A.D. 1308
unipg
DIPARTIMENTO
DI INGEGNERIA

University of Perugia
Department of Engineering

Ph.D. Dissertation
Industrial and Information Engineering

2023

Vision-based Robot Localization: from model-based and data-driven strategies to topological approaches

Simone Felicioni

Supervisor
Prof. Gabriele Costante

Ph.D. Programme Director
Prof. Giuseppe Liotta

Dedicated to my family,
for your constant support
and unconditional love.

Abstract

Localization is the core functionality to enable autonomous navigation capabilities in robotic platforms. In GPS-denied environments, *e.g.*, indoor, underground, or underwater scenarios, robots need to exploit onboard sensors for ego-motion estimation. Among these, vision-based localization has gained considerable attention in recent years within the research community. In particular, it can be categorized into metric and semantic approaches, depending on the pose representation strategy employed. Despite the impressive results obtained by the former, they suffer from a quick degradation of their performance under non-ideal conditions (*e.g.*, scenes with poor illumination and low amount of textures). Contrarily, the latter solutions are recently emerging as a promising alternative in autonomous navigation systems, that trade off precise metric positioning with a more robust and higher-level location representation. State-of-the-art works in this direction, however, often neglect the spatio-temporal relationships between poses that are naturally induced by robotic navigation. Furthermore, these techniques are nearly unexplored for autonomous flying platforms. Inspired by these considerations, this Thesis aims to investigate novel solutions in the vision-based robot localization field.

Contents

1 Introduction **11**

1.1 Motivation 11

1.2 Thesis Outline 15

2 Related work **16**

2.1 Metric Approaches for Localization 16

2.1.1 Geometric paradigms 17

2.1.2 Data-driven paradigms 20

2.1.3 Benchmarks 22

2.1.4 Datasets 24

2.2 Semantic Approaches for Localization 26

2.2.1 Visual Place Recognition 26

2.2.2 Topological Localization 30

3 Visual Odometry **34**

3.1 Overview 34

3.2 First experimental campaign 35

3.2.1 Datasets 35

3.2.2 Implementation details 39

3.2.3 Experimental setup 39

3.2.4 Results and discussion 44
3.3 Second experimental campaign 54
3.3.1 Datasets 55
3.3.2 Implementation details 57
3.3.3 Experimental setup 58
3.3.4 Results and discussion 61

4 Topological Localization **65**
4.1 Overview 65
4.2 Preliminaries 67
4.2.1 Notation 68
4.2.2 Graph Neural Network 69
4.2.3 Graph Convolutional Neural Network 71
4.3 Ground platforms 72
4.3.1 Contributions 73
4.3.2 Proposed approach 73
4.3.2.1 Map building 75
4.3.2.2 Localization procedure 76
4.3.2.3 Implementation details 80
4.3.3 Experimental results 83
4.3.4 Discussion and limitations 88
4.4 Aerial platforms 88
4.4.1 Contributions 89
4.4.2 Proposed approach 90
4.4.2.1 Map building 91
4.4.2.2 Localization procedure 93
4.4.2.3 Implementation details 98
4.4.3 Experimental results 101

4.4.4 Discussion and limitations 108

5 Conclusions **110**

References **112**

A Scientific Publications **131**

List of Figures

1.1 Among the main components of an autonomous platform, *e.g.*, mapping, path planning, or obstacle avoidance, estimating the robot's position is certainly the core one. 12

1.2 Examples of ego-motion estimations in challenging scenarios. In non-ideal conditions, *e.g.*, fast motion dynamics (1.2a) or texture-poor environments (1.2b), the performance of VO algorithms tend to degrade significantly. 13

1.3 Overview of a topological localization solution. A topological map describes the most salient location in the environment. A robotic platform navigates through the environment and, for each observation, estimates the closest graph node to the current position. 14

3.1 Example images from the Unreal dataset. The top row contains frames from the outdoor sequences, while the bottom one depicts examples from the indoor scenarios. 37

3.2 Example images from the Marzaglia dataset. 38

3.3 Summary charts for the EuRoC dataset. The errors reported are obtained with the *default* hyper-parameter configuration. The chart on the left shows the results with respect to the $RMSE_{ATE}$ (in meters), while the one on the right with respect to the $RMSE_{ARE}$ (in degrees). 46
3.4 Memory usage of the compared approaches on the EuRoC dataset with the *default* configuration. 46
3.5 Total execution time of the compared approaches on the EuRoC dataset with the *default* configuration. 47
3.6 Execution Time analysis for the different functional blocks of the geometric methods on EuRoC dataset. The results are obtained by using the *default* hyper-parameter configuration. 47
3.7 Performance analysis for various hyper-parameters configurations on the EuRoC dataset (Workstation). The charts report the results for DSO and VINS-Mono. 48
3.8 Trajectory plot for Monodepth2, Open-VINS Monocular and ORB-SLAM3 Stereo-Inertial on the sequence *MH-05 difficult* of the EuRoC dataset. When ideal conditions are present, *e.g.*, texture-rich environments, both VO and SLAM geometric approaches outperform by far the data-driven ones. In the Figures, the black dashed line represents the reference trajectory (*i.e.*, the ground-truth poses), while the colored line represents the trajectory estimation. In particular, the color depends on the RMSE (ATE) of that specific pose. 50
3.9 Summary chart for the Marzaglia dataset. The errors reported are obtained with the *default* hyper-parameter configuration. The chart shows the results with respect to the $RMSE_{ATE}$ (in meters). 54

3.10 Trajectory plot for ORB-SLAM3 Stereo-Inertial, DSO and Monodepth2 on the sequence *lap02* of the Marzaglia dataset. It can be observed that the pose estimation performance of OS3-SI is very poor. Among the geometric approaches, DSO is the only one estimating a good trajectory. On the other hand, the trajectory computed by MD2 is accurate and similar to the reference one. In the Figures, the black dashed line represents the reference trajectory (*i.e.*, the ground-truth poses), while the colored line represents the trajectory estimation. In particular, the color depends on the RMSE (ATE) of that specific pose. 55
3.11 Example images from the dataset collected in olive cultivations. . 56
3.12 Example images from the dataset collected in olive cultivations. . 58
3.13 Trajectory estimation on Vynrd B (2021-09-01-12-25-09). Some approaches may lose tracking or fail before the end of the trajectory. In this cases, the execution of the algorithm is stopped and estimated poses are no longer provided. In these cases, the corresponding plots show shorter trajectories since only the estimated poses (and the associated ground truth ones) before the algorithm failure are depicted, discarding the the remaining ground truth trajectory points. 64

4.1 Overview of the topological localization training process. First, a graph is built for each environment. Afterward, the robot navigates through the environment to gather the training data, *i.e.*, visual observations associated with the ground-truth node. Finally, the model is trained to estimate the closest graph node to the current robot position. 66
4.2 2D convolution vs graph convolution 71

4.3 GOLN overview. Our framework takes in input a graph map (*top-left* image) and the current observation captured by the robot front-view camera (*top-right* image). In addition, it considers the objects in the surroundings (red squared). It returns the estimation of the closest position graph node to the current robot location. 74

4.4 Architecture of the Graph Network, which updates the input features: the graph (*i.e.*, the node features and the edges) and the global features (*i.e.*, the scene features and the objects' features). The overall network consists of two Node Models and one Global Model. A classification layer estimates the current position node of the robot. 77

4.5 Inference process for the proposed topological localization approach. A pre-processing step is employed to build a graph representation of the environment. Then, a Graph Neural Network (GNN) processes the pre-built topological map and the current observation to predict the closest graph node. 82

4.6 Example of indoor environments generated for the training, validation and test sets. The first row shows the *top-view* of the rooms, while the second row the robot *front-view* camera. 84

4.7 Test-scenario rooms employed for online localization experiments. The first row comprises the *top-view* of the environments, while the second one the robot *front-view* camera. 86

4.8 Overview of the approach. The proposed strategy processes the images collected by the MAV and predicts the closest node in the topological graph that represents the environment. 89

4.9 3D topological graph building process. 4.9a and 4.9b represent the voxel grid and the *skeletonization* of the environment, respectively. The following steps 4.9c-4.9d create the nodes and merge the closest ones to induce sparsity on the graph. A specific node color represents the type node according to its number of neighbors, *i.e.*, end-point (in light blue), corner (in green), or branch (in red). 92

4.10 Overview of the proposed architecture. The feature extraction module combines the outputs of different convolutional stages of ResNet-50 and processes it with PCA to obtain a 512-dimensional vector. This vector is combined with the features of the neighbor nodes by the spatial-correlation module. The temporal-correlation module, instead, propagates information from past graphs through time. Finally, the node scores computed by the FC layer are refined by a geometric consistency check that re-ranks the top k node candidates. 94

4.11 Analysis of the computational time as a function of the number of nodes in the graph. 100

4.12 Examples of the evaluation environments: **Tr-T** set (4.12a); **Tr-A** set (4.12b); **Test** set (4.12c); **Test-Ph** set (4.12d). 102

4.13 Comparison of the R@N metric on the Test-Ph environment. . . . 106

List of Tables

3.1 Workstation quantitative results on the test sequences of the EuRoC dataset. 51

3.2 Workstation quantitative results on the sequences of the Marzaglia dataset. 53

3.3 Vineyards employed for the dataset data collection 56

3.4 Production characteristics and indices of the vegetative-productive balance of the vineyards grouped with respect to different NDVI zones, *i.e.*, low vigor (NDVI < 0.83), medium vigor ($0.84 <$ NDVI < 0.90), and high vigor (NDVI > 0.91). The values are computed by manual inspection in different zones of the crop. 57

3.5 Olive cultivations employed for the dataset data collection . . . 58

3.6 Growth stage of the olive trees based on BBCH scale. Values around $80 - 81$ indicate the beginning of fruit coloring, 85 shows the increase of specific coloring, while 89 denotes the best stage for harvesting (*i.e.*, suitable for oil extraction, when fruits reach the typical color while still remaining turgid). 59

3.7 Quantitative results on the selected sequences. The metrics are detailed in Section 3.3.3. Mean and standard deviation over 5 runs are reported, and the best results for each metric among vision-based and LiDAR-based approaches are highlighted in bold. If two approaches achieve similar average errors, to select the best one we consider that with the smaller standard deviation. 63

4.1 Performance comparison in online localization experiments between GOLN and monocular vision-only ORB-SLAM3. 85

4.2 Accuracy performance of GOLN framework in several environments, either similar to the training set or totally different. 87

4.3 The Table reports mean and standard deviation for MNA and OHA computed on the environments of each evaluation set. 104

4.4 Ablation study on the main components of the proposed approach. Check marks (✓) indicate if a particular component has been enabled. For both the metrics, we report the mean and the standard deviation computed on the environments of each evaluation set. . 107

Chapter 1

Introduction

1.1 Motivation

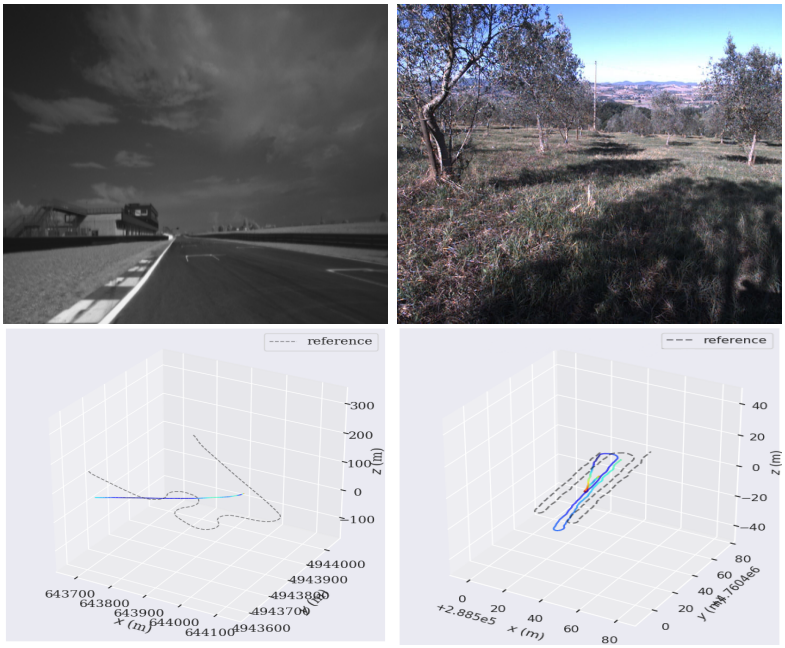
Localization is one of the most crucial components of autonomous platforms as the knowledge of the robot's position is essential to making decisions about future actions [1]. Specifically, robot localization is the process of determining where a mobile robot is located with respect to the environment. When external sources of information, such as a Global Positioning System (GPS), are not available (*e.g.*, in indoor, underground, or underwater scenarios), the robot may exploit different onboard sensors for ego-motion estimation [2; 3].

Vision-based localization, in particular, has attracted the interest of many research studies in the last few years [4; 5; 6]. The most popular solutions proposed for this problem are grounded on the Visual Odometry (VO) and the Simultaneous Localization and Mapping (SLAM) frameworks [7]. VO is the process of estimating the ego-motion of an agent using only visual data (*i.e.*, single or multiple cameras). The pioneering work that devised some fundamental concepts behind vision-based motion estimation dates back to the early 1980s [8], while the term VO was coined in 2004 by [9] to outline that, similarly to wheel odom-

Figure 1.1: Among the main components of an autonomous platform, *e.g.*, mapping, path planning, or obstacle avoidance, estimating the robot's position is certainly the core one.

etry, it incrementally estimates the pose of the vehicle by exploiting the changes that motion induces on the images captured by the onboard cameras.

From a methodological perspective, the VO approaches can be categorized into geometric or data-driven. The former paradigm exploits well-known geometrical relations between image pairs or sequences. These techniques can be further categorized based on the visual cues used for estimation (such as relying on sparse keypoints [10; 11] or leveraging information from all pixels [12; 13]), and on the objective function optimized, *i.e.*, the photometric error (direct methods) [14; 15] or the geometric error (indirect methods) [10; 11]. In addition, Visual Inertial Odometry (VIO) methods have shown remarkable results by combining vision sensors with Inertial Measurement Units (IMUs) [16; 17; 18]. In contrast, data-driven solutions have recently emerged as a viable and more robust alternative to geometric ones. Indeed, they take advantage of the representational power



(a) (b)

Figure 1.2: Examples of ego-motion estimations in challenging scenarios. In non-ideal conditions, *e.g.*, fast motion dynamics (1.2a) or texture-poor environments (1.2b), the performance of VO algorithms tend to degrade significantly.

of Convolutional Neural Networks (CNNs), which has been proven to achieve robustness to non-ideal image conditions in extracting visual features [19; 20].

Despite the impressive results achieved, estimating precise metric transformations between frames is often hindered by many factors. The performance of VO and SLAM approaches tend to degrade under non-ideal conditions, *i.e.*, texture-poor scenes, low amount of illumination, and fast motion dynamics, as shown in Figure 1.2. Most of the challenges arise from the requirement of precise metric positioning, which, in many applications, might be an unnecessary over-complication. If we shift the paradigm and take inspiration from humans and animals, we notice that they do not rely on metric positioning systems. Instead, they take advantage of semantic information and high-level relationships between objects, scenes, and shapes to estimate their position in the environment

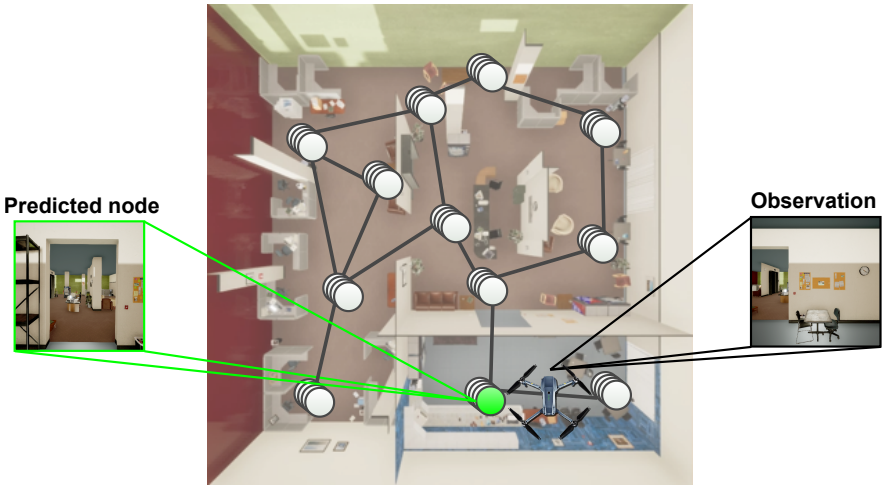


Figure 1.3: Overview of a topological localization solution. A topological map describes the most salient location in the environment. A robotic platform navigates through the environment and, for each observation, estimates the closest graph node to the current position.

