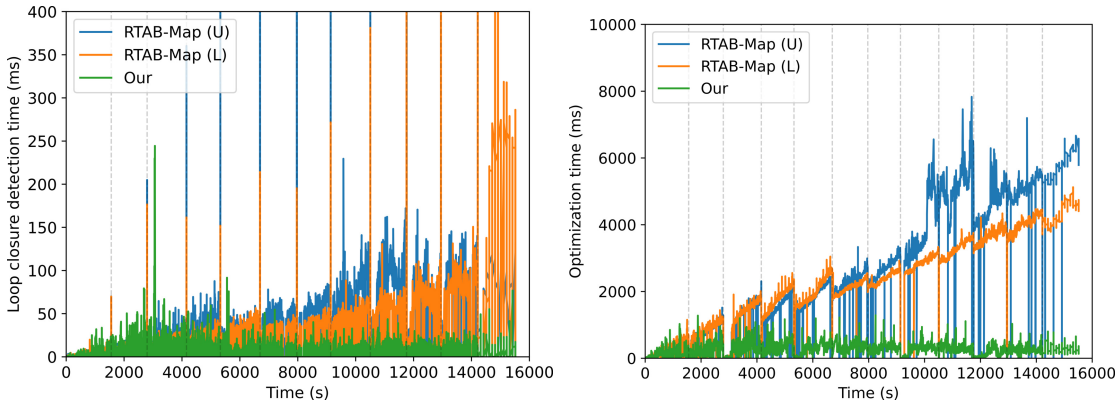
Table 4.4: Relocalization results. The numbering n-(x,y,...) at the end of each sequence name means the relocalizations in the sequence n are computed after the exploration of the sequences (x,y,...).

false positives have been produced. For comparison, the tables also indicate the results obtained from the offline training done in [3], highlighting that there are no substantial differences between offline and incremental training, demonstrating the effectiveness of the proposed continual learning approach.

Since long-term localization is fundamental for map optimization (and drift correction), we computed the absolute trajectory error (ATE) for KITTI and OpenLoris sequences, reported in Table 4.5. Here we compare our approach with RTAB-Map (U) and RTAB-Map (L). For RTAB-Map (L) we limit the max WM size to the maximum size used by our approach. The results show that our approach is always better than RTAB-Map (L), except for KITTI/06, which is a very short sequence, suggesting that our submap selection for optimization is more effective. RTAB-Map (U) can optimize the full map after each loop closure thus achieving a slightly lower ATE, but this strategy is too computationally demanding for real-time exploration of large maps, as shown in Fig. 4.10b.

Sequence	ATE (m)		
	RTAB-Map		Our
	(U)	(L)	
KITTI/00	1.31	1.83	1.78
KITTI/02	3.79	4.46	4.23
KITTI/05	1.37	1.65	1.5
KITTI/06	1.03	1.11	1.15
KITTI/07	1	1.13	1.08
KITTI/09	3.15	3.39	3.33
OpenLoris/Market	1.53	1.7	1.68
Average	1.88	2.18	2.09

Table 4.5: Absolute trajectory errors of RTAB-Map (U = Unbounded), RTAB-Map (L = Time-Limited) and our. For the OpenLoris sequence, the ATE is computed after the exploration of the sequences Market1, Market2, and Market3. The ATE values obtained by RTAB-Map (U) on some of the KITTI sequences are different from those reported in the original paper [5] because we used the default values for the RTAB-Map parameters.



(a) Loop closure detection time

(b) Optimization time

Figure 4.10: Split-down of the node processing time in the two most computing-intensive phases: LCD and Map Optimization.

4.5.2 Results on ORB-SLAM3

ORB-SLAM3 [77] is a popular SLAM solution based on BoWs with a fixed vocabulary of ORB descriptors [64]. When comparing it with our approach we must keep into account that the SLAM pipeline is very different with respect to RTAB-Map and we cannot isolate a single processing step (e.g., LCD) for one-to-one comparison. So we focused on two global aspects: (i) the quality of the reconstructed maps at the end of the exploration; (ii) the processing time of some salient steps.

We run ORB-SLAM3 on the first 8 sequences of the USyd dataset. Our attempts to extend this evaluation to the set of 12 sequences, as done with RTAB-Map, were unsuccessful due to out-of-memory errors on a workstation with 64GB of RAM. This issue arises because ORB-SLAM3 lacks advanced memory management and keeps all the map data in the main memory. We also identified (and corrected) a critical bug in the code due to an integer overflow caused by a larger-than-expected number of keyframes and map points. This clearly means that ORB-SLAM3 was not previously tested on maps of this size (with ∼152K frames). The first attempts were done with RGB images, but the visual tracking was unsatisfactory, so we switched to RGB-D images, that were offline computed from laser scan using RTAB-Map. To further improve the tracking we carefully adjusted some parameters controlling the amount of keypoints per frame and ORB-specific thresholds (see Settings in Table 4.6). These changes led to the extraction of a larger number of keypoints, including keypoints with low saliency; however, this configuration demonstrated superior performance in our experiments. Specifically, we observed that, in certain scenarios, the standard parameterization yielded an insufficient number of keypoints, which proved inadequate for reliable visual odometry under challenging conditions such as unstructured environments or rapid turns with limited image overlap. Performance metrics show an efficient place recognition (∼60 ms) due to the use of inverted-index-based candidate retrieval made possible by the fixed vocabulary. To allow full processing of all frames and maximize tracking accuracy we executed ORB-SLAM3 in offline mode.

Despite all the above optimizations, ORB-SLAM3 demonstrated lower robustness compared to RTAB-Map, likely due to the use of a fixed dictionary of less accurate visual features [128] (refer to the discussion in Section 2.3.3). In particular, during navigation the tracking has sometimes been lost, leading to repeated relocalization and redundant map segments. ORB-SLAM3 was often unable to detect loops when the same path was retraced in the opposite direction, resulting in the presence of false negatives (see Fig. 4.11a). Some false-positive loop closures were also detected, degrading overall map accuracy. For the above reasons, the map generated by ORB-SLAM3 after the 8 sequences (Fig. 4.11b) is globally inconsistent and qualitatively inferior with respect to RTAB-Map (Fig. 4.7).

On the efficiency side, the performance metrics made available by the internal logging mechanism (see Table 4.6). On the other hand, post-relocalization map merge takes ∼1194 ms and post LCD map optimization requires ∼19633 ms. Therefore, even if map optimization is performed asynchronously, it is evident that ORB-SLAM3 has difficulty operating in real-time on large maps.

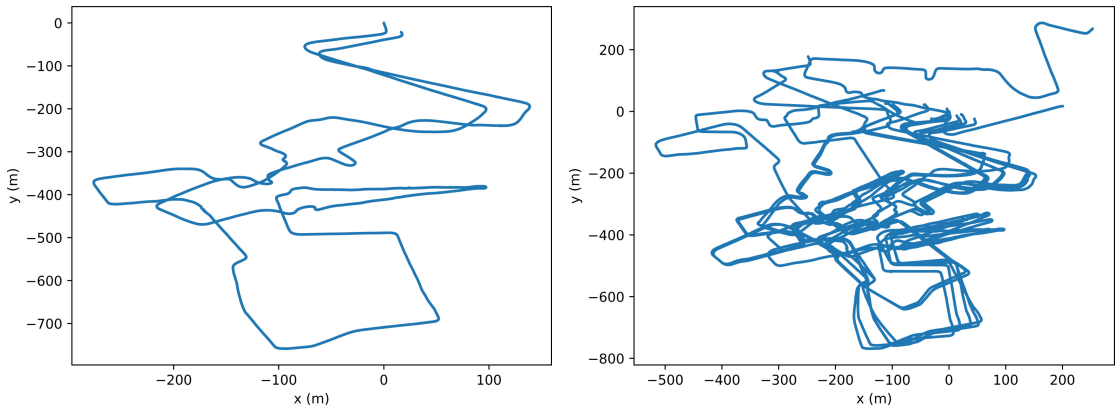
Settings	Sensor	RGB-D
	Resolution	960x604
	# ORB features	2000
	# ORB levels	12
	iniThFAST	4
	minThFAST	2
Place Recognition	Database query	43.55 ms
	Compute Sim3/SE3	10.01 ms
	Total	60.04 ms
Map Merging	# Detected merges	7.2
	Merge Maps	129.881 ms
	Welding BA	259.64 ms
	Opt. Essential Graph	773.0041 ms
	Total	1193.94 ms
Loop	# Detected loops	13.4
	Loop Fusion	66.82 ms
	Opt. Essential Graph	19532.13 ms
	Total	19633.41 ms
Map size	# Keyframes	19583
	# Map points	1241328.8

Table 4.6: ORB-SLAM3 settings and results, averaged over the five runs. For details on single metrics refer to ORB-SLAM3 paper [77].

4.5.3 Results on DROID-SLAM and DPVO

DROID-SLAM is a modern SLAM solution completely based on deep neural networks. However, the current implementation is very resource demanding, because it stores the whole dataset in the GPU memory. As reported in [54], running DROID-SLAM on ETH3D [129], consisting of about 5K frames, requires a GPU with at least 24GB of memory. Considering that in USyd a single sequence is composed of about 45K frames, even if the input images are reduced to 320×240 and the keyframe subsampled, we cannot run it on the hardware at our disposal.

Deep Patch Visual Odometry (DPVO) is an evolution of DROID-SLAM. In our experiments with DPVO on the USyd dataset we noted substantial improvements with respect to to DROID-SLAM, but the system is still not sufficiently scalable to work with very long sequences. Using the DPVO's defaults parameters, with a stride of 5, the default backend for LCD and a maximum buffer size of 1,000,000



(a) (b)

Figure 4.11: (a) The map created by ORB-SLAM3 on a single USyd sequence, with no loop closures detected when the robot navigates the same places in the opposite direction. (b) The map path created by ORB-SLAM3 on the 8 sequences used in the experiment. Because of false closures, loss tracking, and inaccurate map merge, the overall result is significantly worse than that obtained with RTAB-Map.

(to allow inclusion of all USyd frames) we noted that after about 50K frames the system performance start degrading, and at about 70K frames became unresponsive on the workstation used. This is not surprising because all experiments included in the DPVO paper [55], and the associated repository were performed on shorter sequences.

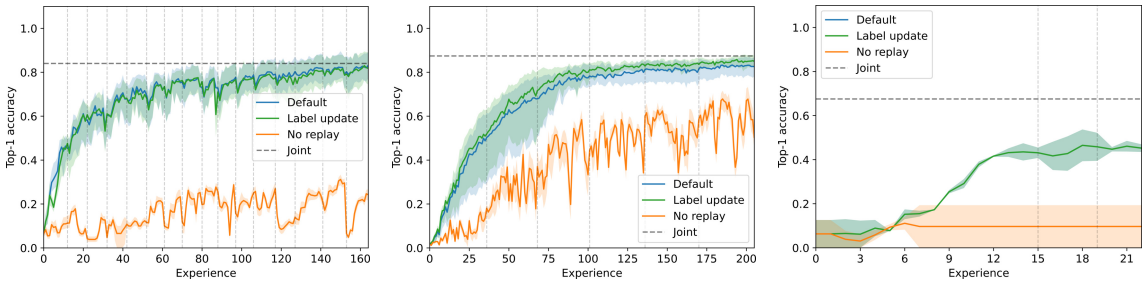
4.6 Additional experiments

Hereafter we report further experiments on two key components of our method, the replay memory and the dynamic clustering algorithm.

4.6.1 Replay Memory

In the default implementation of the proposed approach (used in all the experiments so far), when a node belonging to the current experience is reassigned to another region, we update the corresponding label (see step 5 in Fig. 4.1); however, for simplicity, we do not perform any label update in the replay memory where other reassigned nodes might be present. Hereafter, we reconsider this choice and evaluate the potential benefits of extending label updates to the replay memory. For completeness, we also examine the scenario in which replay memory is not utilized at all. Fig. 4.12 presents the results in the same E1 setting of Fig. 4.5. We immediately note how crucial is the use of the replay memory to control forgetting during the

continual learning process. On the other hand, label updates offer no advantages for the USyd and OpenLoris corridor datasets and only provide a marginal benefit for St. Lucia. We think this is due to the continuous refresh of the replay memory where old patterns are naturally replaced by newer ones as the exploration proceeds.



(a) USyd (b) St. Lucia (c) OpenLoris corridor

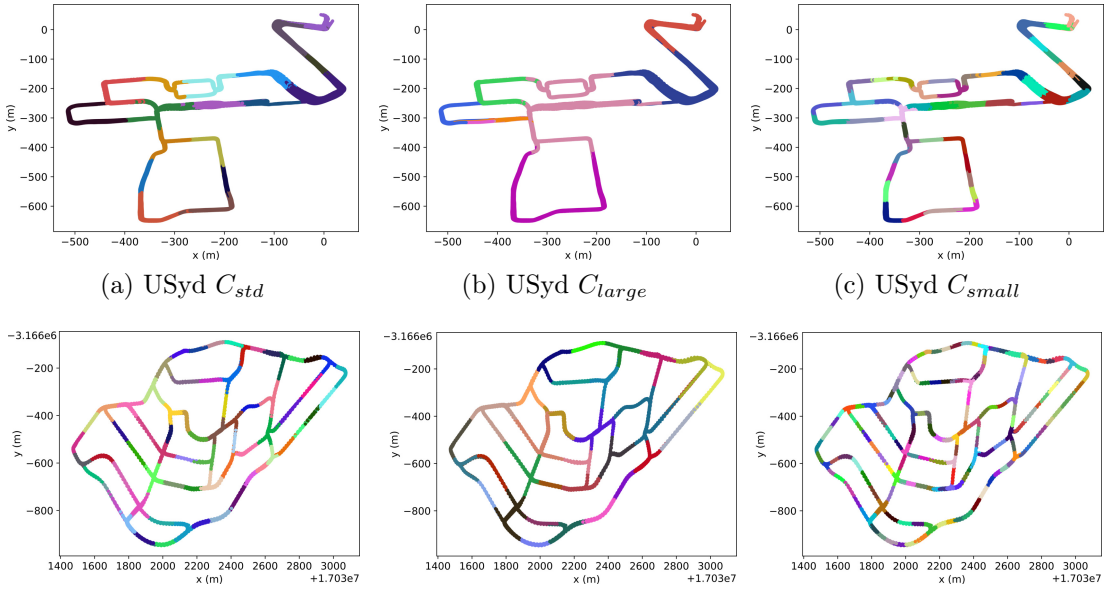
Figure 4.12: The effect of label updates of reassigned nodes in the replay memory. No replay denotes the case where replay memory is not used.

4.6.2 Clustering Granularity

To study the robustness of the proposed approach with respect to the formation of the regions, hereafter we report additional experiments with different cluster granularities. We consider three settings: C_{std} is the standard parametrization used so far ($\eta = 100$ and $\rho = 1000$); C_{small} ($\eta = 50$ and $\rho = 500$) and C_{large} ($\eta = 200$ and $\rho = 2000$) lead to smaller and larger clusters, respectively. Fig. 4.13 shows the clustering results obtained on the USyd and St. Lucia datasets: we observe that the clustering approach produces quite homogeneous and regular clusters for the different granularities. We then evaluated the region prediction CL in the E1 scenario, obtaining the top-3 accuracy reported in Fig. 4.14. Even if increasing the number of regions (i.e. classes) makes the initial prediction slightly more difficult, the accuracy trend is quite similar for the three settings denoting good robustness.

4.7 Conclusions and Future Work

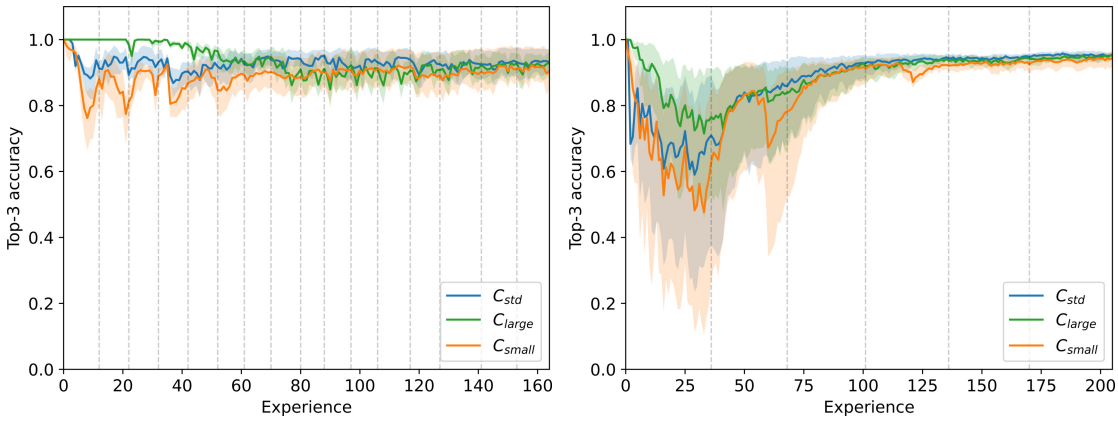
In this work, we introduced a novel approach to improve the long-term localization of a robot on large maps, using a deep neural network trained online and incrementally to pre-filter candidate nodes for LCD and map optimization. The dynamic clustering and the region prediction mechanism proposed are rather simple but very effective, as our experimental results demonstrate: almost all loop closures and relocalizations were detected despite a consistent reduction in the number of



(a) USyd C_{std} (b) USyd C_{large} (c) USyd C_{small}

(d) St.Lucia C_{std} (e) St.Lucia C_{large} (f) St.Lucia C_{small}

Figure 4.13: USyd and St. Lucia maps clustered using the three clustering settings C_{std}, C_{large}, and C_{small}.