[21]. Following these intuitions, visual place recognition and topological localization solutions have recently emerged as alternative paradigms to achieve robust localization even in non-ideal scenarios [22; 23; 24]. The concept of topological localization was pioneered by Benjamin Kuipers in the late seventies [25], and it aims to estimate the position of the robot with respect to a known map given an observation from the current point of view, as shown in Figure 1.3. The peculiarity of this paradigm lies behind the use of a topological map, *i.e.*, a sparse representation of the scene consisting of a set of nodes and a set of edges. In particular, a node represents a salient location in the space, while an edge is a connection between two positions due to, *e.g.*, mutual visibility or reachability. This type of map represents an interesting alternative to the classic metric maps due to its simplicity and storage needs, which might be not appropriate for tasks with high accuracy needs (*e.g.*, obstacle avoidance), but can simplify others (*e.g.*, path planning). At the same time, appearance-based localization can exploit the technological improvements in image processing developed in the last decades

thanks to the emergence of Deep Learning.

The mentioned advantages made topological localization a promising solution and, thus, an active research area within the Robotics and Computer Vision communities. Inspired by these considerations, this Thesis aims to investigate novel and more robust solutions in the vision-based robot localization field.

1.2 Thesis Outline

The Thesis is organized as follows:

Chapter 2 reports the related works in literature. Specifically, it presents the most recent VO paradigms and the comparative studies showing the advantages and disadvantages of the aforementioned approaches. Moreover, semantic approaches are presented as viable alternatives to metric positioning systems, and the differences among the solutions proposed in the literature are highlighted.

Chapter 3 contains two benchmark studies performed on some of the most recent and best-performing state-of-the-art VO/SLAM solutions in several scenarios and environments. The studies include an extensive set of experiments to show the limitations of the tested approaches from different perspectives.

Chapter 4 includes the main contributions of this work, suggesting to shift from a frame-to-frame transformation estimation to a higher-level localization, that is more similar to how humans and animals actually navigate through an environment. Specifically, it presents two novel topological localization approaches for ground and aerial vehicles.

Chapter 5 draws conclusions and shows the challenges and limitations of the proposed approaches, indicating future directions.

Chapter 2

Related work

2.1 Metric Approaches for Localization

Historically, geometric strategies based on sparse feature matching or tracking procedures were the first to be explored. A feature detector is first used to identify candidate points of interest in the initial image, namely features or keypoints, that are distinctive and can be reliably tracked over time (*e.g.*, areas with high contrast or edges). These points are then encoded by a feature descriptor, that provides a compact representation of the information surrounding the keypoint that is invariant to changes in scale, rotation, illumination, and other image transformations.

Nowadays, the VO methods can be categorized as direct or indirect approaches, as well as sparse or dense. Indirect methods first generate an intermediate representation by pre-processing the sensor measurements, then estimate the camera motion by optimizing a geometric error within a probabilistic framework. Differently, direct methods optimize a photometric error by directly employing the sensor values (*e.g.*, light received from a certain direction) for a probabilistic model, thus removing the pre-processing step.

For the second categorization, sparse methods focus on a set of points (*e.g.*, features), whereas dense methods aim to use and reconstruct all pixels in the image domain.

Data-driven approaches for VO have a more recent history than geometric ones. The pioneer works in this direction [26; 27] explored Support Vector Machines (SVMs), Gaussian Processes, and K-nearest neighbor to estimate the camera ego-motion. The authors of [20] were the first to use CNNs to compute more robust features and increase the ego-motion regression performance. The Deep Learning paradigm has been further exploited by LS-VO [19], DeepVO [28], and UA-VO [29].

All the aforementioned approaches are optimized in a supervised manner. However, acquiring reliable pose ground truth labels to supervise the learning process is very burdensome and requires a high-precision positioning system synchronized with the image streams, *e.g.*, a GPS-RTK coupled with a LiDAR technology for outdoor scenarios or a multi-camera tracking system in indoor contexts. For these reasons, unsupervised and self-supervised strategies have emerged as a viable and more practical alternative.

In the following, some of the most recent and best-performing geometric and data-driven methods are presented.

2.1.1 Geometric paradigms

DSO Direct Sparse Odometry (DSO) [14] is based on continuous optimization of the photometric error over a window of recent frames, taking into account a photometrically calibrated model for image formation. DSO samples pixels from the image regions that have intensity gradients, hence it does not depend on keypoint detectors or descriptors. A window of up to N frames is maintained, and every new frame is initially tracked with respect to these reference frames.

The latter is either discarded or used to create a new keyframe, and, once a new keyframe is created, the total photometric error is optimized. Afterward, the marginalization of one or more frames is performed.

Open-VINS (OV) The core of Open-VINS [18] is an Extended Kalman filter that fuses inertial information with sparse visual feature tracks. Visual features are detected with FAST [30], tracked with KLT [31], and finally fused by leveraging the Multi-State Constraint Kalman Filter (MSCKF) sliding window formulation, which updates the estimation of the state without directly estimating the feature states in the filter.

VINS-Mono (VM) VINS-Mono [17] is a robust and versatile monocular visual-inertial state estimator. An initialization procedure, namely a vision-only Structure from Motion (SfM), estimates a graph of up-to-scale camera poses and feature positions, followed by a visual-inertial alignment. Then, for each new image: (i) existing features are tracked by the KLT sparse optical flow algorithm [31]; (ii) Shi Tomasi corner features [32] are detected to maintain a minimum number (100–300) of features in each image, and (iii) keyframes are selected. VINS-Mono adopts a sliding window-based tightly coupled monocular VIO for state estimation. The method uses a visual-inertial bundle adjustment (BA) formulation that considers the set of all IMU measurements, and the set of features that have been observed at least twice in the current sliding window. Additionally, a loop-detection module that identifies places that have already been visited. Feature-level connections between loop closure candidates and the current frame are then established, employing DBoW2 [33] for loop detection. After relocalization, an additional pose graph optimization step is developed to ensure the set of past poses is registered into a globally consistent configuration.

ORB-SLAM ORB-SLAM [11] has three main parallel threads: (i) the tracking thread to localize the camera with every frame by finding feature matches to the local map, minimizing the reprojection error via motion-only BA and detecting new keyframes; (ii) the local mapping thread to manage the local map and optimize it, inserting and deleting keyframes and performing local BA; (iii) the loop closing thread to detect large loops and correct the accumulated drift by performing a pose-graph optimization. This thread launches a fourth thread to perform full BA after the pose-graph optimization, to compute the optimal structure and motion solution. The method is named after the ORB detector [34], whose extracted features are used for tracking, mapping and place recognition tasks.

ORB-SLAM2 [35] handles stereo and RGB-D cameras, including loop closing, relocalization, and map reuse. It has a feature-based method that pre-processes the input to extract features at salient keypoint locations. The input images are then discarded and all system operations are based on these features, so that the system is independent of the sensor being stereo or RGB-D.

In ORB-SLAM Visual Inertial (ORB-SLAM-VI) [36] the tracking optimizes the current frame assuming a fixed map and a backend performs local BA, optimizing a local window of keyframes, including an outer window of fixed keyframes. This approach allows constant time local BA, and by not marginalizing past states, it is possible to reuse them.

ORB-SLAM-3 (OS3) [37] is built on top of the previous works, and can perform visual, visual-inertial, and multi-map SLAM with monocular, stereo, and RGB-D cameras, using pin-hole and fisheye lens models. The novelties of ORB-SLAM3 with respect to previous methods are (i) a feature-based tightly-integrated visual-inertial SLAM system that fully relies on Maximum-a-Posteriori (MAP) estimation, even during the IMU initialization phase; (ii) ORB-SLAM At-

las, a multiple map system that relies on a new place recognition method with improved recall. It keeps active and non-active maps able to handle visual and visual-inertial systems, in monocular and stereo configurations; (iii) a camera model abstraction handle by a camera module, possible thanks to the adoption of Maximum Likelihood Perspective-n-Point algorithm (MLPnP) for relocalization and non rectified frames for stereo-SLAM; (iv) a novel place recognition algorithm, in which candidate keyframes are first checked for geometrical consistency, and then for local consistency with three covisible keyframes.

ROVIO ROVIO [16] is a fully robocentric and direct monocular visual-inertial odometry method. A limited number of visual features, extracted in each frame with a multilevel FAST corner detector and selected with a bucketing technique, are fully integrated into the state of the Kalman filter, along with the system state. Within the prediction step, the new locations of the multilevel patch features are estimated by considering the IMU-driven motion model. For every captured image, a state update is performed and the camera extrinsic as well as the additive IMU biases are co-estimated. Later, the method has been extended in [38] with an iterated extended Kalman filter, and a photometric error model.

2.1.2 Data-driven paradigms

SfMLearner (SFML) SfMLearner [39] is certainly among the first completely unsupervised works that jointly trains a single-view depth CNN and a camera pose estimation CNN from unlabeled video sequences. The key supervision signal for the depth and pose prediction CNNs comes from the task of novel view synthesis: given one input view of a scene, synthesize a new image of the scene seen from a different camera pose. The loss used to train the model is indeed a combination of view synthesis objective, a smoothness objective for the depth

maps, and a regularization term. The input to the pose estimation network is the target view concatenated with all the source views, and the outputs are the relative poses between the target view and each of the source views. The network consists of 7 stride-2 convolutions followed by a 1×1 convolution with $6 \times (\mathrm{N}-1)$ output channels (corresponding to 3 Euler angles and 3-D translation for each source view).

Competitive Collaboration (CC) The authors of [40] introduce Competitive Collaboration to address the problem of joint unsupervised learning. CC is a generic framework in which networks learn to collaborate and compete, thereby achieving specific goals. It is a three-player game consisting of two players competing for a resource that is regulated by a third player, a moderator. For the depth network, the authors experimented with DispNetS and DispResNet where they replaced convolutional blocks with residual blocks. For the flow network, FlowNetC and PWC-Net are considered. The best results were obtained by combining DispResNet and PWC-Net.

Monodepth2 (MD2) The authors of [41] introduce a new depth and pose NN called Monodepth2, with three new contributions. First, they introduce a minimum reprojection loss, designed to robustly handle occlusions. Secondly, they use a full-resolution multi-scale sampling method that reduces visual artifacts. Finally, they include an auto-masking loss to ignore training pixels that violate camera motion assumptions, filtering out pixels that do not change appearance from one frame to the next in the sequence. The depth estimation network is based on the general U-Net architecture, *i.e.*, an encoder-decoder network, with skip connections. The pose network is formed from a ResNet18, modified to accept only two-color images (or six channels) as input and to predict a single 6-DoF relative pose, *i.e.*, the single transformation between that pair of frames.

PackNet (PN) PackNet [42] is a convolutional network architecture for high-resolution self-supervised monocular depth estimation. The authors propose new packing and unpacking blocks that jointly leverage 3D convolutions to learn representations that maximally propagate dense appearance and geometric information while still being able to run in real-time. The other contribution is a novel loss that can optionally benefit from the camera's velocity when available (*e.g.*, from cars, robots, mobile phones) to solve the inherent scale ambiguity in monocular vision. For the Pose Network, the architecture proposed by [43] without the explainability mask is employed.

2.1.3 Benchmarks

Benchmarks help researchers in choosing the best paradigm for their use case, and, thus, the presence of comparison analyses in a variety of settings and scenarios is crucial for a better understanding of the advantages and disadvantages of the approaches in the literature. However, most of the existing comparisons aim to prove the benefits of the proposed approach or show the superiority of its performance in a particular setting [44; 45]. Therefore, the analysis is normally limited to a couple of scenarios and performed against a small set of other approaches [7; 46]. In addition, following this workflow, the experiments are not designed to study the robustness of the algorithms with respect to input or parameter variations, *i.e.*, how changes in the inputs or in the algorithm parameters affect the metrics of interest (*e.g.*, execution time, memory resources, or estimation accuracy).

To cite some examples, the authors of DSO [14] compare their approach mainly against ORB-SLAM (since LSD-SLAM [12] and SVO [15] fail in most of the test sequences). On the other hand, in [17], VINS-Mono performance is studied solely against OKVIS [47] and, besides the tests on EuRoC [48], most of the other

experiments are qualitative (excluding the indoor sequence recorded with the OptiTrack setup). In [49], several VO methods are compared against PackNet. However, experiments are only performed on the KITTI dataset [44], whose scope is limited to automotive scenarios.

The KITTI benchmark, besides being a very popular dataset used in many works, has been established as a VO comparison suite. It has been made available on the authors' website and ranks a considerable amount of approaches, whose number increases as we speak. Nonetheless, as mentioned above, the sequences of the dataset are collected by driving a car and, thus, cannot be used to evaluate the performance of the algorithm in more challenging contexts, such as those related to MAVs.

In the last years, a number of works have emerged with the specific aim to review and benchmark VO and VIO approaches. In [45], the authors compared ORB-SLAM2, LIBVISO2 [10], SPTAM [50], RTAB-MAP [51] and ZED-VO[1] on sequences gathered by using a mobile ground rover. Tests are performed on an NVidia Jetson TX2 board, discussing trajectory accuracy, frames per second processed, and CPU usage. A similar setup is used in [52], where the localization performance of a RealSense T265 and a ZED-Mini VIO cameras are compared against ORB-SLAM2. In both cases, the analysis is limited to ground robot scenarios and they do not take into account visual-inertial or data-driven approaches.

More challenging contexts are considered by [53], where DSO, LDSO [54], ORB-SLAM2 and DynaSLAM [55] are tested on the EuRoC [48] and the TUM [56] datasets, collected with a MAV and a hand-held camera, respectively.

Visual inertial approaches are considered by [57] and [58]. While the former only provides a concise review of VIO strategies, the latter also benchmarks some of them with a special focus on augmented reality applications. Eight methods,

[1] https://github.com/stereolabs/zed-ros-wrapper

including ORB-SLAM2, DSO, and OKVIS are tested on two datasets collected with mobile phones. A generic tool to benchmark vision-based SLAM systems is proposed by [59] and further extended in [60]. In particular, the authors devised a dataset-agnostic and sensor-agnostic framework that can be easily extended to include additional algorithms and scenarios. As a use case example, they compared five vision-based SLAM algorithms.

A detailed benchmark comparison of Monocular VIO methods is proposed by [7]. In this case, the authors considered different computational units and provided an extensive analysis on the EuRoC dataset. Finally, [61] compare different algorithms, including VINS-Mono, VINS-Fusion [62] and ORB-SLAM2, on a dataset collected by the authors with a drone in an indoor scenario. They evaluate the performance of the algorithms on three different Nvidia Jetson boards.

The aforementioned studies, however, never consider data-driven approaches, and, in most cases, they do not account for many recent VO and VIO approaches. Furthermore, tests are performed only on a few scenarios, either limited to particular motion dynamics (*e.g.*, the KITTI dataset) or focused only on specific environments (such as the indoor contexts considered in EuRoC). In addition, none of them provides a detailed and extensive comparison of accuracies, memory usages, and computational times, particularly when varying algorithm hyper-parameters and input resolutions.

2.1.4 Datasets

The availability of public datasets and their heterogeneity are fundamental for researchers to validate the proposed approaches. The most important VO state-of-the-art datasets available in the community are collected in different scenarios, *i.e.*, both outdoor [44] and indoor [48; 56], with a variety of platforms, *e.g.*, aerial vehicles [48], handheld cameras [56], and automotive [44]. However, to further

extend the variety of scenarios and environmental conditions, a simulation engine can be used to collect additional datasets since it guarantees flexibility in generating indoor and outdoor scenarios and allows to arbitrary change a variety of parameters to simulate different conditions (*e.g.*, lighting and weather). In addition, the high degree of photorealism and the advanced ground and aerial simulation models provided by the simulation engine make it possible to collect sequences that are very realistic and, therefore, relevant to investigate the performance of VO/SLAM algorithms. Finally, more specific scenarios are not usually taken into account, despite their importance in different fields. For instance, racing tracks represent a more challenging scenario compared to everyday urban environments due to the lack of features and the high IMU noise caused by the strong vibrations of the vehicle. However, this may provide interesting cues with respect to the robustness of the state-of-the-art approaches and may be of interest for autonomous racing research, which is recently becoming popular thanks to international university challenges such as Roborace[1] or the Indy Autonomous Challenge[2]. Even less common are the datasets for autonomous robotic tasks in agricultural environments, despite precision farming is becoming more and more important nowadays. Those that are available are mainly focused on monitoring and precision agriculture tasks [63], such as crop/weed discrimination, plant phenotyping, leaf detection, canopy volume calculation [64], fruit counting [65] and yield estimation [66]. In [63], the authors present a detailed survey of the publicly available datasets for computer vision tasks in precision agriculture. However, they are not usually associated with a benchmark study that would favor the analysis of the challenges associated with these rural contexts, particularly with respect to the development of effective navigation algorithms and the analysis of agricultural-specific information (*e.g.*, multispectral images).