(a) USyd (b) St. Lucia

Figure 4.14: Top-3 accuracy on USyd and St. Lucia in the seen so far evaluation under the three clustering settings C_{std}, C_{large}, and C_{small}.

selected nodes and processing time. We have shown that continual learning is very well tolerated by the prediction network (with respect to an offline strategy where the network is pre-trained once and then frozen) and considering the top-3 most probable regions allows to select the map portion containing the target nodes for LCD and relocalization. The careful optimization and complete integration into RTAB-Map facilitate the practical applicability of our approach to continual SLAM on large maps. Furthermore, since the proposed region prediction is independent of the specific SLAM pipeline it can be integrated into other SLAM frameworks

(including monocular ones, since neither RGB-D nor stereo data are mandatory). However, to fully leverage region prediction, the target SLAM should integrate a memory management subsystem. A drawback of the current approach is that map optimization is performed on subsets of map nodes, potentially leading to accuracy drops with respect to global optimization. Although in our experiments this was not critical, further validations could be useful to point out potential weaknesses and, if necessary, adapt the map optimization policy (e.g., interleave local and global optimizations).

Future work will also involve replacing the current network architecture (ResNet18) with a more advanced ad hoc place recognition model such as NetVLAD [66] or RegionVLAD [84] and evaluating the feasibility of continual training with these models, to address robustness problems that can arise when conditions are very hard, for example when there is a great variation in lighting and/or viewpoint or when two locations are identical. On the scalability side, the main limitation of the current implementation is given by the fact we do not impose a fixed upper bound on the number of regions, which may lead to increased training time. To keep the latent replay strategy effective, a minimum number of samples per region must always be maintained; therefore, if the number of regions grows unbounded, the total number of samples used during training also increases accordingly, potentially causing a significant rise in training time if not properly constrained. Future developments will include the possibility of merging multiple regions into a single one. Finally, we plan to refine the clustering approach by exploiting additional features beyond the sole spatial location.

Chapter 5

A Synthetic Dataset for Robotics and Navigation Tasks

"Without data, you're just another person with an opinion."

– W. Edwards Deming

In the context of computer vision, numerous tasks require extensive datasets to train robust algorithms capable of generalizing effectively in real-world scenarios. Techniques based on deep learning necessitate substantial data volumes, often acquired under diverse environmental conditions (especially in outdoor scenarios) to ensure adaptability to varying contexts, including lighting conditions, weather variations, seasonal changes, and human interactions. The acquisition of such datasets poses hard challenges: it is a laborious task, time- and energy-consuming, often also costly, and not always feasible.

To address these challenges, the generation of synthetic data has emerged as a promising solution. By creating data synthetically, researchers can simulate diverse environmental conditions in a controlled manner and generate vast amounts of data, thus substituting or integrating real-world datasets for algorithm training and improvement. For this purpose, synthetic data must be qualitatively close to real data, and this is not trivial, since real-world data can be difficult to replicate in a simulation, being subject to complex physical rules.

Our aim is to demonstrate that synthetic data obtained from the famous video game Grand Theft Auto V (GTA V) [130] can be used in conjunction with or even in place of real datasets in robotics and navigation tasks. For this purpose, data were acquired from GTA V for Simultaneous Localization and Mapping (SLAM) and Visual Place Recognition (VPR). Several experiments were conducted to demonstrate

the usability of these synthetic data in both tasks. The advantages of these acquisition methods are remarkable. For instance, a notable challenge in robotics is the long-term or lifelong SLAM [112]. This refers to a robot's ability to deal with very large and long-term explorations, where it can repeatedly traverse the same path under varying conditions, with the environment evolving over time. To achieve accurate pose estimation, the robot must relocalize itself when revisiting the same locations. This problem is known as Loop Closure Detection (LCD) or relocalization [38]. To the best of our knowledge, there are very few existing SLAM datasets in which a robot retraces the same paths multiple times over many hours, under different lighting and weather conditions, within a single acquisition. This would require extensive effort in the real world, but it can be easily achieved by leveraging GTA V. Furthermore, in a simulated environment, when a path is retraced, we can introduce local pose perturbations to add variability in the camera view and make LCD more challenging.

Even if the idea of acquiring datasets from simulators and games like GTA V is not new, in the work presented in this chapter, we specifically focus on synthetic data generation for robotic tasks, where real data are still predominantly used, particularly for VPR. Furthermore, we developed a simple but effective automatic algorithm for generating a VPR dataset from GTA V acquisitions, eliminating the need for human supervision.

We believe our work paves the way for the creation of unprecedentedly large datasets for all the aforementioned robotic vision tasks, composed solely of synthetic data from GTA V, which can be easily and rapidly enriched at no cost. The main novelties and contributions of the work presented in this chapter are:

- The creation of a novel synthetic dataset acquired from GTA V, specifically designed for use in robotics and navigation tasks.
- The development of an autonomous frame selection algorithm to generate a VPR dataset from synthetic data acquired with GTA V, without human supervision.
- The description of the pipeline and tools used to create the dataset to allow other researchers to generate further datasets tuned for their needs easily.
- A series of experiments to support and validate the use of synthetic data acquired from GTA V for SLAM and VPR applications.

The rest of the chapter is organized as follows: Section 5.1 introduces the data that can be extracted from GTA V and the computer vision tasks that can be ad-

dressed utilizing such data. Section 5.2 discusses existing SLAM and VPR datasets and presents other datasets generated using GTA V. Section 5.3 details the generation pipeline, while Section 5.4 and 5.5 present the dataset and the automatic frame selection algorithm for VPR. Section 5.6 reports the experiments conducted for VPR and SLAM, along with key insights. Finally, Section 5.7 provides conclusions and outlines future work.

5.1 Background

With the rapid growth of deep learning, artificial neural networks have begun to be used to solve a wide variety of computer vision tasks, with outstanding performance. To train deep neural networks, researchers started to acquire large datasets, usually task-specific.

5.1.1 Type of data

While some data used in computer vision are easy and cheap to acquire with real-world sensors (e.g., RGB images), others require costly devices or time-consuming manual labeling (e.g., 3D point clouds, segmented images, or bounding boxes). Using GTA V enables the generation of highly accurate and realistic data at no cost.

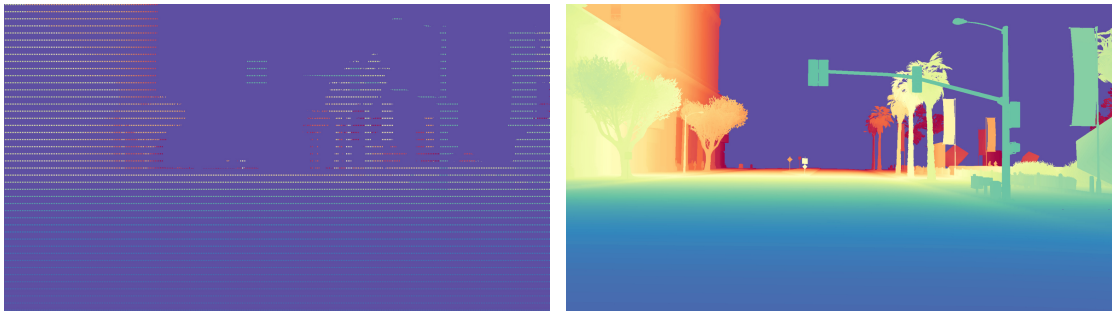
RGB image: uses the RGB color model to represent each pixel as a triplet (R, G, B) with intensity values in [0, 255]. In GTA V, these are screen captures during gameplay.

Depth image: also known as depth map, it encodes the distance from the sensor to scene objects, capturing 3D spatial information. It can be dense (e.g., from RGB-D cameras, see Fig. 5.1b) or sparse (e.g., from laser scans, see Fig. 5.1a), depending on how it is acquired. Unlike sensor-based maps that may have holes, GTA V provides dense, complete depth maps, generated during rendering without noise. GTA V uses an inverse logarithmic encoding (smaller values = greater distances), normalized between 0–1, with a maximum range of 960m.

Point cloud: it is a 3D representation where each pixel becomes a 3D point, optionally with color (Fig. 5.2). Typically acquired via lasers, point clouds can be converted to a depth image and vice versa.

Camera pose: it is a vector $[x, y, z, \phi_x, \phi_y, \phi_z]$ representing the position and the orientation of the camera. Crucial for tasks like SLAM, accurate pose estimation is expensive in the real world, but freely available and precise in GTA V.

Bounding boxes: they are rectangular boxes that localize objects in an image, commonly used in object detection and autonomous driving. They can be defined



(a) (b)

Figure 5.1: On the left, (a) shows a sparse depth map, while on the right, (b) is the corresponding dense depth.

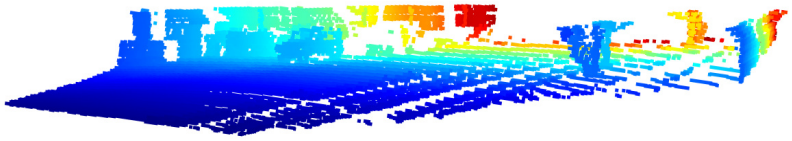


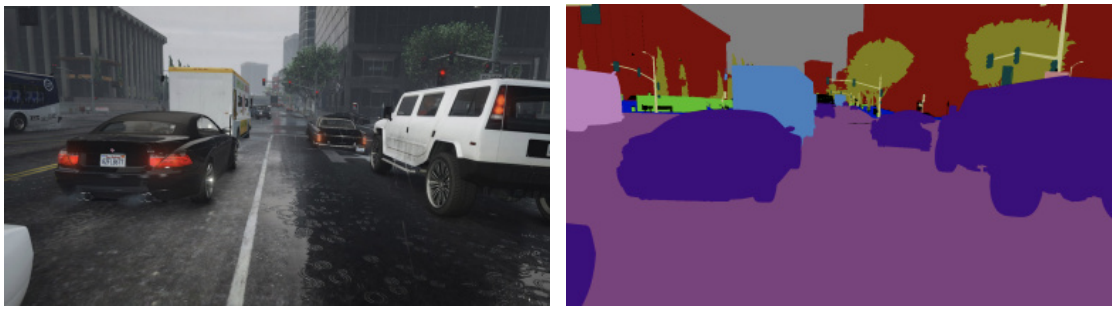
Figure 5.2: Three-quarter view of a point cloud acquired through a 3D laser from the KITTI dataset [119]. Color fades from blue to red, indicating greater proximity to the sensor at blue and greater distance at red.

by (x_1, y_1) and (x_2, y_2) corners or (c_x, c_y, w, h), by using width and height. Fig. 5.3 shows an example.

Figure 5.3: Image with bounding boxes from the PreSIL dataset [131], acquired from GTA V. The blue rectangle are the bounding boxes for the vehicles, while the red ones are for the pedestrians.

Segmented images: divide the image into regions representing objects or semantic classes. They are widely used in tasks such as autonomous driving, robotics, and medical imaging. Fig. 5.4 shows an example from GTA5 [132].

Additional metadata, like motion speed, time of day, or stencil images, can also



(a) (b)

Figure 5.4: On the left, (a) shows an RGB image, while on the right (b) shows the corresponding segmented image. The images are from the GTA5 dataset [132], acquired from GTA V. Each pixel is colored according to the category of the object to which it belongs.

be extracted from GTA V but are beyond the scope of this work.

5.1.2 Tasks

Here we recall from Chapter 2 the main tasks of the computer vision community that can be addressed using data acquired from GTA V.

Place recognition: it is the task of identifying previously visited locations by comparing current sensor data to a database. When using images, it becomes Visual Place Recognition (VPR), framed as an image retrieval problem. It is challenging due to appearance changes (lighting, occlusions, viewpoint, etc.). The process involves extracting feature vectors from images [133] and comparing them using similarity metrics like cosine similarity.

SLAM: Simultaneous Localization and Mapping (SLAM) is the problem of enabling a robot to build a map of an unknown environment while localizing itself within it [1]. It is essential for autonomous navigation, especially where GPS is unreliable (e.g., indoors). There are several approaches in the literature, the current state-of-the-art solutions are often graph-based, representing robot poses as nodes and sensor-derived constraints as edges [35]. When the robot revisits known locations, Loop Closure Detection (LCD) or Relocalization helps refine the map [38]. SLAM commonly uses lasers, cameras, and inertial sensors. In this work, a visual SLAM approach is adopted, relying on images to estimate motion through visual odometry [46].

Other computer vision tasks can be addressed using data from GTA V:

- **Depth estimation** involves predicting depth information from RGB images to mitigate the high cost of depth sensors. However, challenges such as accu-

rate scene structure representation and external factors like lighting persist. Supervised generative AI methods often perform better but require expensive, high-quality datasets.

- **Depth completion** focuses on transforming sparse or incomplete depth maps into dense ones, serving as a pre-processing step for tasks like SLAM. However, highly sparse data remains problematic.
- **Object detection** identifies and localizes objects in images or videos by combining bounding box prediction (localization) with class assignment (classification).
- **Semantic segmentation** goes further by assigning a class label to every pixel, providing a detailed, pixel-level understanding of an image compared to the bounding box approach of object detection.

5.2 Related Works

This section presents an overview of the most common publicly available datasets for SLAM and VPR tasks.

5.2.1 Place Recognition

Table 5.1 reports well-known place recognition datasets. These datasets exhibit multiple illuminations and/or seasonal conditions. Note that none of them are synthetic.

5.2.2 SLAM

Table 5.2 lists well-known and widely used SLAM datasets. All the datasets reported are provided with ground truth. In recent years, several synthetic datasets have been proposed for SLAM, in particular, GTASynth [138] is another dataset extracted from GTA V. In Table 5.2, IMU stands for Inertial Measurement Unit and refers to inertial sensors such as accelerometers and gyroscopes.

5.2.3 Other GTA V datasets

GTA V has already been used to extract datasets for different computer vision tasks. GTASynth [138] provides point clouds with ground truth poses for SLAM, HRSD [146] is aimed at monocular depth estimation, PreSIL [131] targets object detection,

Dataset	# images	Data type	Multiple conditions	Synthetic
Tokyo 24/7 [134]	76000	Mono RGB images	✓	✗
Alderley [135]	29214	Mono RGB images	✓	✗
Nordland Dataset [122]	28865	Mono RGB images	✓	✗
Pittsburgh 250k [136]	∼256000	Mono RGB images	✓	✗
Mapillary [137]	∼1.6 million	Mono RGB images	✓	✗

Table 5.1: The main publicly available place recognition datasets.

and GTA5 [132] focuses on semantic segmentation. Other datasets also exist [147, 148]. While these have shown that GTA V data can be used with results comparable to the use of real data, most of them are not suitable for VPR and SLAM tasks, which require multiple retracing of the same path. We used GTA V RGB-D data for robotics and navigation tasks, particularly VPR, and have post-processed the data to build a dedicated dataset (see Section 5.5). A possible extension of this work involves acquiring all available GTA V data to construct datasets tailored to various tasks. Unlike PreSIL, which is composed of several different data types, most of the existing datasets are task-specific and limited in scope.

5.3 How to capture data from GTA V

To acquire data from GTA V we used three open-source tools: ScriptHook V [149], G2D [150] and DeepGTAV [151]. ScriptHook V is a popular modding tool used primarily to modify and enhance GTA V by adding new features, custom behaviors, or entirely new content that was not originally included in the game. It allows players to run custom scripts written in the .NET framework and it works as the basis for G2D and DeepGTAV. The idea is to use the playable character of GTA V to drive a car along a predefined path (defined using G2D) while data are recorded at a designated frame rate (using DeepGTAV). The process can be summarized in four steps (as depicted in Fig. 5.5):

Dataset	Data type	Synthetic
KITTI odometry [119]	Stereo RGB images Point clouds	✗
TUM RGB-D [139]	RGB-D images	✗
EuRoC MAV [140]	Stereo RGB images IMU	✗
Newer College Dataset [141]	Stereo RGB images Point clouds IMU	✗
NCLT Dataset [142]	360 RGB images Point clouds IMU Odometry	✗
St. Lucia Dataset [121]	Mono RGB images GPS	✗
USyd Campus Dataset[1][115]	Mono RGB images Point clouds GPS Odometry	✗
Virtual KITTI 2 [143]	RGB-D images	✓
ICL-NUIM [144]	RGB-D images	✓
Redwood SLAM [145]	RGB-D images Point clouds	✓
GTASynth [138]	Point clouds	✓

[1] USyd does not have a ground truth.

Table 5.2: The main publicly available SLAM datasets.

a. *Path selection*: thanks to G2D, it is possible to manually place markers on the GTA V map to outline a sparse trajectory. The order of the markers corresponds to the order in which those locations will be visited (highlighted by the vertices numbering in Fig. 5.5a).

b. *Trajectory refinement*: a G2D script converts the sparse set of trajectory points into a dense set that follows the path of the road (see Fig. 5.5b).

c. *Acquisition*: DeepGTAV runs a client-server architecture, used to export data from GTA V. When the acquisition starts, the game enters into a self-driving mode, and the server (written in C++) interacts with the game rendering pipeline to extract frame data and send them to the client (written in Python) as a JSON file. The client can also send messages to the server, to change the game settings (e.g., the weather, the location to reach, etc.). The client directly saves the JSON files received from the server, because processing the

data is time- and computational-demanding and may prevent the acquisition at high frame rates (see Fig. 5.5c).

d. *Post-processing*: some data (e.g., depth images) require post-processing to be stored using a standardized encoding. For this reason, once the acquisition is complete, the different types of data (i.e., RGB, depth image, point cloud, camera pose, bounding boxes and labels, segmented image) are extracted from the corresponding JSON file and stored separately for each frame (see Fig. 5.5d).

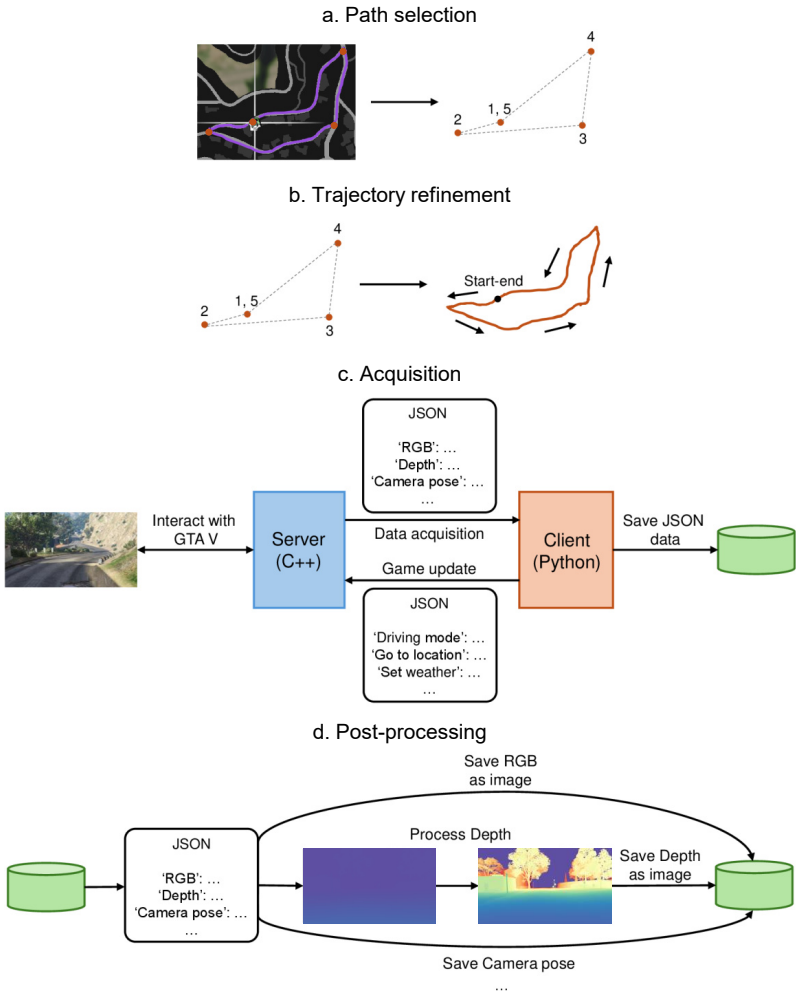


Figure 5.5: Data acquisition process.

The tools used allow the control of various game parameters (driving mode, weather, time of day, car speed), which can be set and modified by the external client during the acquisition process. As a result, it is possible to create a simulated, fully controllable and reproducible environment with which to capture data under different conditions. This is an incredible advantage for real-world open problems,

such as lifelong SLAM [112], for which extensive explorations, even hours or days, under different conditions are required before returning to places already visited, and such datasets do not exist in the community. Furthermore, during trajectory refinement (step b), it is possible to perturb the path and introduce small changes in the poses. This adds variability in the camera view when retracing the same path, which is a desirable condition for SLAM and VPR datasets.

At the end of the acquisition process, captured data can be selected to create various datasets useful for different tasks. For instance:

- RGB, depth images and poses ground truth can be used for monocular depth estimation [152] or SLAM.
- RGB images captured in different weather or day/night conditions can be useful for VPR.
- RGB images, point clouds, and ground truth poses can be used for depth completion or SLAM.
- RGB images with precise object bounding boxes and labels, along with depth images and segmented ground truth images can be useful for object detection and image segmentation, respectively.

The capability of this acquisition approach is remarkable as it allows the collection of a wide variety of data under a large range of controllable conditions for many different tasks very efficiently at almost no cost. Since the game enters into autonomous driving mode during the acquisition process, only outdoor data can be acquired at the moment.

5.4 Acquired dataset

Following the process described in the previous section, a dataset consisting of six different paths of varying lengths has been acquired. For each path, multiple sequences have been recorded, retracing the same path several times under different lighting and weather conditions. The lighting conditions are day and night, while the weather conditions are sunny, overcast and rainy, obtaining five combinations: day/sunny, day/overcast, day/rainy, night/clear and night/rainy[1]. In this first version, only RGB-D images and the corresponding ground truth poses were captured from GTA V. The dataset was acquired at a frame rate of 10 frames per second

[1]Night/overcast is ignored because there is no significant difference compared to night/clear.

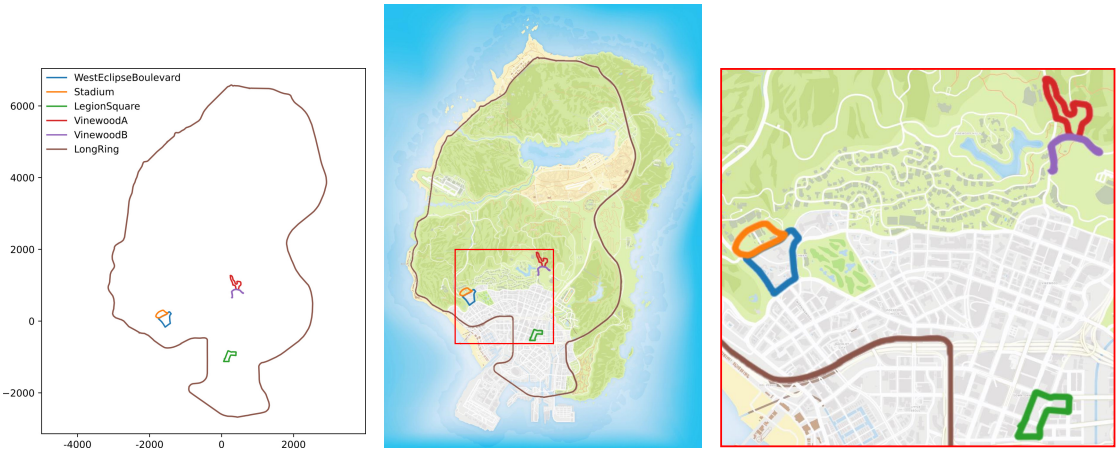
	Day			Night		Total
Path	**Extrasunny**	**Overcast**	**Rain**	**Clear**	**Rain**	
Long Ring	16782	16627	15257	20291	15819	**84776**
Vinewood A	1349	1366	1000	1196	985	**5896**
Vinewood B	1209	1113	966	1031	1114	**5433**
West Eclispe Boulevard	1387	1334	1076	1112	1077	**5986**
Legion Square	1331	1425	1354	1268	1142	**6520**
Stadium	695	662	550	714	618	**3239**
Total	**22753**	**22527**	**20203**	**25612**	**20755**	

Table 5.3: Number of frames across the different sequences of our dataset.

(but also a higher/lower frame rate can be set) and it contains a total of 111,850 high-resolution RGB and depth images, each with dimensions of 1920x1080 pixels. Table 5.3 reports the number of frames for the different sequences contained in the dataset. Each path in the dataset provides a unique perspective of the game's broad environment, encapsulating a variety of settings, such as urban, rural and hilly landscapes (see Fig. 5.6). While five of the six routes are relatively short, one is a very long loop that encircles the entire island of Los Santos (game island). Regarding the SLAM task, this dataset offers a significant advantage: it enables the capture of extensive loops under various controlled conditions. This is crucial for testing SLAM and loop closure detection algorithms, which must be robust to handle lifelong and long-term navigation scenarios. To the best of our knowledge, no existing dataset in the community includes such extensive navigation, featuring a loop closure after a long duration and under different weather and illumination conditions. Larger loops, acquired under different conditions, will be included in the future to provide the SLAM community with a dataset for training and testing truly long-term algorithms. Fig. 5.7 shows an RGB-D image from the Legion Square sequence. The depth image is post-processed, to obtain the classic 16-bit depth encoding.

5.5 VPR dataset

A dataset designed for VPR should contain images of the same location taken from various viewpoints and during varying weather conditions. For this reason, we developed an *automatic frame selection* algorithm to create a dataset for VPR starting from the sequences of RGB images and the corresponding pose ground truth. For



(a) (b) (c)

Figure 5.6: The six paths acquired from GTA V. On the left, (a) highlights the scale and the legend to provide a clear visualization of the paths, while on the center, (b) reports the same paths on the GTA map.

(a) (b)

Figure 5.7: RGB-D image extracted from the Legion Square sequence. On the left, (a) the RGB image, on the right, (b) the corresponding post-processed depth. It is worth noting the high quality of the depth image.

clarity, the creation of the VPR dataset was divided into two distinct steps: (i) *place selection*, to select only univocal places from the sequences (see Algorithm 4) and (ii) *frame selection*, to select multiple images from the sequences that capture each place from different viewpoints or under varying weather conditions (see Algorithm 5).

Each sequence acquired using the procedure described in Section 5.4 contains a variable number of triplets. Each triplet (i, p, c) is composed of an RGB image (i), the corresponding pose ground truth (p), defined as a vector $[x, y, z, \phi_x, \phi_y, \phi_z]$, and information about the time of day and weather conditions (c).

During the place selection step (Algorithm 4), the objective is to select unique places and discard duplicate locations from the set of sequences S. To achieve

this, we determine whether a pose p is new by checking if the linear distance $\mathbf{dist}_l$ (Eq. 5.1) or the angular distance $\mathbf{dist}_a$ (Eq. 5.2) between p and the poses of all previously selected places exceed two thresholds: $t_{l_{new}}$ for the linear distance and $t_{a_{new}}$ for the angular one. If the condition is met, p is considered the pose of a new place. Algorithm 4 returns a set of poses P representing the unique places in the dataset.

After this, the automatic frame selection can take place. The objective here is to select and associate the RGB images contained in the set of sequences S with the selected places P returned by Algorithm 4. To achieve this, we determine whether an RGB image i can be associated with a place in P by verifying if the linear distance $\mathbf{dist}_l$ (Eq. 5.1) and the angular distance $\mathbf{dist}_a$ (Eq. 5.2) between the pose p (corresponding to image i) and a place p' in P are below two thresholds: $t_{l_{same}}$ for the linear distance and $t_{a_{same}}$ for the angular one. If the condition is met, i is considered an RGB image that frames the place p'. Algorithm 5 returns a dictionary that associates each place in P with the set of RGB images (and corresponding conditions) that frame it.

$$\mathbf{dist}_l(p, p') = \sqrt{(x - x')^2 + (y - y')^2 + (z - z')^2}. \tag{5.1}$$

$$\begin{aligned} \mathbf{dist}_a(p, p') &= |(\mathbf{angle}(\phi_z) - \mathbf{angle}(\phi'_z) + \pi) \,\%\, 2\pi - \pi|, \\ \text{where } \mathbf{angle}(\phi) &= \begin{cases} \phi + 2\pi & \text{if } \phi < 0, \\ \phi & \text{otherwise.} \end{cases} \end{aligned} \tag{5.2}$$

Note that in our algorithm, the angular distance $\mathbf{dist}_a(p, p')$ outlined in Eq. 5.2 involves only the rotation around the z-axis, which corresponds to the 2D plane, as the rotations around x and y axes are not relevant in this context.

To ensure proper place selection, the thresholds must follow the rules specified in Eq. 5.3. In fact, if the threshold $t_{l_{same}}$ (or $t_{a_{same}}$) is larger than $\frac{t_{l_{new}}}{2}$ (or $\frac{t_{a_{new}}}{2}$), the same RGB image could be associated with two or more places, which is undesirable for place recognition. In other words, the constraints defined in Eq. 5.3 ensure that in Algorithm 5, no more than one place in P can satisfy the condition of line 9.

$$t_{l_{same}} < \frac{t_{l_{new}}}{2}, \;\; t_{a_{same}} < \frac{t_{a_{new}}}{2}. \tag{5.3}$$

To use the acquired dataset for VPR experimentation, we applied the two al-

Algorithm 4 Automatic place selection for VPR

procedure PLACE_SELECTION(S, $t_{l_{new}}$, $t_{a_{new}}$)
  $S \leftarrow$ set of sequences
  $t_{l_{new}} \leftarrow$ threshold for linear distance for new place
  $t_{a_{new}} \leftarrow$ threshold for angular distance for new place
  $P \leftarrow \emptyset$ //set of selected places
  for each s in S **do**
    for each (i, p, c) in s **do**
      if $\forall p' \in P$,$\mathbf{dist}_l(p, p') \geq t_{l_{new}} \vee$
      $\mathbf{dist}_a(p, p') \geq t_{a_{new}}$ **then**
        $P \leftarrow P \cup \{p\}$
      end if
    end for
  end for
  return P
end procedure

gorithms to the six paths, thus obtaining a dataset specifically suited for this task. Only the sequences acquired under day/sunny conditions (see Section 5.4) were passed to Algorithm 4, while all the sequences are passed to Algorithm 5.

We set $t_{l_{new}} = 100m$, $t_{a_{new}} = 90$, $t_{l_{same}} = 10m$ and $t_{a_{same}} = 20$. We chose $t_{l_{new}}$ and $t_{a_{new}}$ to achieve a good trade-off between the number of selected locations and their variability. Reducing $t_{l_{new}}$ and $t_{a_{new}}$ results in the algorithm selecting more locations, at the cost of introducing more similar or even identical places. For instance, a building seen from a distance and then re-observed up close may be recognized as the same place by humans, which could affect place recognition. The resulting dataset comprises 20919 RGB images, divided into 326 places, with approximately 65 images for each location, captured in its surroundings under five different conditions. For each location, the associated images consistently frame the same recognizable area, demonstrating that the algorithm performs effectively without human supervision (see Fig. 5.8).

5.6 Experiments

We conducted several experiments to evaluate the quality of the synthetic data extracted from GTA V, aiming to demonstrate that these data can be used instead of or in addition to real data. In the following, the results of the experiments on the two tasks selected in this chapter are reported.

Algorithm 5 Automatic frames selection for VPR

procedure FRAMES_SELECTION(S, P, $t_{l_{same}}$, $t_{a_{same}}$)
  $S \leftarrow$ set of sequences
  $P \leftarrow$ set of selected places
  $t_{l_{same}} \leftarrow$ threshold for linear distance for same place
  $t_{a_{same}} \leftarrow$ threshold for angular distance for same place
  $I \leftarrow \emptyset$ //dictionary $\{p : \{(i, c)\}\}$
  for each s in S **do**
    for each (i, p, c) in s **do**
      if $\exists p' \in P$,$\mathbf{dist}_l(p, p') < t_{l_{same}} \wedge$ $\mathbf{dist}_a(p, p') < t_{a_{same}}$ **then**
        $I[p'] \leftarrow I[p'] \cup \{(i, c)\}$
      end if
    end for
  end for
  return I
end procedure

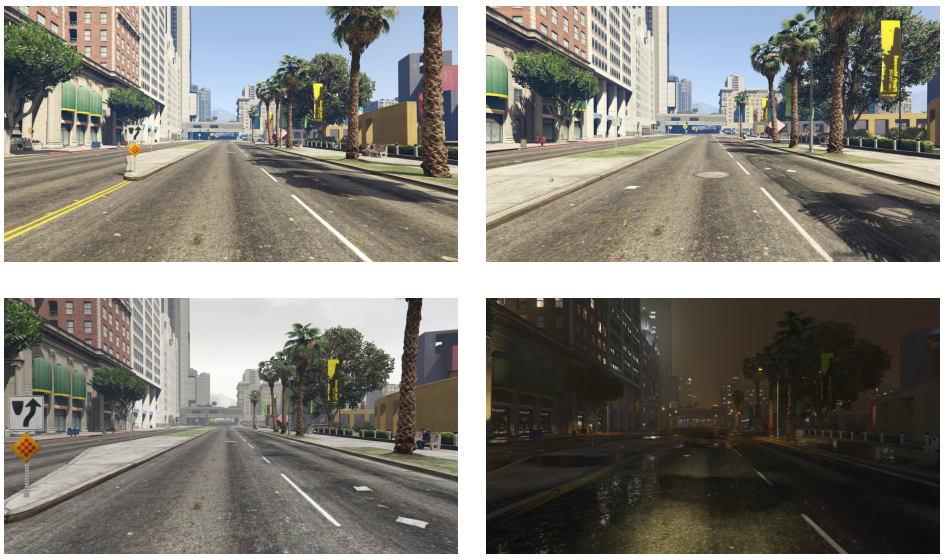


Figure 5.8: Images associated to the same place by the automatic frame selection algorithm.

5.6.1 Visual Place Recognition

In the context of VPR, inspired by [134], we approached the problem as an image retrieval task. Since this study does not aim to propose a new state-of-the-art method for VPR, but rather to demonstrate the usability of GTA V synthetic data for this task, we created a simple but non-trivial scenario: given a day/night image database, when a day (or night) image is presented to the system, it must retrieve the corresponding night (or day) image. Note that, in this scenario, only RGB

images are used.

For these experiments, we used three datasets:

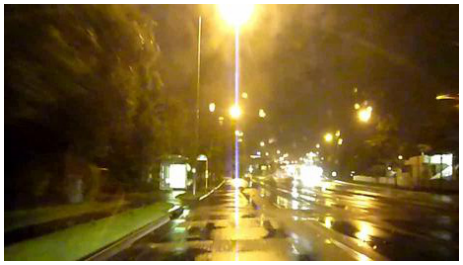
- Alderley [135], a real dataset of paired day/night RGB images captured by a camera on top of a car in an urban environment. For the VPR task, 125 pairs of frames have been manually selected, to obtain enough diversity between places (Fig. 5.9a).
- Tokyo 24/7 query V3 [134], a subset of the Tokyo 24/7 dataset reported in Section 5.2, a real dataset of 375 places with day/night images captured in Tokyo. This dataset is specifically designed for VPR, therefore no frame selection is required (Fig. 5.9b).
- GTA V, the VPR synthetic dataset acquired in our work using GTA V (see Section 5.3). For each place, the automatic algorithm described in Section 5.5 is used to select frames for VPR and to pair day/night images of a place. A total of 326 different places have been automatically selected.

Fig. 5.9 shows image pairs from the three datasets.

To ensure a fair comparison of the results obtained using the Alderley or GTA V datasets during training, only the first 125 locations automatically selected by GTA are used, providing an equal number of places from both real and synthetic datasets. The Tokyo 24/7 dataset was not used for training the system, but only for testing.