[1]https://roborace.com/
[2]https://www.indyautonomouschallenge.com/

2.2 Semantic Approaches for Localization

Vision-based robot localization solutions can be categorized into metric and semantic approaches, depending on the pose representation strategy they employ [23]. As mentioned, the former aims to compute the position and the orientation in the metric 6 degree-of-freedom space and has been extensively explored in the last decades. For those applications where the accuracy of metric approaches is insufficient or not required, semantic pose representations might provide a more robust and efficient alternative. They can be used to enhance SLAM or VO methods (*e.g.*, as loop closing modules [67; 68]), or act as stand-alone localization systems (such as in topological navigation or exploration tasks [69; 70; 71]). These approaches estimate the position by associating the observations with semantic entities that represent spatial locations.

2.2.1 Visual Place Recognition

The approaches that rely on vision sensors to extract semantic information are, in general, referred to as Visual Place Recognition (VPR) methodologies. VPR strategies [22; 23; 24] are designed to recognize places within a map based on visual, structural, and/or semantic cues. Their key modules are:

1. Place modeling: to map image data into a descriptor space.
2. Place mapping: to build and maintain a representation of the world.
3. Belief generation: to match the current observation with a place within the map.

One of the challenges of VPR is that the appearance of a location may change drastically along with different illumination conditions, viewpoints, seasons, distance, occlusion, and/or background clutter. In addition, VPR methods suffer

perceptual aliasing issues, *i.e.*, images from different places may have a similar appearance. Finally, viewpoint variation needs to be handled, since the same location seen from different perspectives might have a completely different appearance.

To address the challenges, VPR algorithms often use methods such as image preprocessing, feature selection, and machine learning techniques to improve robustness and accuracy, as using Convolutional Neural Networks (CNNs) (*e.g.*, ResNet [72] or VGG [73]) to extract robust and meaningful features.

Image-to-image VPR A simple pipeline of a VPR system can be described as follows: the raw image of a query place is converted into a feature vector, which is then used to compute the similarities between the query and the database images. The similarity score of two images indicates a belief about whether or not they are from the same place. Traditionally, VPR systems use handcrafted feature-based models to encode the images, *e.g.*, Bag-of-words (BoW) model. However, researchers started exploring deep learning applications to VPR in 2014 [74], proving that learned features outperform handcrafted ones for visual recognition tasks [75].

Nowadays, NetVLAD [76] is one of the most popular VPR solutions. It exploits an end-to-end trainable pooling layer based on a generalized VLAD (Vector of Locally Aggregated Descriptors) [77]. The underlying idea is to learn image descriptors and aggregate them into a fixed image representation. The VLAD technique is a popular method for aggregating local descriptors into a global image descriptor. The VLAD approach involves three main steps: (i) feature extraction, (ii) quantization, and (iii) aggregation. First, local features are extracted from an image frame, describing distinctive points in the image and providing information about its content. Next, the local features are quantized into a set of

visual words. This is achieved by clustering the local features into a predefined number of clusters using techniques such as k-means clustering. Each local feature is then assigned to its nearest cluster center, and a vector of frequencies is generated to represent the visual words present in the image. Finally, the VLAD technique aggregates the local feature vectors to produce a single fixed-size representation of the image. This is done by computing the difference between each local feature vector and its assigned cluster center and accumulating these differences into a single vector, called the VLAD vector. The resulting VLAD vector is then normalized to make it invariant to changes in image scale and illumination.

NetVLAD extends the VLAD approach by extracting the local features using a CNN and introducing an end-to-end trainable layer that learns to pool the local descriptors into a fixed image representation. The pooling layer is trained along with the rest of the network, which enables it to learn discriminative image representations that are well-suited for VPR tasks.

NetVLAD has been further improved by its more recent version, *i.e.*, Patch-NetVLAD [78], by considering spatial information in the aggregation process. Specifically, it generates a set of patch-level descriptors and aggregates them by including spatial weighting, based on the distance between each patch and the center of the image. This modified procedure allows a more accurate representation of the overall structure of the image. Finally, to generate the image retrievals, Patch-NetVLAD re-ranks the list of potential matches by performing a geometric consistency check using a combination of local and global descriptors extracted from multi-level patches.

Despite the impressive results achieved by [76] and [78], they exhibit two major limitations:

1. Spatial and temporal relationships, which might be present in many robot applications, are not taken into account, and, thus, not exploited.

2. Matching descriptors might become computationally intensive as the number of visual cues increases [22].

Instead, [68] proposes a place recognition pipeline to identify loop closures under viewpoint variation during the robot navigation, specifically requiring a VIO method running in the background.

Sequence-to-sequence VPR The aforementioned image-to-image strategies do not take into account temporal dependencies. Contrarily, sequence-to-sequence VPR [79; 80; 81] and loop closing approaches [82; 83; 84] consider the temporal evolution of the features to match database sequences with the query image stream. The predominant approach to exploit this sequential information is sequence matching, and it was first popularized by [79]. The key idea is to individually compare each frame of the input sequence to the database of images to build a similarity matrix. Then, this matrix is searched for the most likely trajectory by aggregating the similarity scores. The drawbacks of this method are twofold:

- The mechanism of searching through the similarity matrix relies heavily on assumptions of homogeneity regarding the motion model of the robot, making it difficult to generalize to other conditions.
- The computational cost grows linearly with both the sequence length and the size of the map.

More recent works propose to use sequential descriptors to summarize a sequence of images, enabling to directly perform a sequence-to-sequence similarity search. For instance, in [80], the authors propose three different mechanisms to fuse the information from the individual images: (i) late fusion (*i.e.*, images are processed individually and at the end a single sequence descriptor is generated), (ii) early fusion (*i.e.*, a learnable model directly processes all the frames at once), and (iii)

intermediate fusion (*i.e.*, a first learnable stage processes individual frames, then a fusion operation collects all the flows, and finally, another learnable element outputs the sequential descriptor).

2.2.2 Topological Localization

Spatial correlation Different from VPR strategies that model the environment as a database of uncorrelated entities, spatial relationships are considered by topological localization (TL) approaches. TL [24; 85; 86], pioneered by Kuipers [25] in 1978, is a special case of VPR that assumes that topological relationships between locations exist. This assumption is structured through adjacency maps, where nodes and edges represent locations and their adjacency relationships, respectively. As an instance, in [87] localization is achieved with respect to a topological map encoding high-level textual information contained in the environment, *e.g.*, street signs and shop signage. The key insight is that scene texts are high-level information invariant to illumination changes and very distinct for different places when considering spatial correlation. Thus, they may be beneficial for visual place recognition tasks under extreme appearance changes and perceptual aliasing.

Among the possible data structures that can be used to encode topological maps, the graph representation is certainly one of the most adequate. To build models over graph-like inputs, Graph Neural Networks (GNNs) are recently gaining interest as they can naturally exploit spatial correlations between adjacent locations. The authors in [69] are among the first to formalize the robot navigation and localization tasks as a graph-based problem. Their goal is to enable a robot to navigate from one location to another, relying only on its visual input and the topological map of the environment. The action space is decomposed into primitive behaviors (*e.g.*, *find door*, *corridor follow*, *turn left*, *turn right*, *straight*

into room). In order to plan a path, the agent needs to localize itself in the environment, *i.e.*, the topological map. They formulate the localization task as a GNN-based edge classification problem, thus they define the graph as follows:

- The node features depend on the node type: *room, hallway, open space.*
- The edge features can be one of the five options: *corridor follow, turn left, turn right, find door, straight (into room).*
- The global features change at each timestep and are encoded by the features extracted by a CNN from the current visual input.

The graph is cropped to the neighborhood around the last predicted location. Then, the cropped sub-graph (consisting of the nodes, edges, and global features of the current visual input) is passed through the GNN to predict the agent's current edge in the graph. The main disadvantage of this approach is that the current predicted location strongly depends on the previous one since the prediction is performed over the cropped sub-graph. Thus, the starting position of the robot should be known, and a prediction error might cause an unrecoverable loss of the robot.

Temporal correlation Different from sequence-to-sequence VPR, in topological localization, temporal dependencies can be considered only for the query image streams, as in [88], to enforce consistency between predictions of consecutive frames. Indeed, the database does not consist of a sequence of images, but it is represented as a topological map, with topological relationships among nodes. Thus, sequential descriptors cannot be extracted and, then, sequence matching cannot be applied.

More in detail, [88] presents an architecture for topological localization that models spatial and temporal correlation between observations and nodes. Specif-

ically, the authors propose to use a Graph Convolutional LSTM (GCLSTM) to localize an agent in a topological map to enhance image-based navigation in indoor environments. The localization network consists of three modules:

1. **Feature extraction:** the current image and the node images are encoded through a CNN, concatenated, and a similarity score between the current image and each node is extracted with a fully-connected (FC) layer.

2. **Spatio-temporal aggregation:** a GCLSTM is employed to simultaneously handle temporal information from time-series observations and spatial information from the topological maps.

3. **Identification:** a FC layer returns the likelihood of each node, which is fed into the softmax operator to estimate a localization probability.

However, they consider only ground robot platforms and use the last layers of a deep feature extractor (*i.e.*, ResNet-18) to encode the observations, which has been shown to reduce the performance when compared to feature maps from intermediate layers.

Indeed, recent works propose to combine feature maps from multiple layers of a deep network to exploit both lower-level and higher-level information. For instance, [89] redefines the message-passing procedure of GNNs (*i.e.*, message generation, message aggregation, and node update) to process feature maps from different layers and improve the 6 degree-of-freedom positioning performance. When the nodes in a graph encode visual information (*i.e.*, images), this updated procedure also avoids the loss of spatial information occurring when feature maps are vectorized to create a message for standard GNNs. In this case, messages are represented by the concatenation of feature maps and then aggregated through a soft attention mechanism. Differently, [90] proposes to combine the output of the 17 convolutional blocks of ResNet-50 [72] (pre-trained on ImageNet [91]) to

compute the frame features. In particular, for each convolutional block, they apply a 2D average pooling (into size $(4, 4)$), flatten, and apply Principal Component Analysis (PCA), computed per block on the training sequences. It results in a feature vector of length about 100 to 1000 per convolutional block, which is finally concatenated, resulting in a vector size in the range 15k–20k.

Chapter 3

Visual Odometry

3.1 Overview

Benchmark studies help researchers and practitioners in selecting the best components to build an autonomous system. In particular, the best VO method for localization depends on the specific experimental settings where the system runs. However, most of the existing analyses compare only a small subset of the most popular approaches on specific datasets or configurations. In addition, none of them provides a detailed and extensive comparison of accuracies, memory usages, and computational times, particularly when varying algorithm hyper-parameters and input resolutions. To fill the gap, we present two benchmark studies to analyze the performance of several VO and VIO approaches. In the first work [92] we run an extensive set of experiments with different setups, *i.e.*, a GPU-equipped workstation and an Nvidia Xavier AGX board. Estimation accuracy, memory usage, and computational time are compared and discussed by varying the hyper-parameters and the input resolutions. The second work [93] focuses on the agricultural scenario, and we benchmark some of the most recent Vision-based and LiDAR-based Odometry.

To extend the possible test conditions and scenarios, we collect and make available to the community three additional datasets[1]. One gathered with a highly photo-realistic simulation engine, which includes sequences from different environments, both indoor and outdoor. Another one is instead collected with a Formula 3 vehicle on a racing circuit. These two comprise data gathered by a stereo camera and an IMU. Finally, we collect a dataset in agricultural scenarios, *i.e.*, in vineyards and olive cultivations. The latter includes data from a stereo camera, IMU, GPS, a multi-spectral camera, and a LiDAR.

3.2 First experimental campaign

This section presents the setup used to compare the selected VO/VIO approaches in [92]. First, the dataset used for the analysis is detailed. Afterward, the implementation details are introduced and the metrics and the test characteristics are described.

3.2.1 Datasets

During the preliminary study, we analyzed the state-of-the-art datasets available in the community to compare the selected VIO/SLAM techniques. We decided to not include some of them due to the lack of sensor data necessary to run some of the considered algorithms. In particular, the TUM-RGBD [94] does not provide complete IMU data (only the accelerometer is available). Similarly, the ICL-NUIM dataset [95] does not contain IMU readings. Therefore, these two datasets are not adequate to evaluate and compare the VIO algorithms we considered in this paper. The KITTI dataset [44] does provide IMU readings at 100Hz, however, they are not synchronized with images and, therefore, cannot be used.

[1] https://isar.unipg.it/datasets/

A synchronized version of the IMU data exists, however, its rate is 10Hz and is too low for VIO methods to work properly. For these reasons, we selected the following datasets for a fair comparison of the considered techniques.

EuRoC The EuRoC Dataset [48] is composed of 11 sequences recorded by flying an MAV into three indoor environments: *machine hall*, *Vicon room 1* and *Vicon room 2*. It consists of stereo images (collected with an Aptina MT9V034 global shutter, WVGA monochrome, 2×20 FPS) and synchronized IMU measurements (ADIS16448, 200 Hz). The ground-truth poses are collected through a Vicon motion capture system (6D pose) and a Leica MS50 laser tracker (3D position), depending on the environment. The sequences are divided into *easy*, *medium* and *difficult*; this classification depends on the MAV speed, scene texture and illumination.

TUM In contrast to EuRoC, the TUM-VI Dataset [56] is collected with hand-held fisheye cameras. It is composed of 28 sequences recorded in both indoor and outdoor scenarios and provides camera images with 1024x1024 (or 512x512) resolution at 20Hz. The IMU works at 200Hz and has a hardware-level time synchronization with the cameras. Ground-truth poses are not available for all sequences, thus, we split the TUM-VI Dataset into two subsets:

- *TUM_room*: it contains all the sequences that have a complete ground-truth;
- *TUM_no_room*; it contains a subset of the sequences whose ground-truth poses cover only parts of the overall trajectory. In this case, we compute the metrics only on the sub-sequences associated with the ground-truth.

Unreal To further extend the variety of scenarios and environmental conditions, we use a simulation engine to collect an additional dataset. We make this choice since it guarantees flexibility in generating indoor and outdoor scenarios and allows to arbitrarily change a variety of parameters to simulate different conditions (*e.g.*, lighting and weather). In addition, the high degree of photorealism and the advanced MAV simulation model provided by the simulation engine make it possible to collect sequences that are very realistic and, therefore, relevant to investigate the performance of VIO/SLAM algorithms.

In particular, we rely on Unreal Engine 4 (UE4)[1]. The MAV dynamics are simulated by using the AirSim [96] plugin. The simulated drone is equipped with a stereo camera, recording at 20 FPS with 640x480 resolution, and an IMU with a 200Hz sampling rate. We record 24 sequences in 6 different simulated environ-

[1]https://www.unrealengine.com/

(a) Abandoned Factory (b) City Park (c) Urban City

(d) Office Building (e) Office Space (f) Indoor Factory

Figure 3.1: Example images from the Unreal dataset. The top row contains frames from the outdoor sequences, while the bottom one depicts examples from the indoor scenarios.

ments, 4 sequences each. Three of these environments simulate outdoor scenarios (*i.e.*, *Abandoned Factory*, *City Park* and *Urban City*), while the remaining ones refer to indoor contexts (*i.e.*, *Office Building*, *Office Space* and *Indoor Factory*). When running the experiments, to analyze the differences among indoor and outdoor scenes, we group the first three in the *Unreal Outdoor* subset and the latter in the *Unreal Indoor* one. Some example images of this dataset are shown in Figure 3.1.

Marzaglia We collect the Marzaglia dataset with a Formula 3 car at the Marzaglia racing track (in Modena); some images are reported in Figure 3.2. The car was equipped with a RealSense D455, composed of an IMU, recording at 250 Hz, and a stereo camera, which provides images at 30 FPS with a resolution of 640x480 pixels and a field of view of 90 degrees. Images are cropped to 640x430 to remove the vehicle from the picture. Pose ground truth is obtained by collecting GPS measurements with an Emlid Reach M2, while RTK corrections are provided by an Emlid Reach RS2. With this sensor setup we recorded three sequences, each one associated with one of the three laps around the circuit we performed by driving the vehicle.