We trained a deep neural network (see Fig. 5.10) to perform the task. Specifically, we used a triplet neural network consisting of three replicas of the same network with shared weights. The network is composed of a convolutional backbone (ResNet50 [153] in our case) pre-trained on ImageNet [154] with frozen weights serving as the feature extractor, and a final Multi-Layer Perceptron (MLP) comprising two fully connected layers to project the features onto a new latent space. The purpose of the final MLP is to bring the latent representations of the positive pairs (day image with the corresponding night and vice versa) closer together in latent space, while moving the latent representations of the negative pairs farther apart. The metric used to compute the distance between the embeddings is the classic cosine distance; during training, the cosine distance between positive pairs must be minimized, while the cosine distance between negative pairs must be maximized.

To train the neural network in this manner, triplets consisting of an anchor image a, a positive image p, and a randomly chosen negative image n are created. The neural network is used to compute the embeddings of the three images, obtaining e_a, e_p and e_n respectively.



(a) Alderley

(b) Tokyo 24/7

(c) GTA V

Figure 5.9: Example of images from the three datasets, captured under different light/weather conditions.

Given the distance function d (cosine distance) and a margin ϵ, the loss function is computed as shown in Eq. 5.4.

$$\mathcal{L}_{(e_a,e_p,e_n)} = max(d(e_a, e_p) - d(e_a, e_n) + \epsilon, 0), \tag{5.4}$$

which corresponds to

$$\mathcal{L} = \frac{1}{k}\sum_{i=1}^{k} \mathcal{L}_{(e_a^i,e_p^i,e_n^i)} \tag{5.5}$$

on each minibatch with k samples.

In this context, if e_a^i is a day image, both e_p^i and e_n^i must be night images and vice versa.

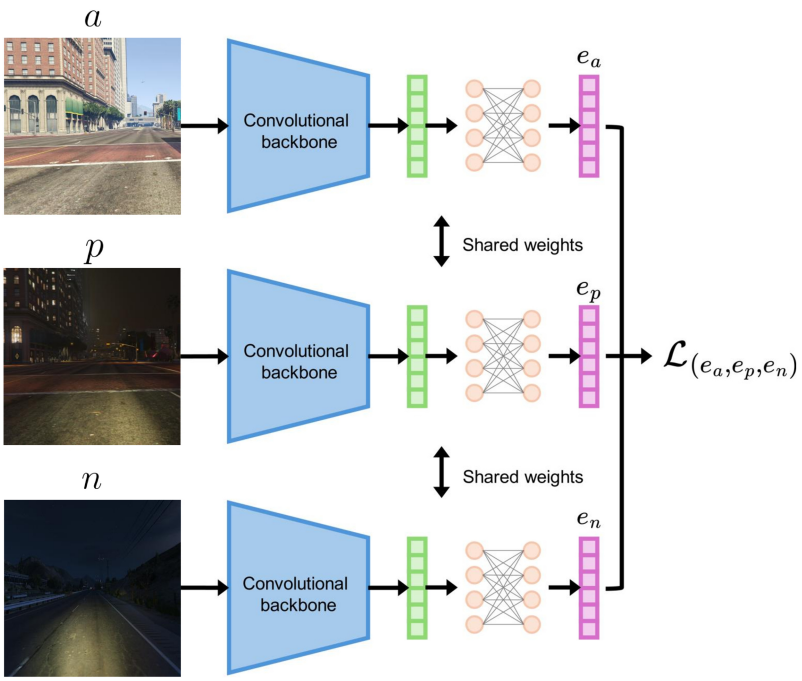
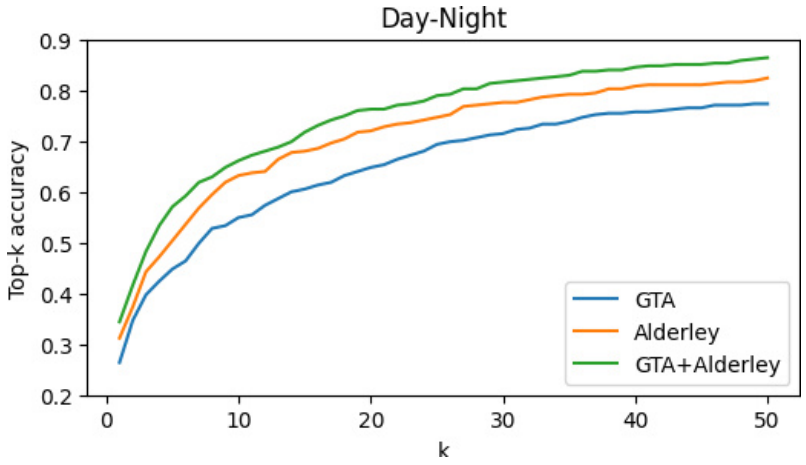


Figure 5.10: Training scheme for the VPR experiments.

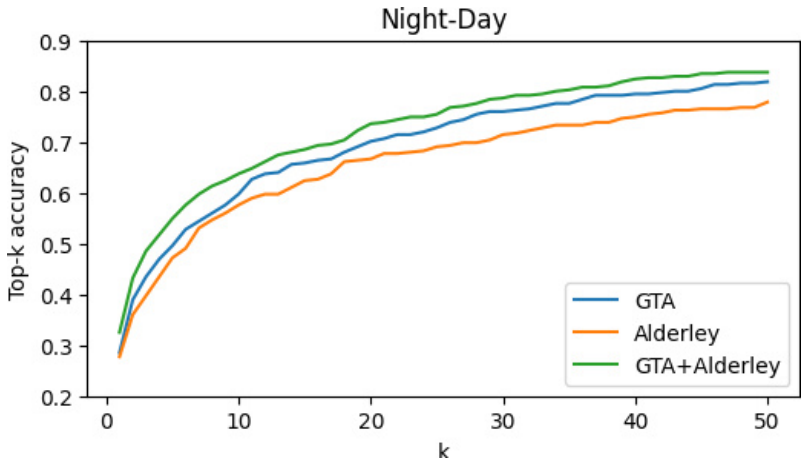
Alderley and GTA V images (which have an aspect ratio of 16:9) have been randomly cropped during training using a window of aspect ratio 4:3 to ensure the same aspect ratio of the images from Tokyo 24/7. The random crop is useful not only to avoid different stretches between different datasets but also to introduce further variability in the data during training, since the position of the window is computed on the fly for each sample. Subsequently, all images were resized to 224x224 pixels to be compatible with the network.

We performed three experiments for VPR: (i) using Alderley as the training set (only real images), (ii) using the GTA V as the training set (only synthetic images), and (iii) using both Alderley and GTA V as the training set (real and synthetic images). In the third experiment, negative images (n_i) were selected from the same dataset as the positive images (a_i), ensuring that if a_i was real, then n_i was also real and vice versa. The Tokyo 24/7 dataset was always used as the test set. We used Tokyo 24/7 day (night) images as database and each night (day) image of Tokyo 24/7 is presented to the network as query, to retrieve the most similar day (night) images. The evaluation metric employed was top-k accuracy, which measures whether the correct prediction y_i is among the top-k predictions, and we varied the values of k to assess how the recall changes when k increases (see Fig. 5.11).

The results obtained using GTA V and Alderley as training sets are comparable, with GTA V even outperforming Alderley in some instances, as illustrated in Fig. 5.11b. This demonstrates that synthetic RGB images from GTA V are sufficient



(a)



(b)

Figure 5.11: Results of the VPR experiments. On the top, (a) shows the recall at different k using day images of Tokyo 24/7 as query and night images as database, while, on the bottom, (b) reports the results of the opposite setting, using night images as query and day images as database.

to train a neural network for this task. Moreover, the results show that combining synthetic and real data during training further improves retrieval performance. Although this may seem intuitive, it has significant implications for the dataset acquisition process. A new dataset can potentially be created using a small number of real images, whose acquisition requires time, energy, and possibly cost, while the majority of the dataset can be composed of synthetic images, which enrich the real data and increase the number of samples at approximately no cost, but resulting in improved system performance.

Sequence	ATE (m)
LegionSquare	1.49
Stadium	1.38
VinewoodA	1.16
VinewoodB	0.94
WestEclipseBoulevard	1.47

Table 5.4: Absolute Trajectory Errors for the five sequences run on RTAB-Map.

5.6.2 SLAM

An additional experiment was conducted in the context of visual SLAM, a more challenging task that requires precise sensor acquisition. We evaluated the paths detailed in Section 5.4 using RTAB-Map [5], a well-known graph-based SLAM approach. To compute the itinerary traced by the car, RTAB-Map requires RGB-D images, so depth images are also used in this experiment. This is crucial because, to achieve accurate odometry and correctly detect loops, the depth images must be very precise, otherwise the results obtained from the ground truth of the poses would be completely wrong. Fig. 5.12 shows the trajectories estimated by RTAB-Map compared to the ground truths. The estimated paths are extremely accurate, with only minor physiological inaccuracies of SLAM. This highlights that GTA V data can be used for this task, proving that RGB and depth images are as accurate as, if not more accurate than, real-world data. While this represents a significant strength, it also poses a disadvantage. In real-world scenarios, robots equipped with real sensors have to deal with noise and measurement errors. Consequently, especially for deep learning-based solutions, the optimal approach might be to combine real-world data with synthetic data from GTA V, or introduce noise in the synthetic depth images, to obtain a dataset that incorporates the variability and imperfections inherent in real-world data.

The metrics used to measure the goodness of the results are based on the accuracy of the localization, which typically also reflects the accuracy of the map. The most common metric is the Absolute Trajectory Error (ATE) [155] which measures the absolute distances between the estimated and the ground truth trajectory, but also other metrics exist [155, 112].

Table 5.4 reports the ATE for the five sequences. The errors observed are relatively small, perfectly in line with the expected errors for paths of comparable size with real data.

In this experiment, RTAB-Map was unable to compute the entire itinerary of the LongRing path due to poor odometry and tracking failures in several segments of the sequence. This further emphasizes the need for such a dataset: exploring

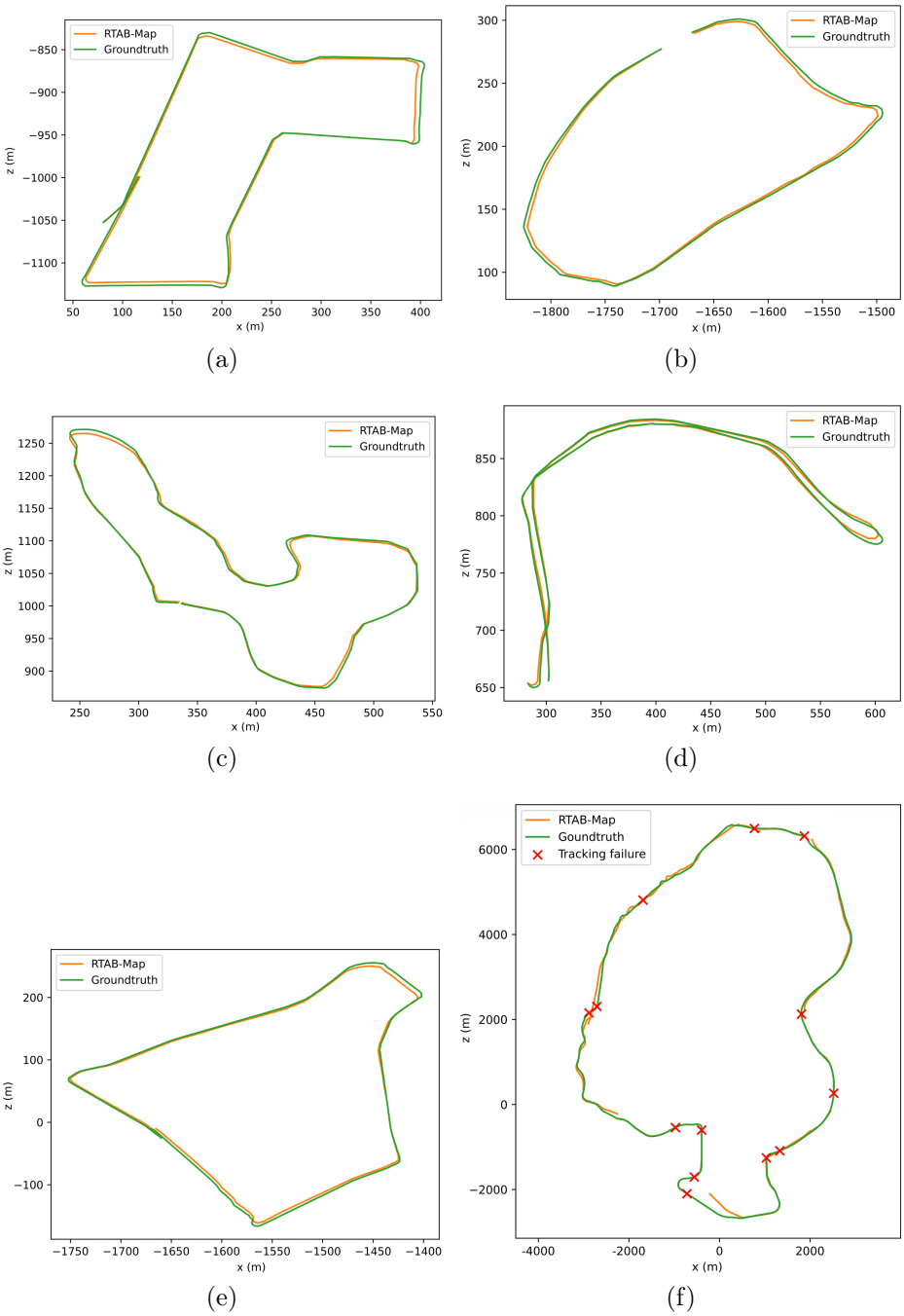


Figure 5.12: Paths estimated by RTAB-Map compared to the ground truth. In order, (a) is LegionSquare, (b) is Stadium, (c) is VinewoodA, (d) is VinewoodB, (e) is WestEclipseBoulevard and (f) is the LongRing. The red crosses on the LongRing path highlight where the tracking is lost.

a wide range of locations, including those with minimal texture or where lighting conditions change (e.g., a car tunnel), requires robust tracking and odometry techniques. Common SLAM solutions, which are based solely on images, are insufficient to handle such scenarios. In Fig. 5.12, the LongRing path was created by align-

ing with the ICP algorithm the subparts obtained from RTAB-Map and the aligned paths are as accurate as for the other sequences, further highlighting data from GTA V is suitable for this task. The red crosses highlight where the tracking is lost; in these points, the images (i) are mainly composed of untextured regions (e.g., the sky, a uniform road), causing common algorithms to fail in detecting and tracking a sufficient number of features to estimate accurate odometry, or (ii) exhibit abrupt changes in illumination conditions, preventing feature tracking (e.g. when entering or exiting a car tunnel).

5.7 Conclusions and Future Work

In this chapter, we introduced a novel synthetic dataset acquired from the famous video game GTA V. We evaluated the dataset on two tasks where the use of synthetic data is uncommon: VPR and SLAM. The experiments demonstrated that data from GTA V can be used as a substitute for, or in conjunction with, real data to improve performance in both tasks. Furthermore, the SLAM experiments revealed the limitations of common visual SLAM systems, which are not able to deal with exploration in the dark or in untextured environments. In the context of VPR, we also proposed a simple but effective algorithm for generating a VPR dataset from image sequences and poses ground truth without requiring human supervision. The proposed pipeline facilitates the creation of large datasets under controllable conditions that are challenging to capture in the real world and at almost no cost. This methodology allows the community to rapidly enrich the plethora of existing datasets for each task with new data acquired under novel conditions or scenarios that are difficult to acquire in real-world settings. Despite the numerous advantages of using synthetic data from GTA V, including scalability, repeatability, and control over conditions, the approach is not without limitations. The absence of realistic sensor noise and artifacts, perfect ground truth, and game-specific biases may hinder generalization to real-world scenarios. Moreover, the physics engine and environmental interactions do not fully reflect real-world complexities, introducing the well known simulation-to-reality gap. Future work will focus on the creation of novel datasets of unprecedented size for each task reported in 5.1.2, using GTA V and the tools and pipeline described in 5.3. The datasets will be generated under new conditions and scenarios that are not typically considered by the community, and will be used to further test the method exposed in the previous chapters.

Chapter 6

Contribution to the community

Scientific research should not only advance theoretical understanding but also provide tangible assets to the broader community. In this chapter, we present the practical contributions resulting from our work, with a particular focus on the tools, codebases, and systems that have been publicly released to support reproducibility, foster collaboration, and accelerate further developments in the field.

These contributions include open-source implementations of the methods developed and reusable software components embedded into RTAB-Map, but also applicable to other complete SLAM solutions. All resources have been designed and documented with usability and accessibility in mind, ensuring they can serve both researchers and practitioners. You can find the entire code along with the instructions to run it at https://github.com/MI-BioLab/continual-learning-regions.

To contextualize these contributions, the chapter begins with a detailed overview of the system architecture developed during this research. This architectural description provides the necessary technical foundation for understanding how the released components fit together, how they implement the core ideas discussed in previous chapters, and how they can be extended or integrated into other workflows.

The accompanying diagrams and UML models have been intentionally simplified to maintain clarity and avoid overwhelming the reader with low-level implementation details. Here, we report only a part of the fields and methods of the classes, fundamental to understanding their role and functioning, avoiding useless details. Nevertheless, the textual descriptions are comprehensive: they cover the purpose of each component, the main implementation choices, and practical usage guidelines. To embed our methodology into RTAB-Map, we did not remove anything from the traditional RTAB-Map workflow; we simply added or changed parts of it to handle continual learning and graph clustering.

6.1 The region package

Our modified RTAB-Map version includes a new package, called `region`, which contains all the classes and utility functions used in our methodology. Fig. 6.1 shows the architecture of the package, which is a broader container for other sub-packages, explained in detail in the following. The sub-packages involved in the continual learning part (e.g., model adaptation, replay memory storage policies and sampling, etc.) are a porting of Avalanche packages from PyTorch to LibTorch, to replicate their functionalities in a C++ environment. Inside the `region` package, there are also two fundamental classes: `TrainThread` and `ContinualLearning`, which are responsible for the continual learning training step.