The scenarios proved to be particularly challenging due to the lack of features and the high IMU noise caused by the strong vibrations of the vehicle. Moreover, it is important to mention that the ground truth was collected by only using GPS

(a) (b) (c)

Figure 3.2: Example images from the Marzaglia dataset.

measurements, thus excluding the rotation.

3.2.2 Implementation details

Hyper-parameters As we mentioned in the previous Sections, one of the objectives of this work is to explore the effect of the different hyper-parameters of the approaches on the performance. Therefore, before providing the implementation details, it is important to distinguish between:

- The *default* hyper-parameter configurations, which are either provided by the respective authors for the publicly available datasets or those we experimentally found to guarantee the best performance on the sequences we collected (*i.e.*, on the Unreal and the Marzaglia datasets).
- The *alternative* hyper-parameter configurations, which we explore to analyze the trade-off between computational efficiency and accuracy.

Platforms For the evaluation of each algorithm, we used two different platforms: a workstation with high computational capabilities and an Nvidia Jetson Xavier AGX embedded board. The latter has been selected as a case study to evaluate the performance of the VO/VIO approaches when run onboard.

3.2.3 Experimental setup

Algorithms For the benchmark analysis, we selected several geometric and data-driven State-of-The-Art methods:

- DSO [14] and ORB-SLAM3 Mono [37] for monocular camera setups;
- VINS-Mono [17], ROVIO [16], and ORB-SLAM3 Mono-Inertial [37] for monocular-inertial setups;

- Open-VINS [18] and ORB-SLAM3 Stereo-Inertial [37] for stereo-inertial setups;
- SfMLearner [39], Competitive Collaboration [40], Monodepth2 [41], and PackNet [42] for monocular data-driven setup.

The characteristics of the aforementioned strategies are reported in Section 2.1.1 and Section 2.1.2.

Metrics The selected approaches are compared by taking into account different aspects: from estimation accuracy to execution time and memory usage. These quantities are evaluated for each dataset and platform, and for different hyper-parameter configurations.

1. **Pose Estimation Accuracy:** To compute the pose accuracy we employ the *evo* toolbox[1]. This utility is specifically designed for the evaluation of the trajectories estimated with visual odometry and SLAM algorithms. It provides options for timestamp association, trajectory alignment and scale adjustment. The package makes available different quantitative metrics to compare the estimated and the ground-truth trajectories. Among these, in this work, we considered the Absolute Pose Error (APE), for both the translational and the rotational components of the poses. More specifically, if we refer to the estimated and the ground-truth pose at timestamp i as $P_{est,i}, P_{gt,i} \in SE(3)$, we can compute:

$$E_i = P_{est,i} \ominus P_{gt,i} = P_{gt,i}^{-1} P_{est,i} \in SE(3) \tag{3.1}$$

with $\ominus$ indicating the inverse compositional operator, which takes two poses and returns the relative pose [97]. We can define the absolute translational

[1] https://github.com/MichaelGrupp/evo

error (ATE) and the absolute rotational error (ARE) as:

$$ATE_{t,i} = ||trans(E_i)|| \tag{3.2}$$

$$ARE_{t,i} = |angle(log_{SO(3)}(rot(E_i))| \tag{3.3}$$

where $log_{SO(3)}(\cdot)$ is the inverse operation of $exp_{SO(3)}(\cdot)$. Finally, we compute the Root Mean Square Error (RMSE) by averaging the ATE and ARE over all the sequence poses:

$$RMSE_{ATE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ATE_i^2} \tag{3.4}$$

$$RMSE_{ARE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ARE_i^2} \tag{3.5}$$

These metrics are computed after performing both scale and reference system alignment.

2. **Execution time:** For each approach, we measure the total time required to process an input frame and provide the associated pose estimate. In addition, in the case of geometric approaches, we also analyze the execution time of the different functional blocks involved in the computation. In particular, we consider:

   - Feature extraction time (or pixel selection, *e.g.*, in DSO);
   - Tracking time (which only applies for DSO, OV, OS, and VM);
   - Marginalization time (which only applies for DSO, OV, and VM);
   - Update time (or optimization step).

   It is important to highlight that the summation of these subparts does not

result in the total execution time for a frame, because those subparts are not always executed and other parts could be present in each specific algorithm. For data-driven approaches, we only consider the total time since we can regard the deep network forward pass as a single atomic operation.

3. **Memory usage:** To ease the analysis of the memory usage, we build a docker image for each approach and, for each sequence, we measure the difference between the amount of memory of a docker container while running the algorithm and its value at the beginning (which is usually close to 0). We then report the peak value of this difference as the maximum memory consumption. Note that the implementations of data-driven methods exploit the GPU acceleration and, therefore, the value reported in the experiments refers to the GPU memory.

4. **Failures:** We also record whether an approach completely fails to provide a pose estimate when processing a specific sequence (*e.g.*, it fails to initialize the feature tracker). This is indicated as a *fail* in the tables reporting the quantitative results. Instead, in the *hyper-parameter* exploration plots, we count the number of sequences where the algorithm experiences a failure.

5. **Ratio of trajectory length:** Some implementations of the geometric strategies stop to provide pose estimates for the incoming images when the performance of some of their functional blocks drop, *e.g.*, they fail to extract or track features. In these cases, it is not possible to compute the $RMSE_{ATE}$ or the $RMSE_{ARE}$. To capture this aspect, after aligning the scale and the reference system of the estimated trajectory with that of the ground truth one, we compute the ratio between their lengths:

$$R_{traj} = \frac{l(t_{est})}{l(t_{gt})} \tag{3.6}$$

where $l(t_*)$ computes the length of the trajectory. $R_{traj} \to 0$ when $l(t_{est})$ is close to zero, which means that the algorithm failed to provide predictions for most part of the sequence. Conversely, $R_{traj} \gg 1$ if t_{est} is longer than the reference one, most likely due to estimation errors. The best value for this metric is achieved when $l(t_{est}) \approx l(t_{gt})$, however, this does not necessarily correspond to a perfect prediction, since the two trajectories could be different (*i.e.*, t_{est} is not accurate) and yet have similar lengths by chance. Therefore, this metric should always be analyzed in conjunction with the $RMSE_{ATE}$ and the $RMSE_{ARE}$, in particular:

- low $RMSEs$ and $R_{traj} \approx 1$ indicate good estimations;
- low $RMSEs$ and $R_{traj} \approx 0$ denote that, despite the accuracy of the approach is high, it fails to provide estimates most of the time, *i.e.*, its robustness is low;
- high $RMSEs$, and $R_{traj} \approx 1$ or $R_{traj} \gg 1$ refer to outcomes with low accuracy.

Hyper-parameter exploration As mentioned before, in this work we study how changing the hyper-parameter configuration of the various approaches affects the trade-off between pose accuracy, execution time and memory usage. To this aim, we explore a set of *alternative* configurations, whose values affect the accuracy and the efficiency of the algorithm. We consider and explore, in particular, the following quantities:

- Input image resolution: downsampling the input frame reduces the computational time, however, this could result in lower accuracy. By referring with R to the resolution of the *default* configuration, we also explore the values $1/2R$ and $3/4R$.

- Maximum number of features: decreasing this number might speed up the algorithm, but fewer features will be tracked. We indicate with MF its value in the *default* configuration and consider $0.8MF$ and $1.2MF$ in the *alternate* ones.

- Solver iterations SI. This hyper-parameter applies only to DSO, ROVIO, and VM. In the *alternate* configuration, we test these approaches by setting it to $1/2SI$ and $2SI$.

- Number of frames in the considered window NF. This hyper-parameter applies only to DSO and Open-VINS. As for the previous one, in the *alternate* configuration, we explore the values $1/2NF$ and $2NF$.

It should be noticed that only the effect of the input image resolution can be studied for data-driven approaches. In this case, we train the models by using the default value and then perform tests on R and by downsampling the input image to $1/2R$ and $3/4R$.

3.2.4 Results and discussion

In this Section, we present the results obtained for each dataset. Besides tables showing quantitative values for the different metrics employed, we use three different plots to study the performance of the approaches:

- *Summary charts*: They offer an overview of the performance of all the selected methods on the AGX Xavier (square) and the workstation (circle) for the *default* hyper-parameters. We differentiate between the translational and rotational errors, by providing for each dataset two charts, one reporting the $RMSE_{ATE}$ on the x-axis and the other one the $RMSE_{ARE}$. The y-axis, instead, refers to the execution time (ms), while the radius of

the marker is proportional to the memory usage. The best trade-off among all the metrics is represented by the methods that have a small radius and lie in the left-bottom corner of the chart. For geometric approaches, the values are averaged over all the sequences of the dataset where no failures are experienced, while for data-driven only those selected as the test set are considered.

- *Hyper-parameter exploration plots*: These kinds of charts report all the parameter variations on the x-axis. For each possible value, the metrics considered are plotted, therefore, the y-axis does not have a proper measure unit. The chart is divided into two subplots to improve readability while keeping the linear scale on the y-axis: (i) the bottom part is a focus on the range 0 to 3 to appreciate small values (*e.g.*, $RMSE_{ATE}$) (ii) the top part covers the range 3 to 160 to see variation on higher values (*e.g.*, execution times in milliseconds).

- *Memory usage and execution time box plots*: These are used to provide a statistical analysis of the computational resources required by the various methods. For geometric ones, we also provide the execution time of the different algorithm phases.

In the following, the complete set of quantitative results is reported only for the EuRoC dataset. Additionally, a selection of quantitative and qualitative results in the Marzaglia dataset is presented to better understand the limitations in challenging scenarios. The exhaustive list of results for the remaining datasets is presented in [92].

EuRoC Figure 3.3 depicts the summary of the baseline parameters for the EuRoC dataset. In the workstation platform, the best methods in terms of trajectory

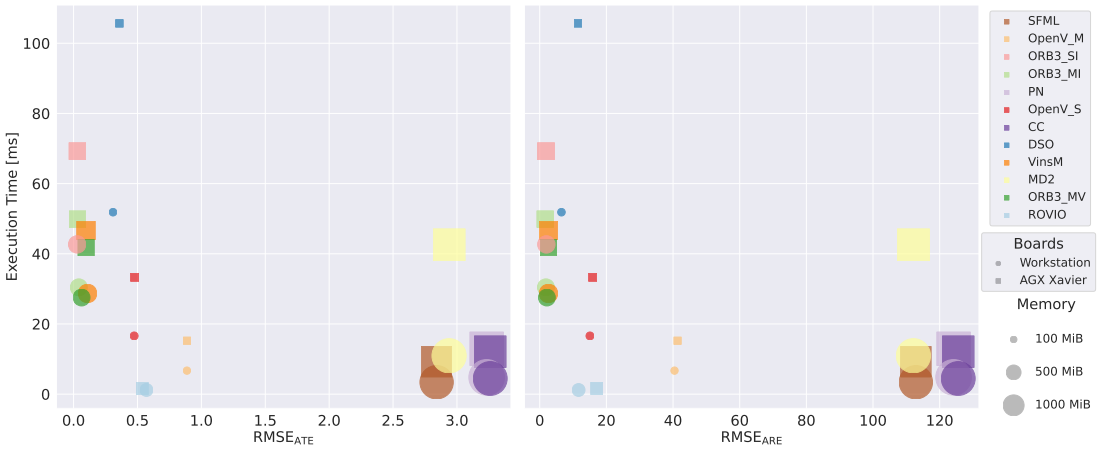
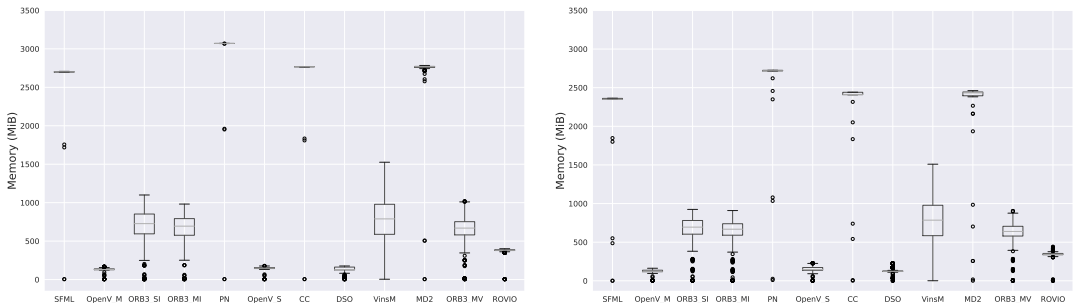


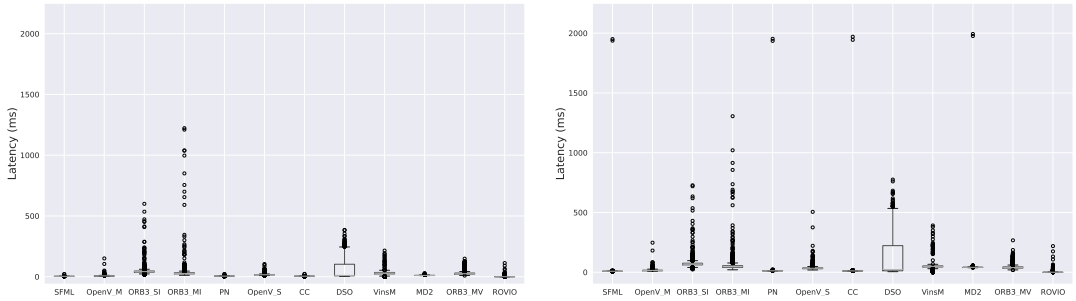
Figure 3.3: Summary charts for the EuRoC dataset. The errors reported are obtained with the *default* hyper-parameter configuration. The chart on the left shows the results with respect to the $RMSE_{ATE}$ (in meters), while the one on the right with respect to the $RMSE_{ARE}$ (in degrees).



(a) Workstation (b) Xavier AGX

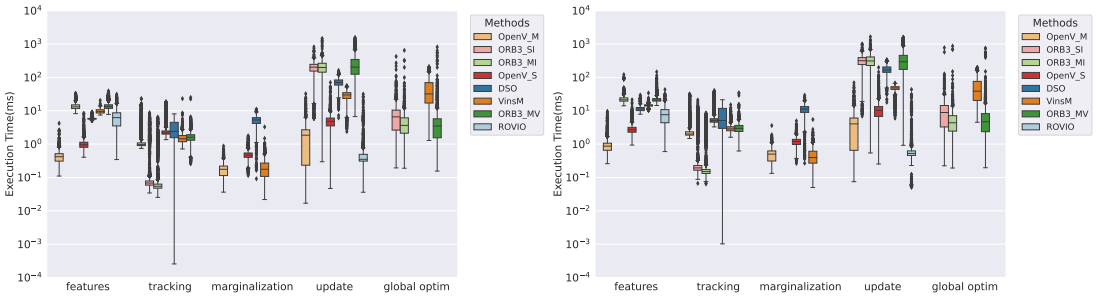
Figure 3.4: Memory usage of the compared approaches on the EuRoC dataset with the *default* configuration.

accuracy are the SLAM approaches, namely ORB-SLAM3 (OS3), obtaining good results in every configuration, and VINS-Mono (VM). It is not surprising that SLAM methods dominate VO ones, especially for EuRoC, which is the testing dataset against which most of the literature methods are compared. Besides, given the higher complexity of OS3 and VM with respect to the others, it is also expected that they have higher latency, due to additional phases (*e.g.*, graph optimization), and require more memory to store the evolving map (although they



(a) Workstation (b) Xavier AGX

Figure 3.5: Total execution time of the compared approaches on the EuRoC dataset with the *default* configuration.



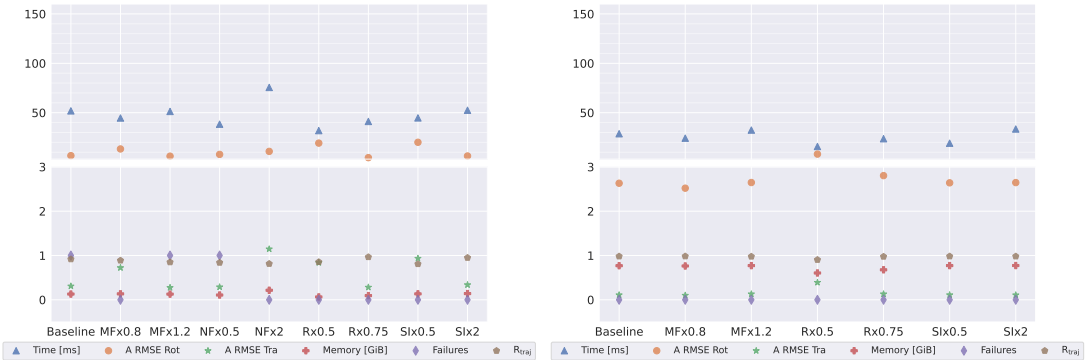
(a) Workstation (b) Xavier AGX

Figure 3.6: Execution Time analysis for the different functional blocks of the geometric methods on EuRoC dataset. The results are obtained by using the *default* hyper-parameter configuration.

never exceed 800 MiB).

Among the VO methods, DSO obtains the smallest errors, followed by ROVIO and Open-VINS (OV). Those methods are also the most memory-efficient, requiring only 100-350 MiB of RAM. Additionally, ROVIO is the fastest algorithm, closely followed by Open-VINS Mono (OV-M) and Open-VINS Stereo (OV-S); DSO instead has a latency that is 55 times higher than ROVIO.

Finally, data-driven methods obtain errors that are considerably higher (in some cases 150 times worse in $RMSE_{ATE}$ and 60 times worse in $RMSE_{ARE}$) than the best one and can require up to 2.5 GiB of RAM to store the CNNs and



(a) DSO (b) VM

Figure 3.7: Performance analysis for various hyper-parameters configurations on the EuRoC dataset (Workstation). The charts report the results for DSO and VINS-Mono.

perform the forward passes, being the most memory greedy approaches. However, they are almost as fast as ROVIO and OV-M.

On the Nvidia Jetson Xavier AGX platform, the classification of the methods is almost equivalent in every analyzed metric. Besides an intuitive upward deviation in terms of time latency for each method, a small deterioration of trajectory error and memory usage can be observed for geometric methods. The cause can be traced back to the use of non-deterministic techniques within the method, such as outliers removal performed using RANSAC for feature extraction, which makes the method itself non-deterministic.

Figure 3.4 and Figure 3.5 offer a better focus on the distribution of memory usage and execution times respectively, on the Workstation (a) and the AGX Xavier (b).

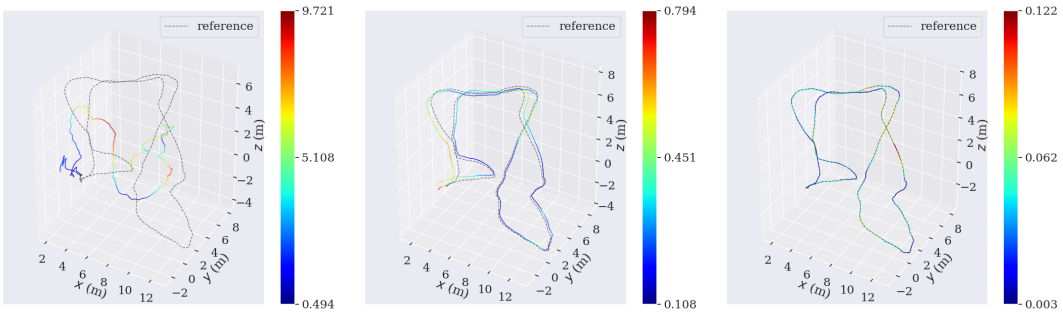
Considering only the geometric methods, the execution times of the main subparts of the algorithms are shown in the y-axis of Figure 3.6 (logarithmic scale):

- *Features*: OS3 methods are the ones that spend more time extracting fea-

tures, while OV ones are the fastest. ROVIO is the one with the greatest variance.

- *Tracking*: it is a quite lightweight phase for each method, but especially for the inertial versions of OS3. DSO has a very high variance. ROVIO is not reported because the tracking is performed along with the update.
- *Marginalization*: this subpart is the fastest for all the considered methods, except for DSO.
- *Update*: it is the heaviest part of the whole method and has a higher impact on the total execution time. The fastest methods to perform the update are ROVIO and OV-M, which are also the winning methods in terms of latency among all.
- *Global optimization*: generally, it is a phase of high variance (since it depends on the map size and the optimization constraints). The algorithm that is most affected by this phase is VM.

Further considerations can be drawn from Table 3.1, which reports the quantitative results of all the approaches on the two sequences selected as the test set for data-driven methods (*i.e.*, *MH-04 difficult* and *MH-05 difficult*). Rotation errors are the main cause of the lower accuracy of data-driven methods. Their poor estimation performance is also reflected by the low R_{traj} values, meaning that estimated and reference trajectories are considerably different. Since data-driven approaches always provide VO estimations, such a bad ratio is not caused by missing poses, but is entirely due to the estimation error. On the other hand, the performance of the geometric approaches is much better, with a R_{traj} close to 1 and a small RMSE. The only exception is made by DSO that exhibits less robustness to challenging sequences, proven by the failure on *MH-04 difficult*. Some qualitative examples are depicted in Figure 3.8.



(a) MD2 (b) OV-M (c) OS3-SI

Figure 3.8: Trajectory plot for Monodepth2, Open-VINS Monocular and ORB-SLAM3 Stereo-Inertial on the sequence *MH-05 difficult* of the EuRoC dataset. When ideal conditions are present, *e.g.*, texture-rich environments, both VO and SLAM geometric approaches outperform by far the data-driven ones. In the Figures, the black dashed line represents the reference trajectory (*i.e.*, the ground-truth poses), while the colored line represents the trajectory estimation. In particular, the color depends on the RMSE (ATE) of that specific pose.

We also report the hyper-parameter variation plots for DSO and VM on the workstation platform (see Figure 3.7). In general, the expected behavior for hyper-parameter variation is the following: (i) improvement in latency and memory consumption along with precision deterioration and more frequent failures when reducing R, MF, NF; (ii) improvement in latency along with precision deterioration when reducing SI; (iii) improvement on precision and reduction of failures along with latency and memory consumption deterioration when increasing MF, NF; (iv) improvement on precision along with latency deterioration when increasing SI.

Somewhat unexpectedly, DSO has a better RMSE using a smaller resolution. This is probably due to noisy visual cues, that in a smaller resolution are not considered or discarded. Moreover, it is interesting to observe that for VM, halving SI and setting $3/4R$ reduces the computational time without significant changes in the accuracies.

		EuRoC									
		MH-04 difficult					MH-05 difficult				
		RMSE (ATE)	RMSE (ARE)	R_{traj}	Memory (GiB)	Time (ms)	RMSE (ATE)	RMSE (ARE)	R_{traj}	Memory (GiB)	Time (ms)
Geometric	DSO	*fail*	*fail*	*fail*	*fail*	*fail*	0.192	2.129	0.98	0.164	36.17
	VM	0.156	1.139	0.99	1.042	30.27	0.135	0.942	1.00	1.077	30.62
	OV-M	0.453	2.102	0.98	0.132	6.98	0.356	1.050	0.98	0.131	7.16
	OV-S	0.149	1.307	0.97	0.167	15.92	0.199	0.760	0.96	0.170	16.39
	ROVIO	1.016	3.039	0.99	0.374	1.08	1.008	4.089	1.01	0.368	1.01
	OS3-M	0.060	1.483	1.03	0.759	25.67	0.073	1.704	1.03	0.794	26.78
	OS3-MI	0.137	1.155	1.01	0.787	29.77	0.054	0.602	0.97	0.818	30.12
	OS3-SI	0.051	1.499	0.97	0.835	42.18	0.044	0.902	0.97	0.868	42.66
Data-driven	SFML	5.528	103.847	0.45	2.639	3.44	5.563	105.656	0.42	2.634	3.37
	PN	6.450	118.905	0.22	2.995	4.83	6.592	120.712	0.17	2.994	4.75
	CC	6.430	130.923	0.24	2.699	4.47	6.666	133.519	0.16	2.700	4.44
	MD2	4.686	96.873	0.66	2.714	10.78	5.053	102.743	0.56	2.713	11.08

Table 3.1: Workstation quantitative results on the test sequences of the EuRoC dataset.

Marzaglia The Marzaglia dataset is certainly the most challenging scenario we considered. From Table 3.2 and Figure 3.9 it can be immediately noticed that nearly all the geometric methods have very poor performance. This is due to the lack of textures and of distinctive visual cues since most of the scene contains flat and regular elements (*i.e.*, the asphalt, the grass, and the sky). As a consequence, the pose estimation RMSEs are considerably higher and they fail to provide estimates for most part of the sequences, resulting in very low R_{traj}. Only DSO resulted in a good trajectory estimation for two out of three tested sequences, both in terms of low $RMSE_{ATE}$ and $R_{traj} \approx 1$.

In contrast, data-driven approaches exhibit better capabilities: the pose estimation accuracy compares to that of geometric methods, however, the R_{traj} is higher and close to 1 for MD2, CC, and PN. We can observe this aspect by observing the qualitative trajectory plots in Figure 3.10. Note also that data-driven methods are trained on the KITTI sequences, hence, despite the context is the same (*i.e.*, automotive), scenarios and camera configurations are different. This proves that they can generalize over similar conditions without the need for re-training procedures.

A more detailed quantitative analysis shows that the $RMSE_{ATE}$ are in the range 53.8(DSO)-214.4(OS3-MV), excluding estimations with too low R_{traj} values. It should be noted that such errors must be considered with respect to the total length of the trajectories, which are in the range $1353m$(lap-01)-$2602m$(lap-03). The approach that performs better is MD2, a data-driven one. Moreover, it should be noticed that the methods can be divided into three groups with respect to the ratio of trajectory length values: (i) $R_{traj} \approx 1$, which comprises CC, MD2, PN, DSO; (ii) $R_{traj} \approx 0.4$ with SFML, OV-S, OV-M, OS3-MV and ROVIO; (iii) $R_{traj} \approx 0.1$ with VM, OS3-MI, OS3-SI. Therefore, for the Marzaglia dataset, the best choices are MD2, CC, and PN, achieving the smallest errors while being fast

		Marzaglia											
		lap-01 (1353m)				lap-02 (1939m)				lap-03 (2602m)			
		RMSE (ATE)	R_{traj}	Memory (GiB)	Time (ms)	RMSE (ATE)	R_{traj}	Memory (GiB)	Time (ms)	RMSE (ATE)	R_{traj}	Memory (GiB)	Time (ms)
Geometric	DSO	123.094	0.53	0.078	27.71	53.804	0.99	0.079	32.87	87.898	0.80	0.081	33.76
	VM	28.296	0.07	0.804	58.63	204.181	0.22	0.867	70.10	35.461	0.11	0.862	63.66
	OV-M	130.889	0.32	0.105	3.20	*fail*	*fail*	*fail*	*fail*	*fail*	*fail*	*fail*	*fail*
	OV-S	117.724	0.35	0.127	7.79	*fail*	*fail*	*fail*	*fail*	*fail*	*fail*	*fail*	*fail*
	ROVIO	87.361	0.49	0.327	0.62	180.139	0.21	0.352	0.51	195.484	0.11	0.354	0.59
	OS3-MV	82.459	0.22	0.776	19.61	211.694	0.55	0.801	18.47	214.437	0.43	0.798	18.37
	OS3-MI	*fail*	*fail*	*fail*	*fail*	53.203	0.23	0.750	21.44	54.495	0.09	0.724	23.66
	OS3-SI	*fail*	*fail*	*fail*	*fail*	148.625	0.19	0.861	37.53	196.508	0.11	0.863	36.40
Data-driven	SFML	123.801	0.43	2.636	3.22	141.664	0.39	2.635	3.30	147.591	0.31	2.641	3.19
	PN	54.190	0.83	2.987	3.52	102.533	0.87	2.992	3.53	122.494	0.67	2.994	3.47
	CC	159.719	1.00	2.704	4.64	194.546	0.80	2.694	4.67	168.713	0.90	2.717	4.68
	MD2	68.172	1.00	2.690	10.95	67.907	1.08	2.689	10.95	96.662	0.91	2.688	11.26

Table 3.2: Workstation quantitative results on the sequences of the Marzaglia dataset.

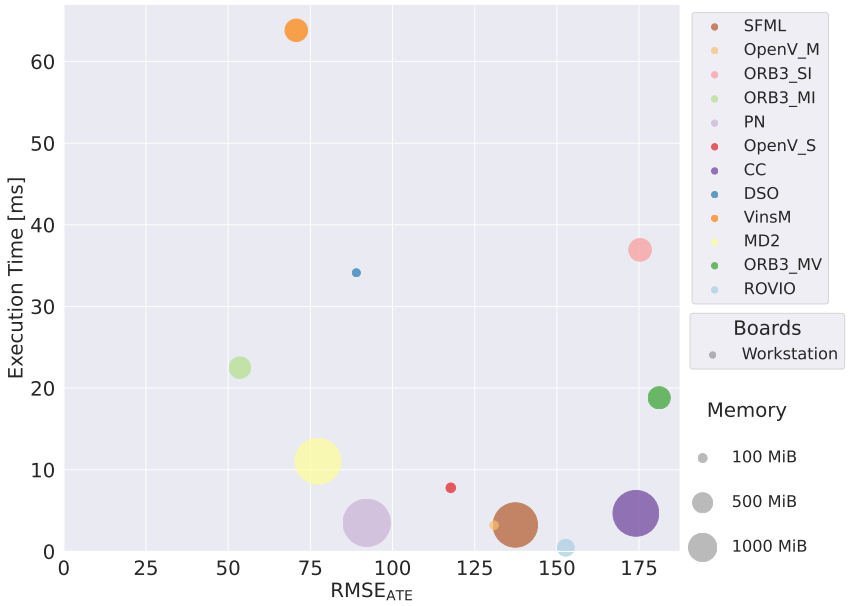


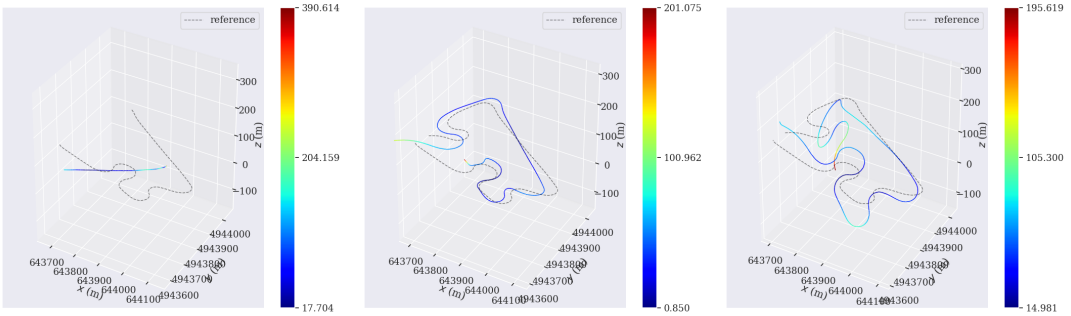
Figure 3.9: Summary chart for the Marzaglia dataset. The errors reported are obtained with the *default* hyper-parameter configuration. The chart shows the results with respect to the $RMSE_{ATE}$ (in meters).

but memory greedy, and DSO, which is at least 3 times slower but only requires 80 MiB.

Regarding execution time and memory usage, we can draw the same conclusions as for the previous dataset, except for DSO which is faster, and VM which is 2 times slower.

3.3 Second experimental campaign

This section contains the benchmark analysis in agricultural scenarios performed in [93]. In particular, it shows two use cases of the proposed dataset, *i.e.*, a comparison between State-of-the-Art LiDAR-based and Vision-based localization approaches and the computation of geo-referenced NDVI maps.



(a) OS3-SI (b) DSO (c) MD2

Figure 3.10: Trajectory plot for ORB-SLAM3 Stereo-Inertial, DSO and Monodepth2 on the sequence *lap02* of the Marzaglia dataset. It can be observed that the pose estimation performance of OS3-SI is very poor. Among the geometric approaches, DSO is the only one estimating a good trajectory. On the other hand, the trajectory computed by MD2 is accurate and similar to the reference one. In the Figures, the black dashed line represents the reference trajectory (*i.e.*, the ground-truth poses), while the colored line represents the trajectory estimation. In particular, the color depends on the RMSE (ATE) of that specific pose.

3.3.1 Datasets

We provide a dataset of rural outdoor unstructured environments gathered in two different arboreal crop scenarios. The dataset, named ARD-VO, consists of sequences collected in vineyards and olive crops located in Umbria, Italy. The alias names, varieties, and locations of the crops within the dataset are reported in Tables 3.3 and 3.5. The sequences include data collected with a stereo-camera rig, a Velodyne LiDAR, a GPS-RTK module with IMU, and a multi-spectral camera. At the same time, inverter states are collected providing additional information about robot motors operation. The data acquisition campaign was performed between the summer and the autumn of 2021, from August to October. The robot was driven in manual mode at an average speed of 1 m/s to guarantee the reliability of the recorded data both for agricultural and navigation analyses. In vineyards, the average traveled distance for each session was about 8000 meters, while in olive groves the trajectory length spans from 400 to 1000 meters.