The `TrainThread` class handles the training thread, which is triggered at each training stage to run in parallel with the main thread. The model used in inference is copied here and then trained. It also contains an instance of the RTAB-Map class `DBDriver`, since it has to update the replay memory in the database after each training stage and read it to reactivate past stored data in multi-session scenarios. It handles the parameters of the training (e.g., number of epochs, batch size, loss function, etc.) and the methods to train the model. Specifically, `isTraining()` is a method called from the main thread to know if the training of the model on the current experience has ended, and the new model, with possibly new neurons, can be copied on the main thread to be used in inference.

The `ContinualLearning` class handles the continual learning part, and it contains a `TrainThread` instance to run the training thread when a new experience of fixed capacity is received, with the method `train`. The method `predict` is instead used by the model in inference to perform the prediction on the current observation. The method `topKRegions` is called in inference mode to return the ids of the top-k regions predicted by the model given the current observation.

6.1.1 Clustering package

In this sub-package (in Fig. 6.2), only the class `Region` is present. It contains all the useful methods and fields to perform clustering. It is implemented as a static class, because regions are recreated at each iteration. This because RTAB-Map nodes are dynamically moved from WM to LTM and vice-versa, so to avoid complicated updates, at each iteration we recompute the clustering information only for the regions that are involved in the clustering of the current node with the method `create`. This greatly simplifies the management of node reassignment, without penalizing performance.

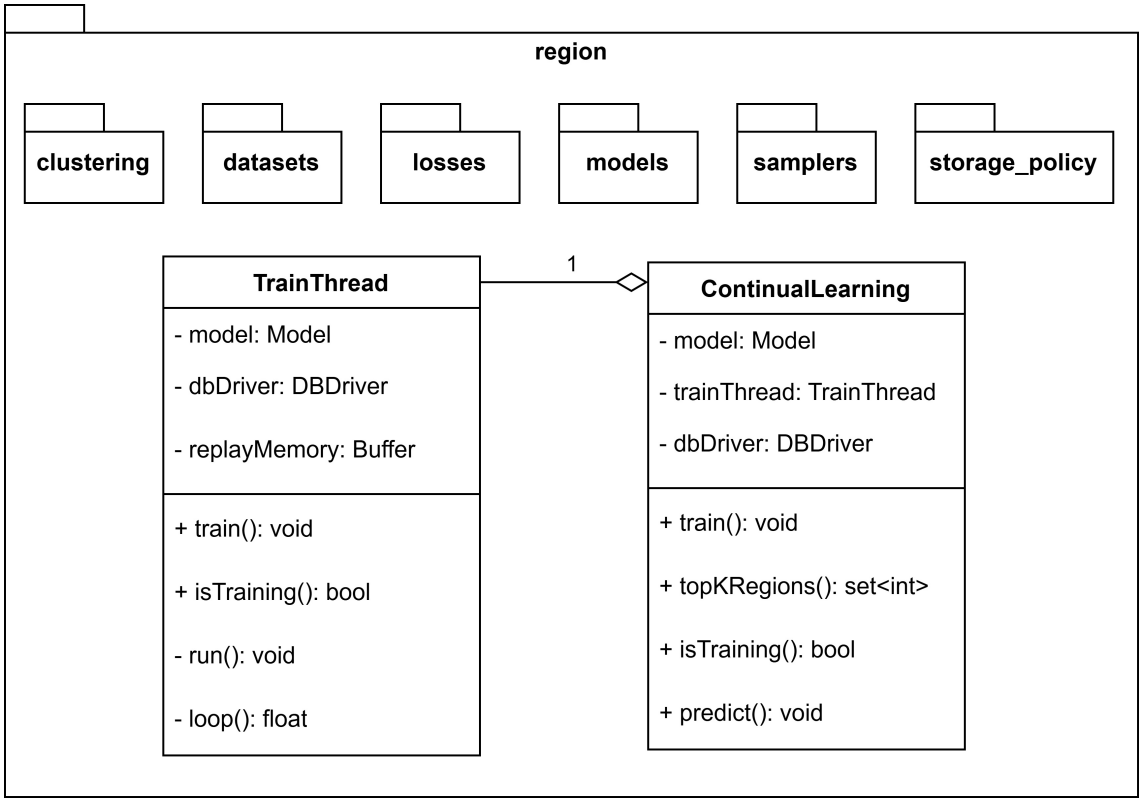


Figure 6.1: The `region` package, the main package added to RTAB-Map. It contains the other sub-packages and the two main classes `TrainThread` and `ContinualLearning`.

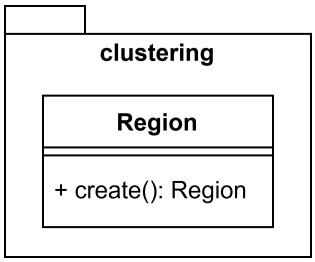


Figure 6.2: The `clustering` package, responsible for the region creation and the dynamic clustering.

6.1.2 Datasets package

In this package are the classes used to handle the LibTorch datasets for training, which extend the `torch::data::Dataset` class (Fig. 6.3). `ExperienceDataset` represents an experience, so the data are composed of images and labels, other than identifiers to the data, to change the label of a particular node if it is reassigned to a different cluster. This is the input dataset used by llf to extract low-level features. On the other hand, `LatentDataset` represents a dataset of latent features, which is used in replay memory and is the input of cls. Here, to help with the

continual learning training, two parameters are kept: `classesInDataset`, which holds the classes seen so far in the dataset (even in multi-session scenarios), and `samplesPerClass`, which is a tensor in which each entry contains the number of samples for this class (i.e., the corresponding region) in the dataset.

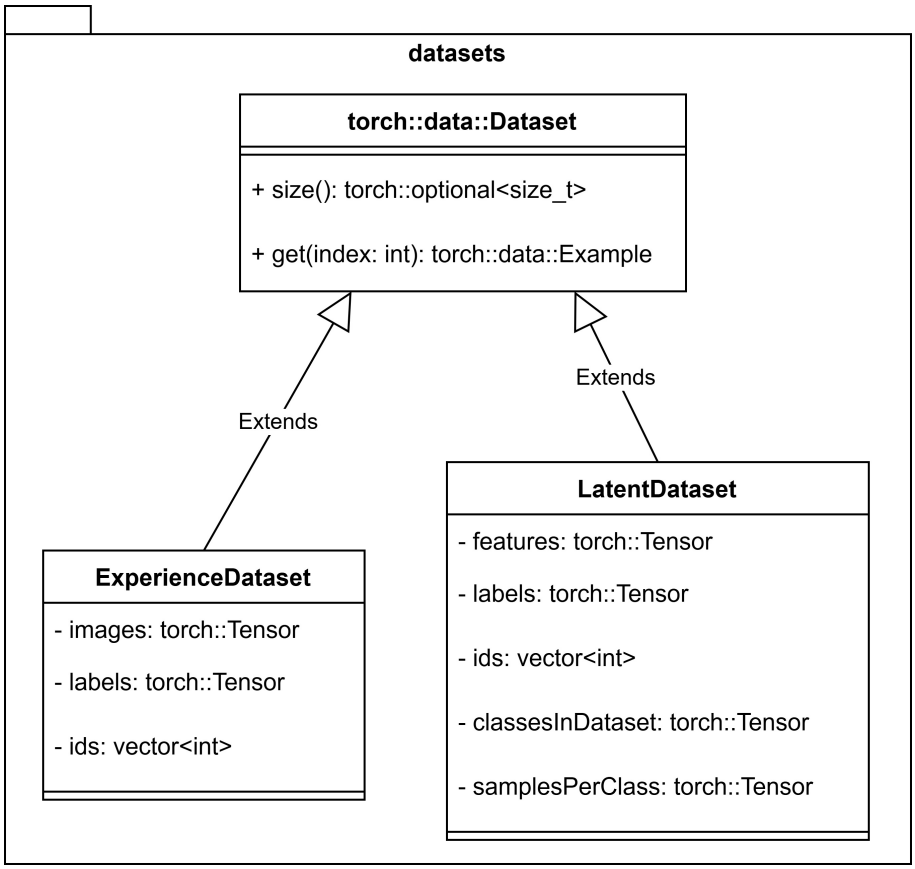


Figure 6.3: The `dataset` package, responsible for the dataset types used in the training process.

6.1.3 Losses package

This package contains the basic classes for the loss functions used in our experiments (Fig. 6.4). Starting from a base abstract class, `CustomLoss`, which extends from `torch::nn::CrossEntropyLossImpl`, we specialized it into the `CrossEntropyLoss` class and the `FocalLoss` class.

6.1.4 Models package

Inside the `models` package are all the classes used to build the neural network model (Fig. 6.5). All of them extends the `torch::nn::Module` class. The weights and the parameters of the model used in RTAB-Map were exported from PyTorch, since

they are those of the ResNet18 pretrained on Places365, so we recreated the same structure used in PyTorch, to enable compatibility. The `BasicBlock` class is the basic ResNet block, used to compose the entire ResNet18 structure. The class `FreezedPart` is the frozen part of the network, llf, composed of the first three ResNet blocks, while `TrainablePart` is the trainable part cls, composed of the last ResNet block. The `FeatureExtractor` is composed of these two objects, with two methods, `extractFreezedFeatures` and `extractTrainableFeatures` to use llf to extract features from images and cls to extract features from the latent replay layer output, respectively. The `IncrementalLinear` class is just an extension of a classical LibTorch `Linear` layer, with the `adapt` method to add new neurons to the classification head if new regions are discovered. Finally, the class `Model` encloses all these concepts, along with the method `isTrained` to check if the model has been trained at least once (i.e., if at least one neuron is present in the classification head). This is necessary because in inference we cannot use the model until at least one neuron is present in the final layer.

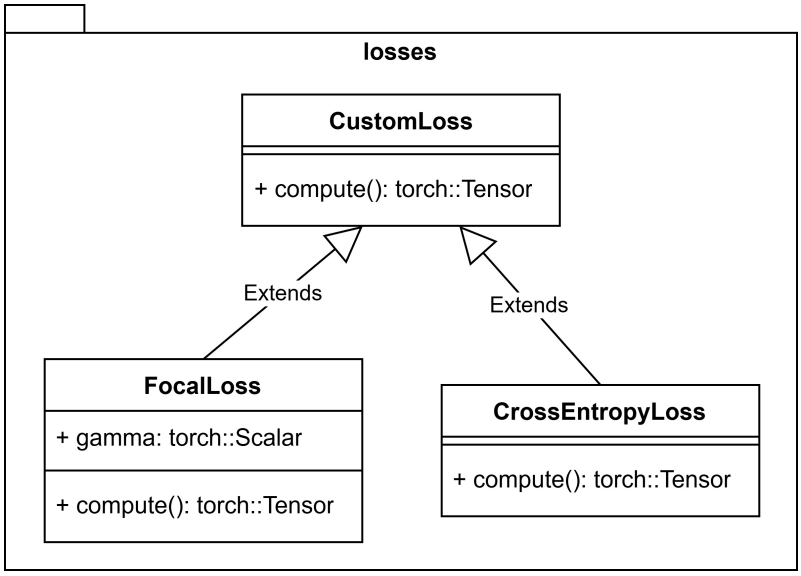


Figure 6.4: The `losses` package, responsible for the loss functions used in the training process.

6.1.5 Samplers package

The `samplers` package (Fig. 6.6) contains the `ReplaySampler` class, which is used to handle sampling from the new experience and the replay memory for the current training stage. Depending on the batch size used for the current experience and the replay memory, and depending on the number of samples in the current experience

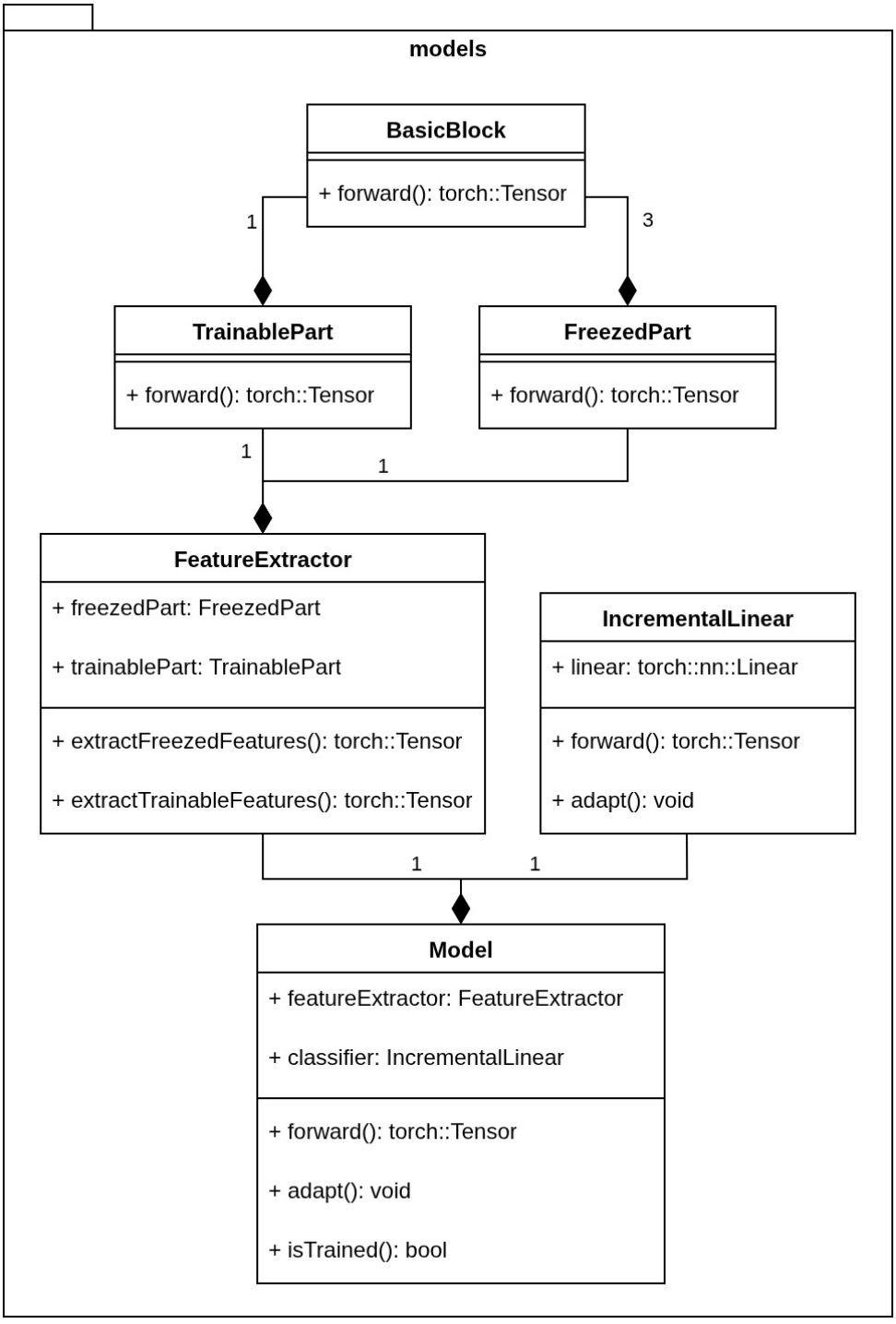


Figure 6.5: The `model` package, responsible for the neural network building blocks.

and the replay memory, it randomizes the samples and decides which ones to pick and put in the current batch, mixing them accordingly.

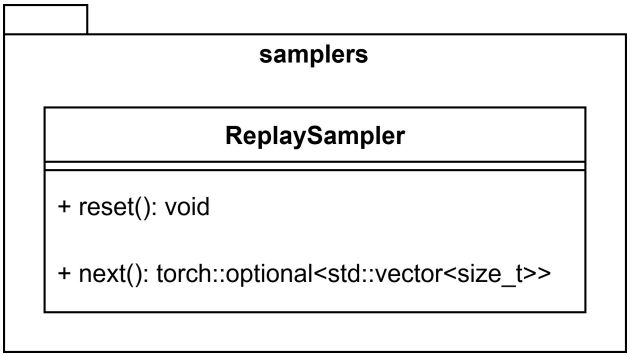


Figure 6.6: The `samplers` package, responsible for the sampling of the batches during training.

6.1.6 Storage_policy package

The classes in this package handle replay memory updates (Fig. 6.7). Depending on the number of samples, the maximum size of the replay memory, and the policy used, these classes decide which samples to keep in memory and which to discard after each training session. The base class `Buffer` represents a generic buffer that contains samples in the form of a `LatentDataset`. It is specified into two subclasses: `ReservoirSamplingBuffer`, which implements a reservoir sampling policy, and `ClassBalancedBuffer`, in which the same number of samples for each class is maintained, and inside each class bucket, a reservoir sampling policy is used.

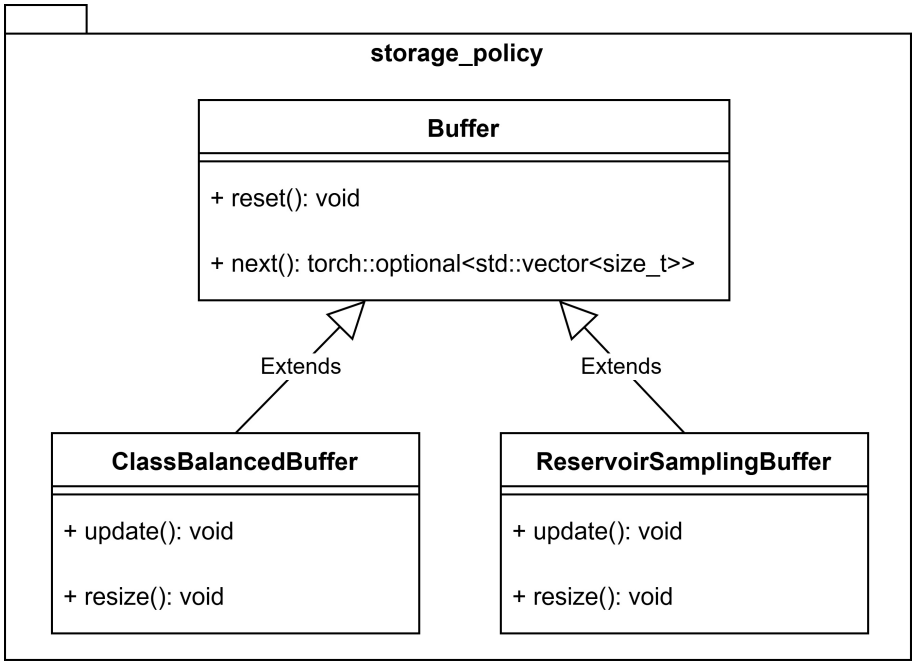


Figure 6.7: The `storage_policy` package, responsible for the storage policies of the replay memory.