Alias name	Crop Variety	lat(N) , lon(E)	# Sessions	Date: dd, mm, yyyy
Vynrd A	Grechetto Todi G5	43.004491, 12.294889	1	3 09 2021
Vynrd B	Grechetto Todi G5	42.812355, 12.418741	2	4 08 2021 1 09

Table 3.3: Vineyards employed for the dataset data collection

Vineyards The vineyard crops are 24 km from each other and have the same "Grechetto Todi G5" [98] cultivar, a local white wine grapevine variety. Inside the vineyards, the objective was to create NDVI maps using Geo-tagged multi-spectral images: yield estimation [99], water stress [100; 101; 102] and pests monitoring [103] are examples of how these maps can be used in agriculture. Additionally, in Table 3.4 we report information on the production characteristics and the indices of the vegetative-productive balance of the vineyards. These values are obtained by performing manual inspections in different zones of the crops. In particular, we provide these values grouped with respect to the NDVI index computed at that location. Examples of dataset samples are shown in Figure 3.11.

(a) (b) (c)

Figure 3.11: Example images from the dataset collected in olive cultivations.

Olive cultivations The olive groves are very close to each other (less than a kilometer) and are located in the agricultural site of Deruta, Umbria. According to the genetic biodiversity of the geographical area, there are three olive varieties

	Vynrd A			Vynrd B		
NDVI	**< 0.83**	**> 0.84 < 0.90**	**> 0.91**	**< 0.83**	**> 0.84 < 0.90**	**> 0.91**
Surface percentage	7%	45%	48%	20%	65%	15%
Avg grape weight (g)	1.57	1.64	1.73	1.06	1.50	1.78
Grape (kg/log)–(t/ha)	1.65–5.5	2.05–6.9	2.89–9.5	1.59–5.3	2.25–7.5	2.67–8.9
Sugars (°Brix)	21.1	21.4	23.7	22.0	21.8	23.0
Titratable acidity (g/L)	5.4	5.5	6.0	5.4	5.3	5.9
Must pH	3.26	3.28	3.21	3.4	3.4	3.3
Polyphenol (mg/g s.s.)	15.9	14.5	8.0	17.2	15.8	10.1
Grapes sun-burn (%)	21%	14%	3%	16%	10%	0%
Foliar surface (m^2/tree)	1.14	2.75	3.90	1.65	2.90	4.22
Pruning wood (kg/tree)	0.20	0.31	0.55	0.25	0.37	0.61
Foliar surface/grape (m^2/kg)	0.71	1.22	1.42	1.02	1.28	1.57
Grape/pruning wood (kg/kg)	8.06	7.27	5.04	6.40	6.10	4.38

Table 3.4: Production characteristics and indices of the vegetative-productive balance of the vineyards grouped with respect to different NDVI zones, *i.e.*, low vigor (NDVI < 0.83), medium vigor ($0.84 <$ NDVI < 0.90), and high vigor (NDVI > 0.91). The values are computed by manual inspection in different zones of the crop.

in these crops: "Moraiolo", "Leccino", and "Frantoiano" [104]. The first olive grove (alias OlvCs A) has only the "Moraiolo" variety (mono cultivar crop), while the second (alias OlvCs B) has all the three types (multi cultivar). Olive crops suffer from the infestation caused by the olive fly pest *Bactrocera Oleae* [105; 106; 107] and continuous monitoring can help the prevention and pest control. Table 3.6 shows the growth stage of the olives for each date in which we performed a collection session (between 14 September and 13 October 2021). The growth stage is expressed with respect to the international BBCH scale [108], which describes the phenological development of olive trees from bud development (0) to senescence (92). Examples are shown in Figure 3.12.

3.3.2 Implementation details

Image sequence post-processing Due to voltage oscillations, shocks, network congestion, and ROS node miscommunications, we experienced spurious

(a) (b) (c)

Figure 3.12: Example images from the dataset collected in olive cultivations.

Alias name	Crop Variety	lat(N) , lon(E)	# Sessions	Date: dd, mm, yyyy
OlvCs A	Moraiolo	42.967206, 12.407057	4	14-23-30 09 13 10 2021
OlvCs B	Moraiolo, Leccino, Frantoiano	42.961702, 12.412744	4	14-23-30 09 13 10 2021

Table 3.5: Olive cultivations employed for the dataset data collection

and corrupted frames among the raw RGB image sets. In order to produce a high-quality dataset, the collected video sequences were, therefore, subjected to a frame-by-frame automated check to ensure their integrity and guarantee frames equally spaced in time. As a result, the image streams in the post-processed sequences are free from corrupted data and characterized by a framerate in the range of 8-10 FPS.

3.3.3 Experimental setup

The ARD-VO dataset represents a challenging scenario to test the robustness of pose estimation algorithms, since several non-ideal conditions are present, *e.g.*, repetitive texture, lighting variations, or non-smooth motions of the robot.

Algorithms This section discusses the results obtained by comparing different pose estimation algorithms. For this purpose, we selected a set of approaches among the most popular for both vision-based (see [92]) and LiDAR-based (see

Crop	Date: dd/mm/yy			
	14/09/21	23/09/21	30/09/21	13/10/21
OlvCs A	80	80	85	89
OlvCs B	80	81	85	89

Table 3.6: Growth stage of the olive trees based on BBCH scale. Values around $80-81$ indicate the beginning of fruit coloring, 85 shows the increase of specific coloring, while 89 denotes the best stage for harvesting (*i.e.*, suitable for oil extraction, when fruits reach the typical color while still remaining turgid).

[109] and [110]) pose estimation. In particular, we considered:

- DSO [14] and ORB-SLAM2 Mono [35] for monocular camera setups;
- VINS-Mono [17] for monocular-inertial setups;
- ORB-SLAM2 Stereo [35] for stereo camera setups;
- Open-VINS [18] for stereo-inertial setups;
- FLOAM [111] and LeGO-LOAM [112] for LiDAR-based setups.

We performed comparative experiments on a selection of eight sequences, considering two sequences for each cultivation field. Quantitative evaluation is performed with respect to the ground truth trajectories obtained with the GPS-RTK measurements provided by the Swiftnav Duro device.

Metrics The selected VO-VIO approaches have been evaluated through three distinct metrics: (i) the pose estimation accuracy, (ii) the ratio of the estimated trajectory with respect to the ground truth trajectory, and (iii) the error rate with respect to the trajectory length. Since the algorithms rely on some non-deterministic operations, multiple runs on the same sequence might exhibit different results. Therefore, we perform a total of 5 experiments per algorithm and report the mean and standard deviation of the results for each metric.

1. **Pose Estimation Accuracy:** As mentioned in Section 3.2.3, the *evo* toolbox is employed to compute the pose accuracy of the selected approaches. In this work, we consider the Root Mean Square Error (RMSE) of the translational error ATE over all the (N) sequence poses:

$$RMSE_{ATE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} ATE_i^2} \tag{3.7}$$

   This value is computed after performing both scale and reference system alignment.

2. **Trajectory Duration Ratio:** Whenever a VO-VIO algorithm fails in estimating the trajectory, it may stop providing pose estimations. This behavior is strictly related to the authors' implementation and may vary between different algorithms. This aspect has to be taken into consideration since the APE value could turn out to be misleading. For this reason, an additional metric has been taken into account, which is the ratio between the time duration of the estimated trajectory Δt_{est} and the time duration of the ground truth trajectory Δt_{gt}. More specifically:

$$R_{ts} = \frac{\Delta t_{est}}{\Delta t_{gt}} = \frac{ts_{end,est} - ts_{start,est}}{ts_{end,gt} - ts_{start,gt}}, \tag{3.8}$$

   where, *e.g.*, $ts_{end,est}$ is the last timestamp of the resulted output estimation and $ts_{start,gt}$ is the first timestamp of the ground truth trajectory. This results in $R_{ts} \approx 1$ when the algorithm succeeds in estimating most of the trajectory. Conversely, $R_{ts} \to 0$ when the algorithm stops the estimation at the beginning of the sequence. Hence, the closer R_{ts} is to 1, the more this metric indicates a low failure rate.

3. **Error rate:** This metric takes into consideration the percentage of error

with respect to the length of the estimated sequence. In particular, this error ratio is computed as follows:

$$R_e = \frac{RMSE_{ATE} * 100}{R_{ts} * L}, \tag{3.9}$$

where L is the total length of the sequence, while $R_{ts} * L$ considers the portion of the sequence that has been predicted by the algorithm.

4. **Failure ratio:** Finally, failure ratio is taken into account since multiple runs are performed. An experiment is considered to be failed either if the algorithm fails and does not return any estimation, or if the estimated trajectory is less than 10% of the ground-truth one. The ratio is computed over 5 runs, as follows:

$$F = \frac{N}{5}, \tag{3.10}$$

where N is the number of failed runs (*i.e.*, if one runs out of 5 fails, the failure ratio is 0.2).

3.3.4 Results and discussion

In Table 3.7, the quantitative values of the proposed metrics are reported for the different experiments, while in Figure 3.13 an illustrative set of trajectory estimation examples is shown. The results show that LiDAR-based algorithms provide remarkable performance in predicting the trajectory of the sequences and, as expected, they outperform all the vision-based methods. In general, FLOAM has very high performance in terms of translational error, however, it tends to lose tracking. Contrarily, LeGO-LOAM, on the other hand, exhibits higher robustness and guarantees low estimation errors.

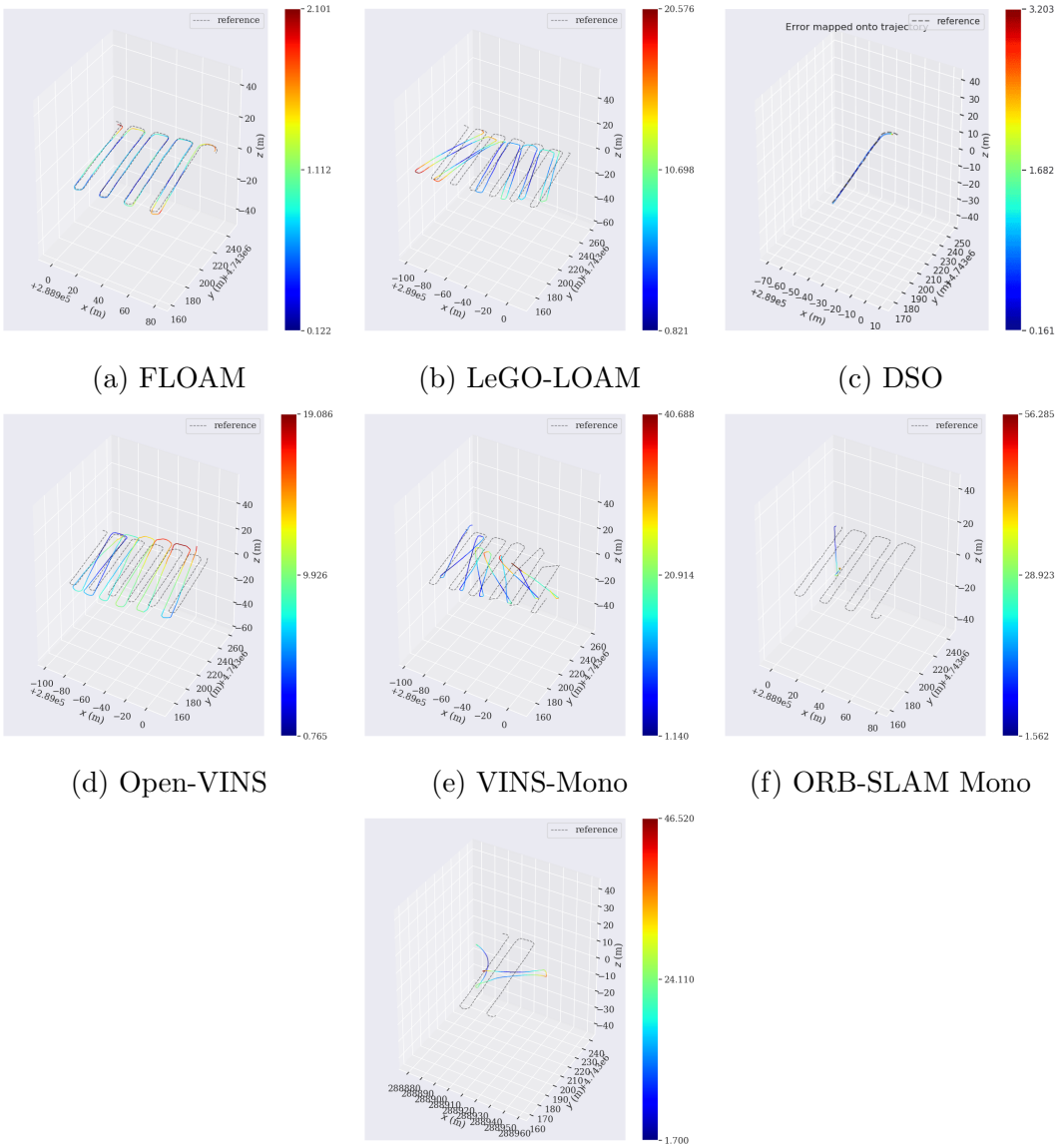
Among the vision-based methods, the best performing ones are VINS-Mono

and Open-VINS. However, Open-VINS has a very high standard deviation, which highlights that non-deterministic operations have a significant impact on its performance. ORB-SLAM Mono achieves worse results in terms of RMSE than its stereo version, although the latter has, in general, a lower trajectory duration ratio (*i.e.*, R_{ts}). Instead, DSO is the approach that experiences more failures.

It is worth mentioning that all the vision-based approaches suffer from hairpin turns: that is due to the accumulated drift error, leading to estimate a larger curvature. This is an additional aspect that makes the proposed dataset interesting, since it represents a challenging scenario to test the robustness of VO-VIO approaches.