6.2 RTAB-Map modifications

Since our method is only used to modify the RTAB-Map transfer-retrieval process, we did very few specific modifications to the original code.

The first modification involves the addition of the `regionId` in the `Signature` class (Fig. 6.8a), which contains all the important information and data about a node. The main class is `Rtabmap`, which incorporates the main cycle in the `process` method, in which transfer-retrieval, loop closure detection, and graph optimization are performed. It holds an instance of the RTAB-Map class `Memory`, in which are implemented methods to handle transfer-retrieval. Here, we added an instance of the `ContinualLearning` class, to handle online training and inference of the model and different methods to create experiences as new observations are acquired, to retrieve nodes depending on the belonging region as predicted by the network and to perform clustering (Fig. 6.8c). In Fig. 6.8b only the main methods are shown for clarity. More specifically:

- `setRegionToSignature` is used to set the `regionId` in a particular node.
- `addInExperience` is used to add the current observation into the current experience
- `reactivateTopKRegions` allows to retrieve from LTM the nodes belonging to the top-k regions, according to the network prediction.
- `getRegion` recreates the region corresponding to the given id.
- `moveFromRegion` allows reassigning a node to a different region, if required by the clustering algorithm.

Other methods have been added in this class, but they are related to these concepts and not essential to understanding the functioning of the system. Finally, `DBDriver` was also modified (Fig. 6.8d), enriched with new methods to handle the transfer-retrieval based on the `regionId`, and clustering and replay memory for multi-session scenarios. Here we faced another problem: to correctly compute the clustering information for a given region, we need to have in WM all the nodes involved in the clustering process, that are the nodes of the current region and the nodes of adjacent regions that could be reassigned to other regions, but they could be in LTM. To avoid retrieving at each iteration all the information about these nodes (e.g., observations, features, timestamp, etc.) that is time-consuming, we added the method `loadClustering` to perform a light retrieval, specifically used only for clustering, to retrieve only the spatial information of the nodes involved from LTM,

which is the only information needed for clustering. In this way, the retrieval of hundreds of nodes in each iteration is very fast ($\sim 5ms$) and does not affect the real-time nature of the process compared to the complete retrieval of the entire node data, which would have required several seconds. Furthermore, the nodes retrieved in this way are not added to WM, the spatial information are just retrieved, used for clustering, cached, and then deleted if not used in successive iterations, not affecting at all the main transfer-retrieval process.

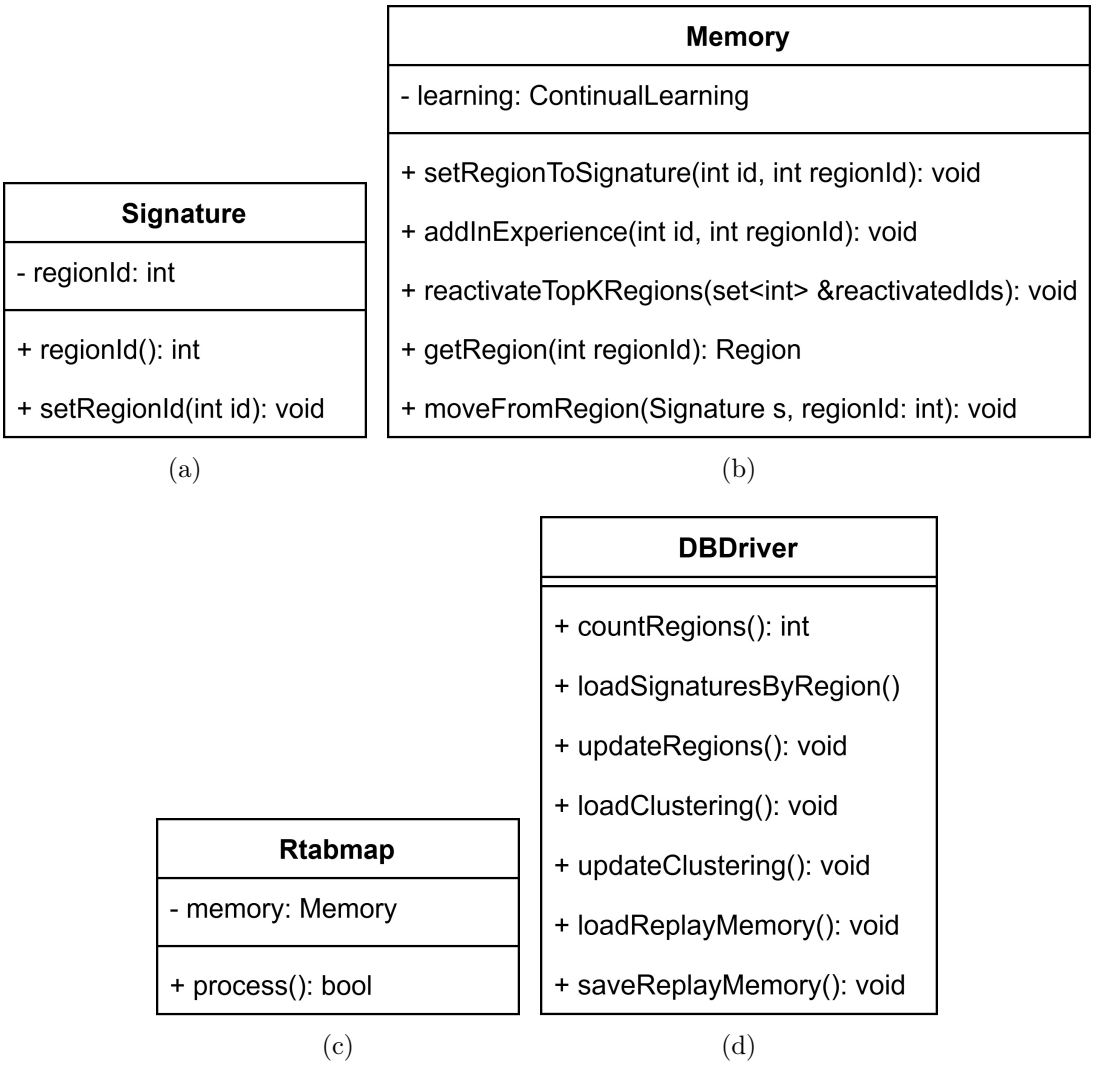
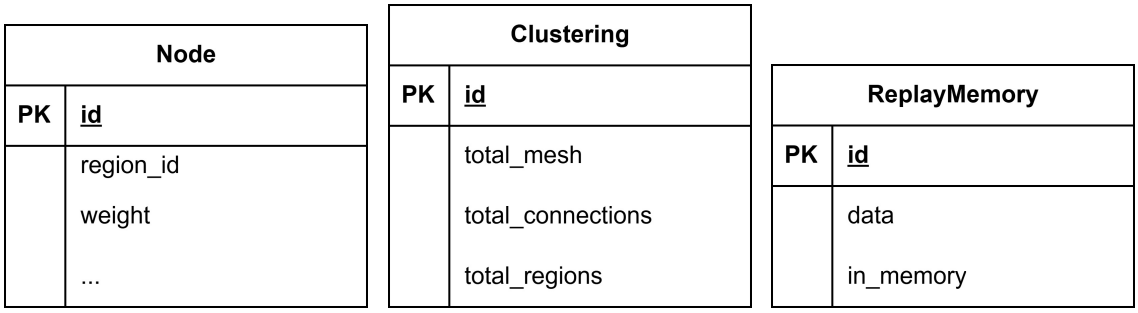


Figure 6.8: The RTAB-Map classes modified to incorporate our methodology.

6.2.1 Database modifications

We also modified the LTM structure, which is implemented as an SQLite local database, to incorporate our method. Table `Node` is modified as shown in Fig. 6.9a,

to include also the column `region_id`, in which we store the ids of the regions assigned by the clustering algorithm. We also added two new tables to handle clustering and replay memory in multi-session scenarios. Table `Clustering` in Fig. 6.9b is only used to maintain the global information about the clustering: the total mesh ϕ, the total number of links in the graph, and the total regions created, to allow the system to restart from these data when a new session is launched. Table `ReplayMemory` in Fig. 6.9c stores the replay memory features with the column `in_memory`, which is a boolean that specifies whether the data is currently in memory or if it was removed by the specific replay memory update policy. When a new session is launched, the replay memory is immediately restored with the replay memory used from the last session. Maintaining all the data in the database enables more sophisticated policies; for example, data belonging to specific samples can be reactivated if needed, but this is left for further extensions.



(a) (b) (c)

Figure 6.9: The RTAB-Map database tables modified to incorporate our methodology.

6.2.2 Main cycle modifications

For a correct understanding of the functioning of the new RTAB-Map version, in Fig. 6.10 we reported how the main RTAB-Map class was modified, in order to embed all the code described before. The new parts introduced are highlighted in red. Inside the `init` method, all parameters are loaded, along with the neural network weights. In fact, at the end of each training stage on a new experience, the model weights are stored, to reload them in multi-session scenarios. In this way, the network preserves past knowledge from previous explorations. If no stored weights are present, the network starts with the Places365 weights. The method `process`, the core cycle or RTAB-Map, was composed of six main steps (in black in Fig. 6.10), we modified/introduced additional steps (in red in Fig. 6.10):

1. **Memory update**: RTAB-Map checks if the new data is good for creating and adding a new node to the graph. For example, if the robot does not move, the new node is not created. If created, it is inserted in the Short-Term Memory (STM), a memory that is filled with the newest nodes, to avoid detecting loop closures at each new observation.

2. **Bayes filter update**: this is a fundamental step to update the loop closure hypothesis probabilities as explained in [98].

3. **Retrieval of nodes from DB** : in this step, the retrieval process takes place. Here we added the model inference to predict which are the top-k most likely regions and then the retrieval of N_3 depending on this prediction. We also check here if a previous active training ended, in order to always use the newest trained model for inference.

4. **Loop closure detection**: with the nodes in WM, loop closure detection is performed.

5. **Graph optimization**: if a loop closure has been detected, the graph is optimized accordingly.

6. **Graph clustering**: after the optimization, we added the clustering of the graph. Here, the spatial information for the regions is retrieved from LTM if needed and the clustering is performed, so the class is assigned to the current observation.

7. **Experience update**: the new data is added to the current experience.

8. **Training**: if the number of samples contained in the current experience reaches the maximum size of the experience, a new training step is triggered. Here, the model is copied into the train thread and the continual learning process takes place. At the end of the training, the model weights are stored.

9. **Transfer nodes to DB**: the transfer process takes place. The WM nodes that are $\notin N_1 \cup N_2 \cup N_3$ are transferred back to LTM.

6.2.3 RTAB-Map GUI

RTAB-Map incorporates a native graphic user interface to inspect the graph under construction, in which the nodes in WM connected to the current node are shown. Thanks to this, we can also visualize what is happening in the transfer-retrieval

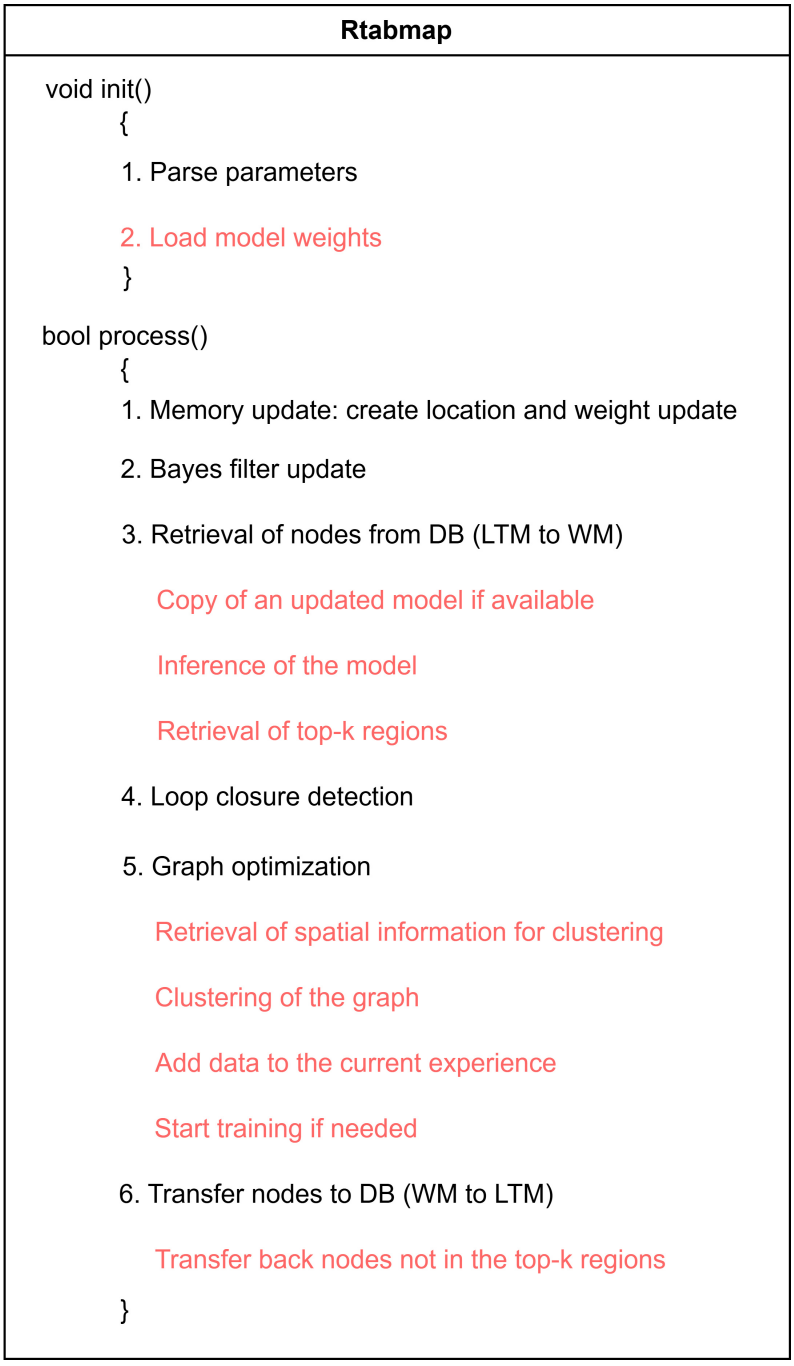


Figure 6.10: The main RTAB-Map class methods, modified to incorporate our methodology.

modified process. In the following, we report some screenshots taken from videos published in the github repository, in which the system is shown at work.

Loop closure detection

Fig. 6.11 shows an example of RTAB-Map GUI, while exploring the sequence 00 of the KITTI odometry dataset [119]. On the left column there are three images of places. The top one corresponds to the observation of the last node inserted into the graph, with the respective node id. Under it, there is the best matched observation over a threshold (chosen as a parameter), that can be highlighted with three different colors: (i) green, if accepted as global loop closure or relocalization, (ii) yellow, if accepted as local match (e.g., when retracing a known path in the opposite direction), and (iii) red, if not accepted as good match. The last image of the column corresponds to the current observation, which does not correspond to the last node created because RTAB-Map does not create a new node for each new observation, to avoid the explosion of graph complexity. In the right part, there is the graph under construction, where each circle is a node, and the lines are the edges connecting the nodes. Here, a large part of the sequence has already been explored, and the corresponding graph has already been constructed, but the RTAB-Map GUI only displays the nodes in WM connected to the current node, so you can see that at this moment only a small subset of nodes are in WM. Fig. 6.12 shows the next time instant, when the car has returned to a known place and the right nodes have been brought back to WM thanks to the modified node retrieval process, allowing RTAB-Map to recognize the match, highlighted in green in the left column. Now, on the right there are the nodes retrieved in WM, corresponding to the nodes of the top-3 regions predicted by the neural network.

Relocalization

Fig. 6.13 shows an example of RTAB-Map GUI while exploring the Week3 sequence of the USyd campus dataset [115]. Here, Week1 sequence has already been explored, but is not shown in the graph because relocalization still not occurred, so (i) the old regions are not in WM and (ii) the old graph is not connected to the current nodes in WM. Fig 6.14 shows the next time instant, when the relocalization occurs and the nodes of the old sequence are retrieved in WM, as shown in the graph displayed. Here, the nodes of the current sequence are correctly aligned with the nodes of the old sequence, allowing for a correct localization and mapping when retracing the known path.

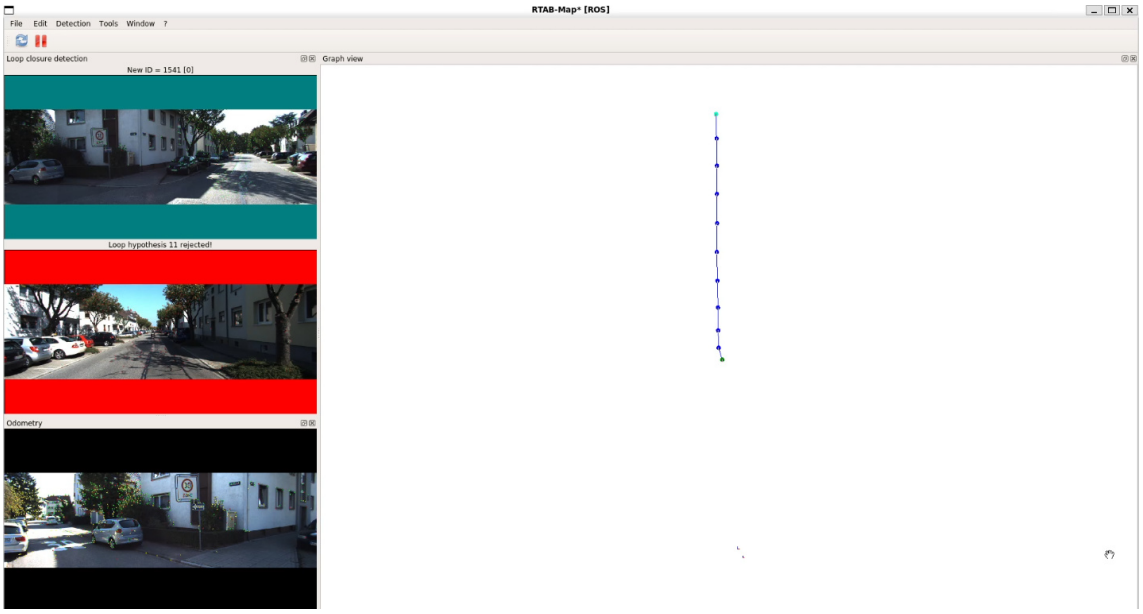


Figure 6.11: RTAB-Map GUI showing a part of the exploration of the sequence 00 of the KITTI odometry dataset.

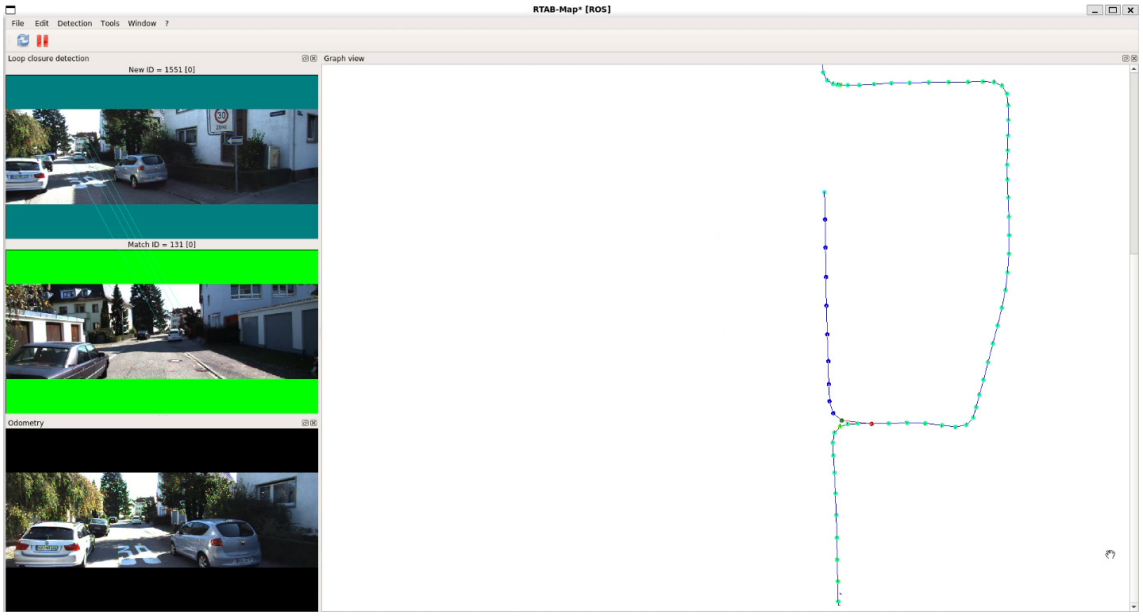


Figure 6.12: RTAB-Map GUI showing a loop closure detected during the exploration of the sequence 00 of the KITTI odometry dataset.

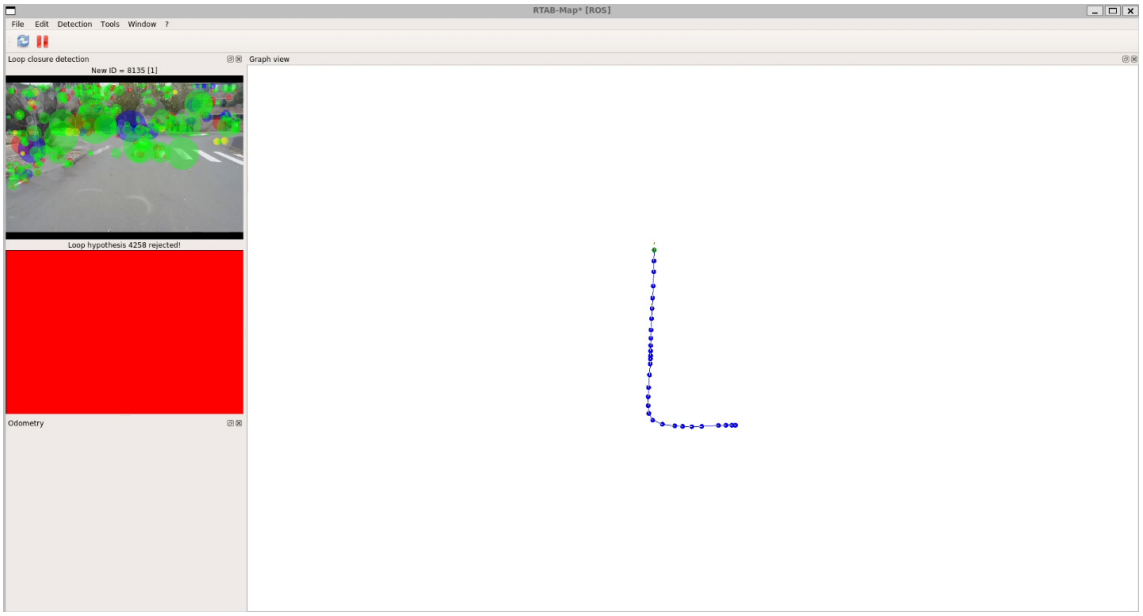


Figure 6.13: RTAB-Map GUI showing a part of the exploration of the sequence Week3 of the USyd campus dataset.

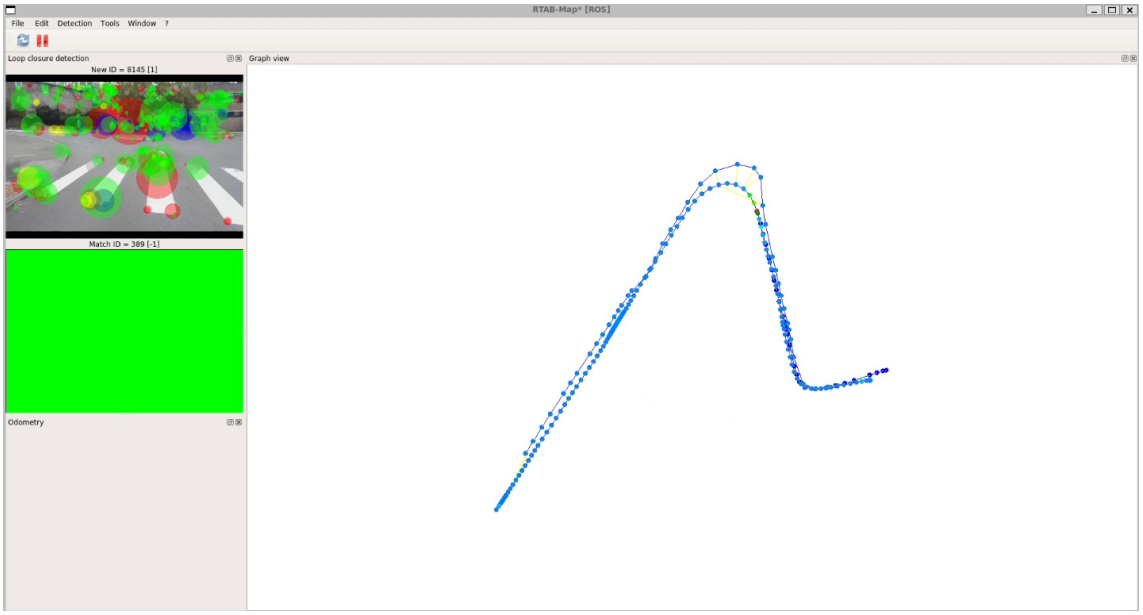


Figure 6.14: RTAB-Map GUI showing a relocalization detected during the exploration of the sequence Week3 of the USyd campus dataset.

Chapter 7

Conclusions

This thesis has explored methods to improve the robustness and scalability of SLAM systems in the context of long-term and lifelong operation. The contributions developed throughout the work can be grouped under three complementary directions: (i) the introduction of a region prediction mechanism to pre-filter map nodes for loop closure detection and long-term localization; (ii) the demonstration that continual learning can be effectively integrated into a SLAM pipeline, making it possible to adapt online without catastrophic forgetting; and (iii) the creation of a novel synthetic dataset from GTA V, which enables large-scale and low-cost experimentation under diverse and challenging conditions.

7.1 Problems addressed and contributions

Despite its simplicity, this approach addresses both robustness and scalability issues, the two open problems of lifelong SLAM, as explained in Section 1.1.

The region mechanism is effective in addressing robustness, as information taken into a region (e.g., a room in a building or a square in a city) can collectively describe that place much more robustly than single observations, which may not be robust to changes in viewpoint, illumination, or occlusion. Furthermore, as a region is explored again, new observations of this region are collected, allowing the neural network to update its knowledge over time if the appearance of that place has changed. This is in contrast with the static world assumption, improving robustness to new environmental conditions.

On the other hand, the neural network which works as a pre-filter is extremely effective in improving scalability. In fact, having a system that allows discarding the less likely regions of the map a priori is critical, especially in very large maps, because it drastically reduces the number of checks to be made for long-term lo-

calization. Imagining long-term navigations of hours or even days, the number of nodes accumulated in the map graph can be on the order of millions, and comparing the current observation with all the previous ones becomes unfeasible in real-time. Selecting at each instant only a small subset of past observations to consider makes the system extremely more scalable to truly long-term navigations.

7.2 Broader impact and future directions

The implications of these contributions extend beyond the specific implementations discussed in the thesis. By demonstrating that robustness and scalability can be simultaneously addressed within the same framework, this work points toward a general methodology for enabling lifelong SLAM. The proposed region mechanism is agnostic to the particular SLAM pipeline, making it suitable for integration into monocular, stereo, or RGB-D systems, as well as multi-modal frameworks. At the same time, the exploration of synthetic datasets shows that realistic yet controllable environments can become a powerful tool for systematically testing SLAM under conditions that are rare, costly, or impractical to reproduce in the real world.

Future research will aim to replace the baseline prediction model with more advanced architectures specifically designed for place recognition (e.g., NetVLAD or RegionVLAD), to extend clustering strategies with richer features, and to explore continual learning techniques that further reduce memory requirements while maintaining adaptation capabilities. On the dataset side, the pipeline proposed for GTA V opens the door to generating synthetic datasets of unprecedented scale and diversity, which will help stress-test SLAM systems under scenarios such as extreme lighting changes, long-term weather variations, and large-scale unstructured exploration.

Taken together, these contributions move one step closer to the long-standing vision of lifelong SLAM: a system that is able to operate indefinitely, in diverse and dynamic environments, without sacrificing either robustness or scalability.

Appendix A

Additional Material

	Dataset	USyd	St. Lucia	OpenLoris corridor
Continual learning	**Training sequences**	Week1, Week3, Week4, Week5, Week6, Week7, Week8, Week10, Week12, Week14, Week15, Week17, Week20, Week22, Week23, Week24	100909_0845, 100909_1000, 100909_1210, 100909_1410, 110909_1545, 180809_1545	corridor1 corridor3 corridor4
	# Training sequences	16	6	3
	Experience size (n_{exp})	600	600	600
	# Training experiences	165	206	23
	Test sequences	Week33, Week38, Week39, Week40, Week41, Week45, Week48, Week49, Week50, Week51	190809_0845, 190809_1410, 210809_1000, 210809_1210	corridor2 corridor5
	# Test sequences	10	4	2
	Total frames	153740	203536	20357
	Epochs (n)	4	4	4
	Batch size	64	64	64
	Replay memory batch size	64	64	64
	Learning rate	0.0001	0.0001	0.0001
	Replay memory size (n_{replay})	500	2000	500
Clustering	**Desired average cardinality** (η)	1000	1000	100
	Mesh shape factor (ϕ)	1	1	1
	Radius upper bound (m) (ρ)	100	100	10

Table A.1: Settings of E1: training/test sequences and hyperparameters used.

	Datasets	**USyd**	**KITTI**	**OpenLoris market**
	Sequences used	Week1, Week3, Week4, Week5, Week6, Week7, Week8, Week10, Week12, Week14, Week15, Week17	00, 02, 04, 05, 06, 07, 09	market1, market2, market3
	Total frames	465496	16027	22618
	# Regions retrieved (top-k)	3	3	3
Continual learning	**Experience size** (n_{exp})	500	500	500
	Epochs (n)	4	4	4
	Batch size	64	64	64
	Replay memory batch size	128	128	128
	Learning rate	0.0001	0.0001	0.0001
Clustering	**Desired average cardinality** (η)	1000	600	100
	Mesh shape factor (ϕ)	1	1	1
	Radius upper bound (m) (ρ)	100	100	15

Table A.2: Settings of E2: training/test sequences and hyperparameters used.

Bibliography

[1] H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110.

[2] Cesar Cadena et al. "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age". In: *Trans. Rob.* 32.6 (2016), pp. 1309–1332.

[3] Matteo Scucchia and Davide Maltoni. "Region Prediction for Efficient Robot Localization on Large Maps". In: *Robotics, Computer Vision and Intelligent Systems*. 2024.

[4] Matteo Scucchia and Davide Maltoni. "Continual Learning of Regions for Efficient Robot Localization on Large Maps". In: *IEEE Transactions on Robotics* (2025), pp. 1–20.

[5] Mathieu Labbé and François Michaud. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation". In: *Journal of Field Robotics* 36.2 (2018), pp. 416–446.

[6] Giovanni Emanuele Corazza, ed. *The Cyber-Creativity Process. How Humans Co-Create with Artificial Intelligence*. The Seven C's of Creativity. Palgrave Macmillan Cham, 2025.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[8] K. Simonyan and A. Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *International Conference on Learning Representations (ICLR)*. 2015.

[9] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[10] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1724–1734.

[11] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[12] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *North American Chapter of the Association for Computational Linguistics*. 2019.

[13] Alec Radford et al. *Improving Language Understanding by Generative Pre-Training*. Tech. rep. OpenAI, 2018.

[14] Kai Han et al. “A Survey on Vision Transformer”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.1 (2023), pp. 87–110.

[15] Liyuan Wang et al. “A Comprehensive Survey of Continual Learning: Theory, Method and Application”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 46.8 (2024), pp. 5362–5383.

[16] Davide Maltoni and Vincenzo Lomonaco. “Continuous learning in single-incremental-task scenarios”. In: *Neural Networks* 116 (2019), pp. 56–73.

[17] David Rolnick et al. “Experience Replay for Continual Learning”. In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[18] David Isele and Akansel Cosgun. “Selective Experience Replay for Lifelong Learning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (2018).

[19] James Kirkpatrick et al. “Overcoming catastrophic forgetting in neural networks”. In: *Proceedings of the National Academy of Sciences* 114 (2016), pp. 3521–3526.

[20] Andrei A. Rusu et al. “Progressive Neural Networks”. In: *ArXiv* (2016).

[21] Nathan Kirchner et al. “RobotAssist – A Platform for Human Robot Interaction Research”. In: *Proceedings of the Australasian Conference on Robotics and Automation (ACRA 2010)*. 2010.

[22] T. Bailey and H. Durrant-Whyte. “Simultaneous localization and mapping (SLAM): part II”. In: *IEEE Robotics Automation Magazine* 13.3 (2006), pp. 108–117.

[23] Randall C. Smith and Peter Cheeseman. "On the Representation and Estimation of Spatial Uncertainty". In: *The International Journal of Robotics Research* (1986).

[24] H.F. Durrant-Whyte. "Uncertain geometry in robotics". In: *IEEE Journal on Robotics and Automation* 4.1 (1988), pp. 23–31.

[25] Arthur Gelb and The Analytic Sciences Corporation. *Applied Optimal Estimation*. The MIT Press, 1974.

[26] R. E. Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45.

[27] B.M. Bell and F.W. Cathey. "The iterated Kalman filter update as a Gauss-Newton method". In: *IEEE Transactions on Automatic Control* 38.2 (1993), pp. 294–297.

[28] Dimitrios G. Kottas et al. "On the Consistency of Vision-Aided Inertial Navigation". In: *Experimental Robotics: The 13th International Symposium on Experimental Robotics*. Ed. by Jaydev P. Desai et al. Springer International Publishing, 2013.

[29] E.A. Wan and R. Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*. 2000, pp. 153–158.

[30] Michael Montemerlo et al. "FastSLAM: a factored solution to the simultaneous localization and mapping problem". In: *Eighteenth National Conference on Artificial Intelligence*. 2002, pp. 593–598.

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

[32] F. Dellaert et al. "Monte Carlo localization for mobile robots". In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. Vol. 2. 1999, 1322–1328 vol.2.

[33] Anastasios I. Mourikis and Stergios I. Roumeliotis. "A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. 2007, pp. 3565–3572.

[34] F. Lu and E. Milios. "Globally Consistent Range Scan Alignment for Environment Mapping". In: *Auton. Robots* 4.4 (1997), pp. 333–349.

[35] Giorgio Grisetti et al. “A Tutorial on Graph-Based SLAM”. In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43.

[36] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. “Real-time monocular SLAM: Why filter?” In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 2657–2664.

[37] Margarita N. Favorskaya. “Deep Learning for Visual SLAM: The State-of-the-Art and Future Trends”. In: *Electronics* (2023).

[38] Konstantinos A. Tsintotas, Loukas Bampis, and Antonios Gasteratos. “The Revisiting Problem in Simultaneous Localization and Mapping: A Survey on Visual Loop Closure Detection”. In: *IEEE Transactions on Intelligent Transportation Systems* 23.11 (2022), pp. 19929–19953.