OlvCs A								
	2021-09-30-12-05-36 (184m)				2021-09-30-12-17-38 (337m)			
Vision-based	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**
DSO	$\mathbf{3.94 \pm 0.12}$	0.99 ± 0.00	$\mathbf{2.15 \pm 0.07}$	0.4	$fail$			1.0
Open-VINS	6.61 ± 2.09	$\mathbf{1.00 \pm 0.00}$	3.59 ± 1.13	0.0	14.36 ± 1.41	$\mathbf{0.96 \pm 0.00}$	4.28 ± 0.41	0.0
VINS-Mono	9.91 ± 0.01	0.96 ± 0.00	5.61 ± 0.01	0.0	5.08 ± 0.03	$\mathbf{0.96 \pm 0.00}$	$\mathbf{1.57 \pm 0.01}$	0.0
ORB-SLAM Mono	23.39 ± 0.02	0.91 ± 0.02	13.9 ± 0.27	0.0	14.23 ± 0.10	0.93 ± 0.00	4.55 ± 0.03	0.0
ORB-SLAM Stereo	5.97 ± 4.34	0.57 ± 0.22	5.18 ± 1.39	0.0	$\mathbf{3.26 \pm 2.64}$	0.38 ± 0.17	2.13 ± 1.01	0.0
LiDAR-based	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**
FLOAM	$\mathbf{0.51 \pm 0.00}$	0.65 ± 0.00	0.43 ± 0.00	0.0	$\mathbf{0.49 \pm 0.01}$	0.58 ± 0.01	0.25 ± 0.01	0.0
LeGO-LOAM	0.56 ± 0.01	$\mathbf{0.99 \pm 0.01}$	$\mathbf{0.31 \pm 0.01}$	0.0	0.68 ± 0.01	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.20 \pm 0.00}$	0.0
OlvCs B								
	2021-10-13-09-56-36 (529m)				2021-10-13-10-22-41 (507m)			
Vision-based	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**
DSO	$fail$			1.0	$fail$			1.0
Open-VINS	34.26 ± 12.4	0.99 ± 0.00	6.48 ± 2.34	0.0	27.52 ± 17.3	$\mathbf{0.99 \pm 0.00}$	5.43 ± 3.42	0.0
VINS-Mono	$\mathbf{16.51 \pm 0.01}$	0.99 ± 0.01	$\mathbf{3.14 \pm 0.01}$	0.0	29.74 ± 0.01	0.98 ± 0.00	5.95 ± 0.01	0.0
ORB-SLAM Mono	59.65 ± 0.18	0.99 ± 0.00	11.3 ± 0.03	0.0	55.68 ± 0.31	0.98 ± 0.00	11.2 ± 0.08	0.0
ORB-SLAM Stereo	24.27 ± 1.20	$\mathbf{1.00 \pm 0.00}$	4.59 ± 0.22	0.2	$\mathbf{16.68 \pm 7.10}$	0.75 ± 0.13	$\mathbf{4.12 \pm 1.54}$	0.0
LiDAR-based	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**
FLOAM	$\mathbf{0.72 \pm 0.03}$	0.58 ± 0.00	0.23 ± 0.01	0.0	$\mathbf{0.79 \pm 0.00}$	0.56 ± 0.01	0.27 ± 0.01	0.0
LeGO-LOAM	0.85 ± 0.03	$\mathbf{0.99 \pm 0.00}$	$\mathbf{0.16 \pm 0.01}$	0.0	0.73 ± 0.01	$\mathbf{1.00 \pm 0.00}$	$\mathbf{0.14 \pm 0.01}$	0.0
Vynrd A								
	2021-09-03-17-43-40 (443m)				2021-09-03-18-25-40 (772m)			
Vision-based	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**
DSO	11.86 ± 6.28	0.50 ± 0.13	4.89 ± 1.97	0.0	53.94 ± 0.12	$\mathbf{0.99 \pm 0.00}$	7.06 ± 0.01	0.6
Open-VINS	$\mathbf{8.07 \pm 5.41}$	$\mathbf{0.99 \pm 0.00}$	$\mathbf{1.82 \pm 1.22}$	0.0	32.65 ± 19.3	0.90 ± 0.00	4.26 ± 2.52	0.0
VINS-Mono	16.18 ± 1.11	0.98 ± 0.02	3.69 ± 0.17	0.0	$\mathbf{19.71 \pm 1.07}$	0.97 ± 0.01	$\mathbf{2.61 \pm 0.11}$	0.0
ORB-SLAM Mono	33.29 ± 9.77	0.81 ± 0.23	9.22 ± 0.12	0.0	53.31 ± 0.15	$\mathbf{0.99 \pm 0.00}$	6.97 ± 0.02	0.2
ORB-SLAM Stereo	11.69 ± 1.70	0.48 ± 0.10	5.70 ± 1.46	0.0	51.44 ± 3.10	0.77 ± 0.14	8.77 ± 1.06	0.2
LiDAR-based	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**
FLOAM	$\mathbf{0.39 \pm 0.00}$	0.99 ± 0.01	$\mathbf{0.09 \pm 0.00}$	0.0	$\mathbf{1.02 \pm 0.01}$	0.99 ± 0.00	$\mathbf{0.13 \pm 0.00}$	0.0
LeGO-LOAM	3.08 ± 0.05	$\mathbf{1.00 \pm 0.01}$	0.69 ± 0.01	0.0	7.50 ± 0.71	$\mathbf{1.00 \pm 0.00}$	0.97 ± 0.01	0.0
Vynrd B								
	2021-08-04-11-28-41 (308m)				2021-09-01-12-25-09 (1044m)			
Vision-based	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**
DSO	14.54 ± 8.49	0.74 ± 0.25	5.74 ± 1.73	0.2	$\mathbf{0.81 \pm 0.00}$	0.11 ± 0.00	$\mathbf{0.77 \pm 0.00}$	0.8
Open-VINS	19.08 ± 8.5	$\mathbf{1.00 \pm 0.00}$	6.17 ± 2.75	0.0	9.74 ± 1.54	$\mathbf{0.99 \pm 0.00}$	0.94 ± 0.14	0.0
VINS-Mono	13.50 ± 0.84	0.87 ± 0.01	5.02 ± 0.38	0.0	15.41 ± 0.01	0.96 ± 0.00	1.53 ± 0.00	0.0
ORB-SLAM Mono	23.18 ± 0.04	$\mathbf{1.00 \pm 0.00}$	7.51 ± 0.01	0.0	27.41 ± 1.79	0.48 ± 0.09	5.58 ± 1.00	0.0
ORB-SLAM Stereo	$\mathbf{9.95 \pm 0.45}$	$\mathbf{1.00 \pm 0.00}$	$\mathbf{3.23 \pm 0.15}$	0.0	20.56 ± 0.10	0.25 ± 0.00	7.87 ± 0.06	0.2
LiDAR-based	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**	**RMSE** (m)	$\mathbf{R}_{ts}$	$\mathbf{R}_e$ (%)	**F**
FLOAM	$\mathbf{0.72 \pm 0.00}$	0.94 ± 0.00	$\mathbf{0.25 \pm 0.00}$	0.0	$\mathbf{0.80 \pm 0.01}$	0.66 ± 0.00	$\mathbf{0.11 \pm 0.00}$	0.0
LeGO-LOAM	1.74 ± 0.18	$\mathbf{1.00 \pm 0.01}$	0.56 ± 0.05	0.0	7.91 ± 0.77	$\mathbf{1.00 \pm 0.00}$	0.75 ± 0.07	0.0

Table 3.7: Quantitative results on the selected sequences. The metrics are detailed in Section 3.3.3. Mean and standard deviation over 5 runs are reported, and the best results for each metric among vision-based and LiDAR-based approaches are highlighted in bold. If two approaches achieve similar average errors, to select the best one we consider that with the smaller standard deviation.



(a) FLOAM (b) LeGO-LOAM (c) DSO

(d) Open-VINS (e) VINS-Mono (f) ORB-SLAM Mono

(g) ORB-SLAM Stereo

Figure 3.13: Trajectory estimation on Vynrd B (2021-09-01-12-25-09). Some approaches may lose tracking or fail before the end of the trajectory. In this cases, the execution of the algorithm is stopped and estimated poses are no longer provided. In these cases, the corresponding plots show shorter trajectories since only the estimated poses (and the associated ground truth ones) before the algorithm failure are depicted, discarding the the remaining ground truth trajectory points.

Chapter 4

Topological Localization

4.1 Overview

Vision-based localization has attracted the interest of many research studies. The most popular solutions proposed in the last decades for this problem are grounded on the Visual Odometry (VO) and the Simultaneous Localization and Mapping (SLAM) frameworks [7; 92]. Their main product is a pose (and a map for SLAM systems) expressing the location of the robot relative to a metric reference system. Estimating precise metric transformations between frames is, however, often hindered by many factors. The performance of VO and SLAM approaches tend to degrade under non-ideal conditions, *i.e.*, texture-poor scenes, low amount of illumination, and fast motion dynamics, the latter being typical in MAV applications. In addition, their success is tightly linked to the accuracy of camera calibration procedures.

Most of the challenges arise from the requirement of precise metric positioning, which, in many applications, might be an unnecessary over-complication. If we shift the paradigm and take inspiration from humans and animals, we notice that they do not rely on metric positioning systems. Instead, they take advan-

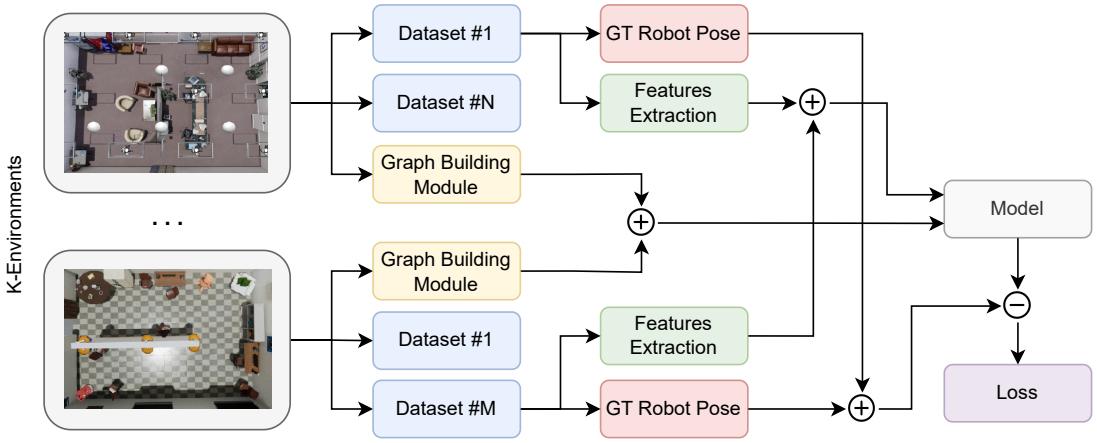


Figure 4.1: Overview of the topological localization training process. First, a graph is built for each environment. Afterward, the robot navigates through the environment to gather the training data, *i.e.*, visual observations associated with the ground-truth node. Finally, the model is trained to estimate the closest graph node to the current robot position.

tage of semantic information and high-level relationships between objects, scenes, and shapes to estimate their position in the environment [21]. Following these intuitions, visual place recognition and topological localization solutions have recently emerged as alternative paradigms to achieve robust localization even in non-ideal scenarios [22; 23; 24; 87].

Topological localization aims to estimate the position of the robot with respect to a semantic map and can be seen as either a stand-alone module when precise metric positioning is not required, or it can be integrated into metric SLAM systems (*e.g.*, loop closure or drift correction), or in aid of different tasks (*e.g.*, exploration or map building). The peculiarity of this paradigm lies behind the use of a topological map, *i.e.*, a sparse representation of the scene consisting of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. A node $v \in \mathcal{V}$ represents a salient location in the space, while an edge $e \in \mathcal{E}$ is a connection between two positions due to, *e.g.*, mutual visibility or reachability.

Besides the map, the localization framework requires the current observation

of the robot, *e.g.*, in visual-based approaches, it is represented by the image collected by the robot's front-view camera. Thus, for the training process, every visual observation collected during the robot navigation needs to be associated with the corresponding ground-truth node, *i.e.*, the closest position of that observation in the graph. Then, the message-passing framework of the Graph Neural Networks (GNNs) can be exploited to operate with graph-structured data (*e.g.*, the topological maps) and propagate information among neighbor nodes. The overview of the training process proposed in our works is shown in Figure 4.1. In the following Sections, two approaches will be discussed for topological localization for ground and aerial platforms, respectively. The latter represents a more challenging use case since an MAV is free to move in the 3D space, which involves a more complex motion dynamic and increases the number of locations to be modeled in the semantic map and, thus, the chance of perceptual aliasing. Indeed, aliasing is a phenomenon where different places have a visually similar appearance and is one of the main causes of the failure in VPR problems. Hence, it requires a more sophisticated architecture to address the additional challenges.

4.2 Preliminaries

The strength of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) lies in the capability of exploiting the interconnections between input data, which are spatial connections for CNNs and temporal connections for RNNs. In particular, an image can be represented as a regular grid in Euclidean space (*i.e.*, pixels are regularly interconnected). Similarly, samples in a temporal sequence are interconnected through consecutive timesteps. But there is an increasing number of applications where data are represented as graphs with complex relationships and non-Euclidean properties, for example in image under-

standing [113], social networks [114; 115] and text analysis [116]. The complexity of graph data is challenging for machine learning approaches: a graph is an irregular data structure with a variable number of nodes, and each node may have a different number of neighbors. Thus, easy operations in the image domain can be difficult to apply in the graph domain. Historically, the first approaches applied to graph-structured data were used in a pre-processing step to simplify the graph information, and they were not part of the training process. Contrarily, Graph Neural Network (GNN), a recent novel technique, is an end-to-end machine learning model that learns a representation of graph-structured data and fits a predictive model on them.

GNNs [117] create an embedding of the graph nodes in a low-dimensional space and the training process learns the structural properties of the graph. The key insight is the propagation of node information among its neighbors through message-passing. When the task is classification or regression at a node level, this embedding can be used directly. Otherwise, if a graph or a sub-graph level classification or prediction is required, pooling techniques must be used to obtain a global representation.

4.2.1 Notation

A graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{A})$, where $\mathcal{V}$ is the set of vertices or nodes, $\mathcal{E}$ is the set of edges and $\mathcal{A}$ is the adjacency matrix. Let $v_{\mathrm{i}} \in \mathcal{V}$ denote a node and $e_{\mathrm{i,j}} = (v_{\mathrm{i}}, v_{\mathrm{j}}) \in \mathcal{E}$ denote an edge pointing from v_{i} to v_{j}. The adjacency matrix $\mathcal{A}$ is a $N \times N$ matrix with $N = |\mathcal{V}|$ and for each node it specifies connection (which can be weighted) with others:

$$\begin{cases} \mathcal{A}_{\mathrm{i,j}} = w_{\mathrm{i,j}} > 0 & \text{if } e_{\mathrm{i,j}} \in \mathcal{E} \\ \mathcal{A}_{\mathrm{i,j}} = w_{\mathrm{i,j}} = 0 & \text{if } e_{\mathrm{i,j}} \notin \mathcal{E} \end{cases}$$

The neighborhood of a node v_i is defined as $\mathcal{N}_i = \{v_j \in \mathcal{V} | (v_i, v_j) \in \mathcal{E}\}$. The degree of a node is the number of edges connected to it, formally defined as $degree(v_\mathrm{i}) = \sum \mathcal{A}_{\mathrm{i},:}$. A graph may have node attributes $\mathbf{X}_\mathcal{V}$, where $\mathbf{X}_\mathcal{V} \in \mathbb{R}^{N \times F_v}$ is a node feature matrix with $\mathbf{x}(v_i) \in \mathbb{R}^{F_v}$ representing the feature vector of node v_i. A graph may also have edge attributes $\mathbf{X}^e$, where $\mathbf{X}^e \in \mathbb{R}^{E \times F_e}$ is an edge feature matrix with $\mathbf{x}^e(v_i, v_j) \in \mathbb{R}^{F_e}$ representing the feature vector of an edge $e_{i,j} = (v_i, v_j)$.

A directed graph is a graph where edge directions are explicitly specified, *i.e.*, all edges are directed from one node to another. For a directed graph $\mathcal{A}_{i,j} \neq \mathcal{A}_{j,i}$, while for an undirected graph $\mathcal{A}_{i,j} = \mathcal{A}_{j,i}$. Thus, an undirected graph is a graph where edges are bidirectional.

4.2.2 Graph Neural Network

The first idea of GNN was presented by Gori et al. in 2005 [118] and then elaborated by Scarselli et al. in 2009 [119]. GNNs use the graph structure and node features to learn a representation vector of a node or the entire graph. Iteratively, the representation of a node is updated by aggregating representations of its neighbors. The key idea is to generate node embeddings based on local neighborhoods by using learnable layers. After k iterations of aggregation, this representation captures the structural information within its k-hop neighborhood. Formally, the k-th layer of a GNN is defined as:

$$\mathbf{h}_v^k = \mathrm{UPDATE}^{(k)} \left(\mathbf{h}_v^{k-1}, \mathrm{AGGREGATE}^{(k)} \left(\{ \mathbf{h}_u^{k-1}, \forall u \in \mathcal{N}_v \} \right) \right) \tag{4.1}$$

where $\mathbf{h}_v^k$ is the feature vector of node v at the k-th iteration and $\mathcal{N}_v$ is the set of nodes adjacent to v. The initialization step consists of setting $\mathbf{h}_v^0 = \mathbf{X}_\mathcal{V}$. The main difference is the approach to aggregate information across the layers, for

instance, the first approach uses average neighbor messages:

$$\mathbf{h}_v^k = \sigma\left(W_k \sum_{u \in \mathcal{N}_v} \frac{\mathbf{h}_u^{k-1}}{|\mathcal{N}_v|} + B_k \mathbf{h}_v^{k-1}\right), \forall k > 0 \tag{4.2}$$

where:

- $\mathbf{h}_v^0 = \mathbf{X}_v$: the initial embeddings are node features;
- $\mathbf{h}_v^k$: k-th layer embedding of v;
- σ: non-linearity (*e.g.*, ReLU or tanh);
- W_k, B_k: shared parameters;
- $\sum_{u \in \mathcal{N}_v} \frac{\mathbf{h}_u^{k-1}}{|\mathcal{N}_v|}$: average of neighbor's previous layer embeddings;
- $\mathbf{h}_v^{k-1}$: embeddings vector of the previous layer of v.

Two main properties of this approach can be outlined as:

1. Parameter sharing: aggregation parameters are shared among all nodes.
2. Inductive capability: generalization capabilities for new graphs, even with a different number of nodes.

This iterative process is computationally intensive, so modern implementations try to speed it up, for example by limiting the number of iterations or by using random walks for processing the neighborhood.