[39] Saba Arshad and Gon-Woo Kim. “Role of Deep Learning in Loop Closure Detection for Visual and Lidar SLAM: A Survey”. In: *Sensors* (2021).

[40] Sen Wang et al. “DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2043–2050.

[41] Sen Wang et al. “DeepVO: Towards end-to-end visual odometry with deep Recurrent Convolutional Neural Networks”. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017, pp. 2043–2050.

[42] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[43] Edgar Sucar et al. “iMAP: Implicit Mapping and Positioning in Real-Time”. In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2021, pp. 6209–6218.

[44] Zihan Zhu et al. “NICE-SLAM: Neural Implicit Scalable Encoding for SLAM”. In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 12776–12786.

[45] Friedrich Fraundorfer and Davide Scaramuzza. “Visual Odometry : Part II: Matching, Robustness, Optimization, and Applications”. In: *IEEE Robotics Automation Magazine* 19.2 (2012), pp. 78–90.

[46] Davide Scaramuzza and Friedrich Fraundorfer. “Visual Odometry [Tutorial]”. In: *IEEE Robotics Automation Magazine* 18.4 (2011), pp. 80–92.

[47] S. B. Nashed. “A Brief Survey of Loop Closure Detection: A Case for Rethinking Evaluation of Intelligent Systems”. In: *NeurIPS 2020 Workshop: ML Retrospectives* (2020).

[48] Sherif A. S. Mohamed et al. "A Survey on Odometry for Autonomous Navigation Systems". In: *IEEE Access* 7 (2019), pp. 97466–97486.

[49] Alex Kendall, Matthew Grimes, and Roberto Cipolla. "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization". In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2938–2946.

[50] Clement Godard et al. "Digging Into Self-Supervised Monocular Depth Estimation". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 3827–3837.

[51] Yifan Jin et al. "A Mono SLAM Method Based on Depth Estimation by DenseNet-CNN". In: *IEEE Sensors Journal* 22.3 (2022), pp. 2447–2455.

[52] Paul-Edouard Sarlin et al. "SuperGlue: Learning Feature Matching With Graph Neural Networks". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020, pp. 4937–4946.

[53] Kuiyuan Fu et al. "An Effective End-to-End Image Matching Network with Attentional Graph Neural Networks". In: *2022 IEEE 17th Conference on Industrial Electronics and Applications (ICIEA)*. 2022, pp. 1628–1633.

[54] Zachary Teed and Jia Deng. "DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras". In: *Advances in Neural Information Processing Systems*. Vol. 34. 2021.

[55] Zachary Teed, Lahav Lipson, and Jia Deng. "Deep patch visual odometry". In: *Proceedings of the 37th International Conference on Neural Information Processing Systems*. 2023.

[56] Qing Li et al. "LO-Net: Deep Real-Time Lidar Odometry". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019, pp. 8465–8474.

[57] Tianyi Liu et al. "LiDAR Odometry by Deep Learning-Based Feature Points with Two-Step Pose Estimation". In: *Remote Sensing* 14.12 (2022).

[58] Sk Aziz Ali et al. "DELO: Deep Evidential LiDAR Odometry using Partial Optimal Transport". In: *2023 IEEE/CVF International Conference on Computer Vision Workshops (ICCVW)*. 2023, pp. 4519–4528.

[59] Sourav Garg, Tobias Fischer, and Michael Milford. "Where Is Your Place, Visual Place Recognition?" In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence (IJCAI-21)*. Ed. by Zhi-Hua Zhou. International Joint Conferences on Artificial Intelligence Organization, 2021, pp. 4416–4425.

[60] Carlo Masone and Barbara Caputo. "A Survey on Deep Visual Place Recognition". In: *IEEE Access* 9 (2021), pp. 19516–19547.

[61] Stephanie Lowry et al. "Visual Place Recognition: A Survey". In: *IEEE Transactions on Robotics* 32.1 (2016), pp. 1–19.

[62] David G Lowe. "Object recognition from local scale-invariant features". In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. 1999, pp. 1150–1157.

[63] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. "SURF: speeded up robust features". In: *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*. 2006, pp. 404–417.

[64] Ethan Rublee et al. "ORB: An efficient alternative to SIFT or SURF". In: *2011 International Conference on Computer Vision*. 2011, pp. 2564–2571.

[65] Zetao Chen et al. "Convolutional Neural Network-based Place Recognition". In: *ArXiv* (2014).

[66] Relja Arandjelovic et al. "NetVLAD: CNN Architecture for Weakly Supervised Place Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 5297–5307.

[67] Gaoshuang Huang et al. "DINO-Mix: Enhancing Visual Place Recognition with Foundational Vision Model and Feature Mixing". In: *Scientific Reports* 14 (2024), p. 10071.

[68] Maxime Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2024.

[69] Hanjie Liu et al. "Visual SLAM Based on Dynamic Object Removal". In: *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. 2019, pp. 596–601.

[70] Qingliang Luo et al. "Semantic-based Dynamic Object Separation Algorithm for Visual SLAM in Dynamic Environment". In: *2022 IEEE 17th Conference on Industrial Electronics and Applications (ICIEA)*. 2022, pp. 937–942.

[71] Phani K. Uppala, Abhishek Bamotra, and Raj Kolamuri. *Dynamic Object Removal for Effective SLAM*. arXiv preprint arXiv:2303.10923. 2023. URL: https://arxiv.org/abs/2303.10923.

[72] Jiafeng Cui and Xieyuanli Chen. "CCL: Continual Contrastive Learning for LiDAR Place Recognition". In: *IEEE Robotics and Automation Letters* 8.8 (2023), pp. 4433–4440.

[73] Dongshuo Zhang et al. "CurricularVPR: Curricular Contrastive Loss for Visual Place Recognition". In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2024, pp. 7916–7921.

[74] Milad Ramezani et al. "Pose-Graph Attentional Graph Neural Network for Lidar Place Recognition". In: *IEEE Robotics and Automation Letters* 9.2 (2024), pp. 1182–1189.

[75] Friedrich Fraundorfer et al. "Visual Word Based Location Recognition in 3D Models Using Distance Augmented Weighting". In: *Proceedings of the Fourth International Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT)*. IEEE, 2008, pp. 331–338.

[76] Konstantinos A. Tsintotas et al. "SeqSLAM with Bag of Visual Words for Appearance Based Loop Closure Detection". In: *Advances in Service and Industrial Robotics*. 2019.

[77] Carlos Campos et al. "ORB-SLAM3: An Accurate Open-Source Library for Visual, Visual–Inertial, and Multimap SLAM". In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 1874–1890.

[78] Adrien Angeli et al. "Fast and Incremental Method for Loop-Closure Detection Using Bags of Visual Words". In: *IEEE Transactions on Robotics* 24.5 (2008), pp. 1027–1037.

[79] Tudor Nicosevici and Rafael Garcia. "On-line visual vocabularies for robot navigation and mapping". In: *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2009, pp. 205–212.

[80] Yogesh Girdhar and Gregory Dudek. "Online Visual Vocabularies". In: *2011 Canadian Conference on Computer and Robot Vision*. 2011, pp. 191–196.

[81] Aram Kawewong et al. "Online and Incremental Appearance-based SLAM in Highly Dynamic Environments". In: *I. J. Robotic Res.* 30 (2011), pp. 33–55.

[82] Tudor Nicosevici and Rafael Garcia. "Automatic Visual Bag-of-Words for Online Robot Navigation and Mapping". In: *IEEE Transactions on Robotics* 28.4 (2012), pp. 886–898.

[83] Mathieu Labbé and François Michaud. "Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation". In: *IEEE Transactions on Robotics* 29.3 (2013), pp. 734–745.

[84] Ahmad Khaliq et al. "A Holistic Visual Place Recognition Approach Using Lightweight CNNs for Significant ViewPoint and Appearance Changes". In: *IEEE Transactions on Robotics* 36.2 (2020), pp. 561–569.

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

[86] Pengshuai Hou et al. "Forest: A Lightweight Semantic Image Descriptor for Robust Visual Place Recognition". In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 12531–12538.

[87] Burak Kaleci, Osman Parlaktuna, and Uğur Gürel. "A comparative study for topological map construction methods from metric map (In English)". In: *2018 26th Signal Processing and Communications Applications Conference (SIU)*. 2018, pp. 1–4.

[88] Xiang Gao et al. *14 Lectures on Visual SLAM: From Theory to Practice*. 2017.

[89] Bernhard Kerbl et al. "3D Gaussian Splatting for Real-Time Radiance Field Rendering". In: *ACM Transactions on Graphics (TOG)* 42 (2023), pp. 1–14.

[90] Hidenobu Matsuki et al. "Gaussian Splatting SLAM". In: *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024, pp. 18039–18048.

[91] Long Wen et al. "Gassidy: Gaussian Splatting SLAM in Dynamic Environments". In: *2025 IEEE International Conference on Robotics and Automation (ICRA)*. 2025.

[92] Zunjie Zhu et al. *Robust Gaussian Splatting SLAM by Leveraging Loop Closure*. arXiv preprint arXiv:2409.20111. 2024.

[93] Liyuan Zhu et al. "LoopSplat: Loop Closure by Registering 3D Gaussian Splats". In: *2025 International Conference on 3D Vision (3DV)*. 2025.

[94] Simon F. G. Ehlers et al. "Map Management Approach for SLAM in Large-Scale Indoor and Outdoor Areas". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 9652–9658.

[95] Peng Yin et al. "BioSLAM: A Bioinspired Lifelong Memory System for General Place Recognition". In: *IEEE Transactions on Robotics* 39.6 (2023), pp. 4855–4874.

[96] Christina Kassab et al. *Language-EXtended Indoor SLAM (LEXIS): A Versatile System for Real-time Visual Scene Understanding*. 2024.

[97] Konstantinos A. Tsintotas et al. "Sequence-based mapping for probabilistic visual loop-closure detection". In: *2021 IEEE International Conference on Imaging Systems and Techniques (IST)*. 2021.

[98] Mathieu Labbé and François Michaud. "Memory management for real-time appearance-based loop closure detection". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2011, pp. 1271–1276.

[99] Joseph Rowell, Lintong Zhang, and Maurice Fallon. "LiSTA: Geometric Object-Based Change Detection in Cluttered Environments". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. 2024, pp. 3632–3638.

[100] Lukas Schmid et al. "Khronos: A Unified Approach for Spatio-Temporal Metric-Semantic SLAM in Dynamic Environments". In: *Robotics: Science and Systems (RSS)*. Berkeley, CA, USA, 2024.

[101] João Carlos Virgolino Soares et al. "Visual Localization and Mapping in Dynamic and Changing Environments". In: *Journal of Intelligent & Robotic Systems* 109.4 (2023), p. 95.

[102] Jingxing Qian et al. *POV-SLAM: Probabilistic Object-Aware Variational SLAM in Semi-Static Environments*. 2023.

[103] Niclas Vödisch et al. "Continual SLAM: Beyond Lifelong Simultaneous Localization and Mapping Through Continual Learning". In: *Robotics Research*. Springer Nature Switzerland, 2023, pp. 19–35.

[104] Dasong Gao, Chen Wang, and Sebastian Scherer. "AirLoop: Lifelong Loop Closure Detection". In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 10664–10671.

[105] Rahaf Aljundi et al. "Gradient based sample selection for online continual learning". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[106] Niclas Vödisch et al. "CoVIO: Online Continual Learning for Visual-Inertial Odometry". In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2023, pp. 2464–2473.

[107] Baicheng Li et al. “Learn to Memorize and to Forget: A Continual Learning Perspective of Dynamic SLAM”. In: *Computer Vision – ECCV 2024*. 2024, pp. 41–57.

[108] Winston Churchill and Paul Newman. “Experience-based navigation for long-term localisation”. In: *The International Journal of Robotics Research* 32.14 (2013), pp. 1645–1661. DOI: 10.1177/0278364913499193.

[109] Technical University of Munich Computer Vision Group TUM Department of Informatics. *Useful tools for the RGB-D benchmark*. URL: https://cvg.cit.tum.de/data/datasets/rgbd-dataset/tools.

[110] Jürgen Sturm et al. “A benchmark for the evaluation of RGB-D SLAM systems”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2012, pp. 573–580.

[111] Berthold Horn. “Closed-form solution of absolute orientation using unit quaternions”. In: *Journal of the Optical Society of America A* (1987).

[112] Xuesong Shi et al. “Are We Ready for Service Robots? The OpenLORIS-Scene Datasets for Lifelong SLAM”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2020.

[113] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[114] D. Maio, D. Maltoni, and S. Rizzi. “Dynamic clustering of maps in autonomous agents”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 18.11 (1996), pp. 1080–1091.

[115] Wei Zhou et al. *The USyd Campus Dataset*. 2019. URL: https://dx.doi.org/10.21227/sk74-7419.

[116] Bolei Zhou et al. “Places: A 10 Million Image Database for Scene Recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 40.6 (2018), pp. 1452–1464.

[117] Mark Sandler et al. “ MobileNetV2: Inverted Residuals and Linear Bottlenecks ”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[118] Tsung-Yi Lin et al. “Focal Loss for Dense Object Detection”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42.2 (2020), pp. 318–327.

[119] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[120] Xaxxon. *Xaxxon AutoCRAWLER Autonomous Robot*. URL: `http://www.xaxxon.com/xaxxon/autocrawler`.

[121] Arren J. Glover et al. "FAB-MAP + RatSLAM: Appearance-based SLAM for multiple times of day". In: *2010 IEEE International Conference on Robotics and Automation*. 2010.

[122] David Olid, Javier M. Fácil, and Javier Civera. "Single-View Place Recognition under Seasonal Changes". In: *CoRR* (2018).

[123] Kai Ming Ting. "Precision and Recall". In: *Encyclopedia of Machine Learning*. Springer Publishing Company, Incorporated, 2010, pp. 781–781.

[124] Lorenzo Pellegrini et al. "Latent Replay for Real-Time Continual Learning". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020, pp. 10203–10209.

[125] C. T. Fan, Mervin E. Muller, and Ivan Rezucha. "Development of Sampling Plans by Using Sequential (Item by Item) Selection Techniques and Digital Computers". In: *Journal of the American Statistical Association* 57.298 (1962), pp. 387–402.

[126] Vincenzo Lomonaco et al. "Avalanche: an End-to-End Library for Continual Learning". In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2021.

[127] 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.

[128] Shaharyar Ahmed Khan Tareen and Zahra Saleem. "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK". In: *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. 2018.

[129] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. "BAD SLAM: Bundle Adjusted Direct RGB-D SLAM". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[130] Rockstar Games. *GTA V*. 2013.

[131] Braden Hurl, Krzysztof Czarnecki, and Steven Waslander. "Precise Synthetic Image and LiDAR (PreSIL) Dataset for Autonomous Vehicle Perception". In: *2019 IEEE Intelligent Vehicles Symposium (IV)*. 2019.

[132] Stephan R. Richter et al. "Playing for Data: Ground Truth from Computer Games". In: *European Conference on Computer Vision (ECCV), Part II*. Vol. 9906. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 102–118.

[133] Divya Srivastava et al. "Content-Based Image Retrieval: A Survey on Local and Global Features Selection, Extraction, Representation, and Evaluation Parameters". In: *IEEE Access* (2023).

[134] Akihiko Torii et al. "24/7 place recognition by view synthesis". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[135] Michael J. Milford and Gordon F. Wyeth. "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights". In: *2012 IEEE International Conference on Robotics and Automation*. 2012.

[136] Akihiko Torii et al. "Visual Place Recognition with Repetitive Structures". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013.

[137] Frederik Warburg et al. "Mapillary Street-Level Sequences: A Dataset for Lifelong Place Recognition". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[138] Giovanni Curnis, Simone Fontana, and Domenico G. Sorrenti. "GTASynth: 3D synthetic data of outdoor non-urban environments". In: *Data in Brief* (2022).

[139] Jürgen Sturm et al. "A Benchmark for the Evaluation of RGB-D SLAM Systems". In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. 2012.

[140] Michael Burri et al. "The EuRoC micro aerial vehicle datasets". In: *The International Journal of Robotics Research* (2016).

[141] Milad Ramezani et al. "The Newer College Dataset: Handheld LiDAR, Inertial and Vision with Ground Truth". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2020.

[142] Nicholas Carlevaris-Bianco, Arash K. Ushani, and Ryan M. Eustice. "University of Michigan North Campus long-term vision and lidar dataset". In: *International Journal of Robotics Research* (2015).

[143] Yohann Cabon, Naila Murray, and Martin Humenberger. *Virtual KITTI 2*. 2020.

[144] Ankur Handa et al. "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM". In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014.

[145] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. "Robust Reconstruction of Indoor Scenes". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[146] Aakash Rajpal et al. "High-Resolution Synthetic RGB-D Datasets for Monocular Depth Estimation". In: *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. 2023.

[147] Benjamin Kiefer, David Ott, and Andreas Zell. "Leveraging Synthetic Data in Object Detection on Unmanned Aerial Vehicles". In: *2022 26th International Conference on Pattern Recognition (ICPR)*. IEEE Computer Society, 2022, pp. 3564–3571.

[148] Zhe Cao et al. "Long-term human motion prediction with scene context". In: *Computer Vision – ECCV 2020*. Springer International Publishing, 2020.

[149] Alexander Blade. *ScriptHook V*. URL: `https://www.dev-c.com/gtav/scripthookv/`.

[150] Anh-Dzung Doan et al. "G2D: from GTA to Data". In: *arXiv preprint arXiv:1806.07381* (2018).

[151] David Ott. *DeepGTAV*. URL: `https://github.com/David0tt/DeepGTAV`.

[152] Clément Godard et al. "Digging Into Self-Supervised Monocular Depth Estimation". In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[153] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[154] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009.

[155] Amey Kasar. "Benchmarking and Comparing Popular Visual SLAM Algorithms". In: *Asian Journal for Convergence in Technology (AJCT)* (2019).