As already mentioned, graph processing tasks can be categorized as node-level (node classification or regression) or graph-level (graph classification or regression). For the former, each node $v \in \mathcal{V}$ has a target value y_v and the goal is to learn a representation vector $\mathbf{h}_v$ of v so that its label can be predicted as $y_v = f(\mathbf{h}_v)$. In this case, the node representation $\mathbf{h}_v^K$ of the final iteration is used

for the prediction. For the latter, the entire graph $\mathcal{G}$ is associated with a label $y_{\mathcal{G}}$, and the goal is to learn a graph representation $\mathbf{h}_{\mathcal{G}}$ to predict the target value as $y_{\mathcal{G}} = g(\mathbf{h}_{\mathcal{G}})$. In this case, a READOUT function to aggregate node features from the final iteration is needed:

$$\mathbf{h}_{\mathcal{G}} = \text{READOUT}(\{\mathbf{h}_v^K | v \in \mathcal{V}\})$$

This READOUT function can be a simple sum or a more sophisticated pooling function, while choosing the AGGREGATE and UPDATE functions is crucial.

4.2.3 Graph Convolutional Neural Network

Graph Convolutional Networks (GCNs) generalize the operation of convolution from grid data to graph data, *i.e.*, from perfectly regular graphs (*i.e.*, images) to irregular ones. They differ from graph embedding methods since, instead of encoding the graph to a lower dimension, convolutional operations are performed on the entire input graph itself. Images can be seen as regular graphs, hence the idea is to generalize the locality and invariance properties for generic graphs. As illustrated in Figure 4.2, a graph convolution can be generalized from a 2D convolution. In 2D convolution, each pixel may be seen as a node and its neighborhood is determined by the filter size. The 2D convolution takes the weighted average of pixel values of the node along with its neighbors. The graph convolution takes

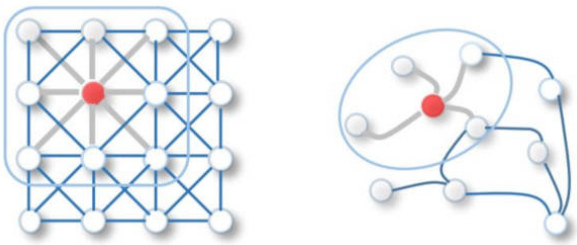


Figure 4.2: 2D convolution vs graph convolution

.

the weighted average value of the node features along with its neighbors, but different from image data, the neighbors of a node are unordered and size variable. Indeed, graphs have an irregular structure, and this variability (*e.g.*, node degree, proximity, and neighborhood structure) provides important information about the data. The main idea is to generate a node representation by aggregating its own features x_v and its neighbors' features x_u, where $u \in \mathcal{N}_v$. In Kipf et al. [120], the AGGREGATION function consists of:

$$\mathbf{h}_v^k = \sigma\left(W_k \sum_{u \in \mathcal{N}_v \cup v} \frac{\mathbf{h}_u^{k-1}}{\sqrt{|\mathcal{N}(u)||\mathcal{N}_v|}}\right), \tag{4.3}$$

where:

- The same matrix W_k is used for both self and neighbor embeddings.
- Per-neighbor normalization instead of a single average.

4.3 Ground platforms

The core ability of an autonomous mobile robot lies in sensing and perceiving the environment, enabling it to gather information, interpret it, and subsequently interact with its surroundings. Ground robots are widely employed in a variety of applications, *e.g.*, in warehouses, logistical companies, agriculture businesses, and healthcare institutions, to improve operational efficiency, enhance speed, ensure precision, and increase safety. Localization is surely the main component of an autonomous robot. In the past, robotic localization has been mostly tackled with Visual Odometry (VO) or Simultaneous Localization and Mapping (SLAM) algorithms. Despite the remarkable results achieved in metric positioning accuracy [92], they have some shortcomings such as the amount of memory required (especially in large environments) to maintain a global map and the lack of robustness

in non-ideal environments (*e.g.*, texture-less scene, low illumination, complex motion dynamics).

Contrarily, humans rely on a topological description of the space to localize themselves and do not need precise metric positioning. The main source of information for human localization is given by the objects within the environment, which can provide coarse spatial information, visual cues and describe the high-level characteristics of the scene. Inspired by the capabilities of humans, we present a novel and lightweight framework (from a memory consumption point of view) to perform topological localization and to overcome these limitations [121].

4.3.1 Contributions

The main contributions of the proposed study [121] are the following:

- In contrast to other topological works, whose graph maps only associate nodes to positions, we build a topological map that jointly models the overall scene and the objects as graph nodes.
- The proposed framework does not leverage metric information and is based on a lightweight map, which considers only salient points in the environment. This approach is characterized by a low memory requirement with respect to traditional VO algorithms.
- As a result of the training of our network via random environments, this model achieves good results in terms of robustness in unseen places.

4.3.2 Proposed approach

In this study, we propose a novel algorithm, named Graph Object-based Localization Network (GOLN), to localize a robot in an environment exploiting a

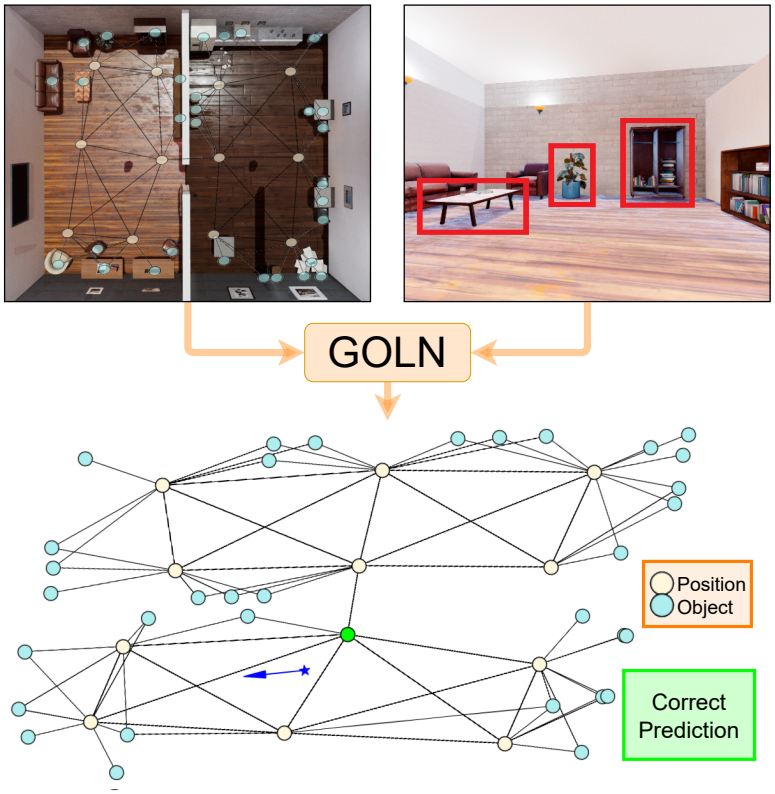


Figure 4.3: GOLN overview. Our framework takes in input a graph map (*top-left* image) and the current observation captured by the robot front-view camera (*top-right* image). In addition, it considers the objects in the surroundings (red squared). It returns the estimation of the closest position graph node to the current robot location.

previously built topological object-based map. Therefore, we describe the world with this topological map, structured as a graph. We employ a Graph Neural Network (GNN) model, based on [122], which operates with graph-structured data (*e.g.*, a topological map) by propagating information through message-passing among neighbor nodes. In order to localize the robot, GNN processes the graph map and the current observation captured by the front-looking camera describing the robot point of view. For the training, several simulated environments are generated, and each environment is associated with one graph map. In addition, the proposed framework requires a number of training data, which have been

collected by driving the robot and consist of a sequence of images and the corresponding ground-truth nodes, *i.e.*, the closest graph node to the current position of the robot. Then, the framework is trained in a supervised manner to estimate the closest position node to the current robot position. In the inference phase, the framework only requires the topological map and the observations provided by the robot, and returns in output the estimation of the current position node. The overall architecture is shown in Figure 4.1 and is detailed below.

4.3.2.1 Map building

Object-based topological map The topological map is a sparse representation of an environment that focuses only on salient points. In this case, the map is built as an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, consisting of a set of nodes $\mathcal{V}$ and a set of bidirectional edges $\mathcal{E}$. Inspired by the literature [123; 124], we explicitly consider in the map the relationships (visual and spatial) existing between objects and locations. In particular, a node $v \in \mathcal{V}$ may be a position node or an object node, therefore an edge $e \in \mathcal{E}$ is a connection between two close positions or between a position and an object. In particular, $e_{ij} = 1$ if the position or the object j is in the range of visibility of the position i, otherwise $e_{ij} = 0$, *i.e.*, no edge exists between node i and j. If the same object is seen from different positions, then the associated object node is connected with the corresponding position nodes. Hence, the number of objects connected to each position node is variable and depends on the structure of the environment. Moreover, two position nodes are connected together through an edge in the following cases: (i) if they correspond to consecutive positions during the robot exploration for the map building, or (ii) if they are close enough in space and visible to each other, *e.g.*, there are no walls among them. An example of a graph map is shown in Figure 4.3.

Graph construction procedure In this Section, we describe the graph building phase. The robot is manually driven by using a joystick or a keyboard, and it collects a new position node i and a variable number of object nodes every Δt_i seconds. In particular:

$$\Delta t_i = max(\Delta p_{i-1}, \Delta c),$$

where Δp_{i-1} is the processing time required by the computation of the last position node and its associated object nodes, while Δc is the minimum time between two consecutive position nodes.

To address the viewpoint variation challenge, *i.e.*, the same scene might have a completely different appearance if observed from a different perspective, we capture images by rotating the camera with respect to the yaw angle (*i.e.*, the heading) four times by 90 degrees in order to have a complete view of the scene (*i.e.*, a 360°-view). Each image is associated with a node in the topological map and, therefore, we collect four overlapping nodes that describe the same location. Additionally, for each position node v_i this procedure stores a variable number of object nodes, depending on how many objects are seen from the i-th position. Specifically, we select the objects in front of the agent with a distance lower than a given threshold (*i.e.*, 600 cm), and we check that they are not hidden by obstacles.

4.3.2.2 Localization procedure

The proposed framework aims to estimate the current position node of the ground robot given the pre-built topological map and the current observation. First, visual features are extracted with a CNN from the graph nodes and the current observation. Then, a GN model based on [122] processes the data and returns the updated graph as output. Finally, a classification layer estimates the current position node of the robot. The architecture is shown in Figure 4.4 and detailed

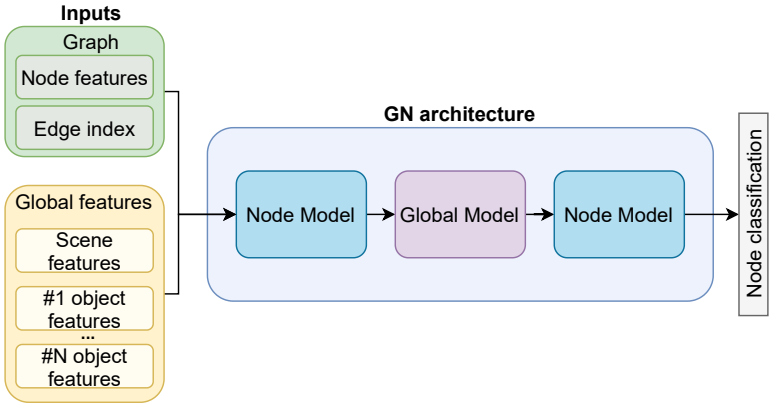


Figure 4.4: Architecture of the Graph Network, which updates the input features: the graph (*i.e.*, the node features and the edges) and the global features (*i.e.*, the scene features and the objects' features). The overall network consists of two Node Models and one Global Model. A classification layer estimates the current position node of the robot.

below.

Features Extraction Module The node images and the current observation of the robot are processed by a pre-trained 34-layer ResNet [72] to extract the visual features. The feature vector of each node $v_i \in \mathcal{V}$ is defined as $\mathbf{x}(v_i) = [p_i, \mathbf{b}_i] \in \mathbb{R}^{513}$, where:

$$p_i = \begin{cases} 1, & \text{if } v_i \text{ is a position node} \\ 0, & \text{if } v_i \text{ is an object node} \end{cases} \tag{4.4}$$

$$\mathbf{b}_i = [b_{i1}, ..., b_{i512}] \text{ is the ResNet embeddings vector} \tag{4.5}$$

Therefore, the overall node features matrix is indicated with $\mathbf{X}_V \in \mathbb{R}^{N \times 513}$ and

includes the feature vectors of all the N nodes in the graph:

$$\mathbf{X}_V = \mathbf{x}(v_i), \forall i = 1, ..., N \tag{4.6}$$

Global features $\mathbf{u}(\mathcal{I}_k) \in \mathbb{R}^{M+1\times 513}$ are instead extracted from the current observation $\mathcal{I}_k$, and described as follows:

$$\mathbf{u}(\mathcal{I}_k) = [\mathbf{u}_{kp}, \mathbf{u}_{ko_1}, ..., \mathbf{u}_{ko_M}], \tag{4.7}$$

where:

$$\mathbf{u}_{kp} = [1, u_{kp1}, ..., u_{kp512}] \tag{4.8}$$

$$\mathbf{u}_{ko_j} = [0, u_{ko_j1}, ..., u_{ko_j512}] \tag{4.9}$$

Both $\mathbf{u}_{kp}$ and $\mathbf{u}_{ko_j}$ are 513-dimensional vectors and follow the structure of $\mathbf{x}(v_i)$. In particular, $\mathbf{u}_{kp}$ describes the k-th scene captured by the robot during the navigation, while $\mathbf{u}_{ko_j}$ and M are the j-th object and the number of objects visible in the same scene, respectively.

Graph Processing Module The node features $\mathbf{X}_V$ and the global features $\mathbf{u}(\mathcal{I}_k)$ are processed and combined through a GNN based on [122]. The network consists of a combination of computational blocks, as shown in Figure 4.4, namely Node Model and Global Model.

The former (see Algorithm 1) updates each node's features by combining position nodes, object nodes, and the global features of the current observation. The latter (see Algorithm 2) integrates the updated node features with the global features. The Node Model uses the following functions:

- $\Phi^v : \mathbb{R}^{513} \rightarrow \mathbb{R}^{256}$ implements the message-passing framework and combines

Algorithm 1 Node Model

function $NodeModel(\mathbf{X}_\mathcal{V}, \mathcal{E}, \mathbf{u}(\mathcal{I}_k))$
  for $i \in \{1...N\}$
    $\mathcal{N}_i = \mathbf{x}(v_z),\ \forall z : \mathbf{e}_{iz} \in \mathcal{E}$
    $\tilde{\mathbf{X}}_\mathcal{V} \leftarrow \Phi^v(\mathbf{X}_\mathcal{V}, \mathcal{N}_i)$
  end for
  $\tilde{\mathbf{u}}(\mathcal{I}_k) \leftarrow \rho^{u\rightarrow v}(\mathbf{u}(\mathcal{I}_k))$
  $\mathbf{X}'_\mathcal{V} \rightarrow \Phi^{v,u}(\mathbf{X}_\mathcal{V}, \tilde{\mathbf{X}}_\mathcal{V}, \tilde{\mathbf{u}}(\mathcal{I}_k))$
  return $\mathbf{X}'_\mathcal{V}$
end function

Algorithm 2 Global Model

function $GlobalModel(\mathbf{X}'_\mathcal{V}, \mathbf{u}(\mathcal{I}_k))$
  $\hat{\mathbf{X}}'_\mathcal{V} \leftarrow \rho^{v\rightarrow u}(\mathbf{X}'_\mathcal{V})$
  $\mathbf{u}'(\mathcal{I}_k) \leftarrow \Phi^{u,v}(\hat{\mathbf{X}}'_\mathcal{V}, \mathbf{u}(\mathcal{I}_k))$
  return $\mathbf{u}'(\mathcal{I}_k)$
end function

the information from both position and object nodes. In particular, the features of each node $\mathbf{x}(v_i)$ and its neighbors $\mathcal{N}_i$ are separately fed into a Multi-Layer Perceptron (MLP) to create the associated message and then averaged into a 256-dimensional vector to update $\mathbf{x}(v_i)$.

- $\rho^{u\rightarrow v} : \mathbb{R}^{M+1\times 513} \rightarrow \mathbb{R}^{513}$ updates the global features $\mathbf{u}(\mathcal{I}_k)$ by averaging the scene embeddings $\mathbf{u}_{kp}$ and the M objects embeddings $\mathbf{u}_{ko_j}, \forall j = \{1, ..., M\}$.

- $\Phi^{v,u} : \mathbb{R}^{1282} \rightarrow \mathbb{R}^{513}$ is an MLP network with the same structure as Φ^v that processes the concatenation of the node features matrix $\mathbf{X}_V$ before message passing, the combined node feature matrix $\tilde{\mathbf{X}}_V$, and the aggregated global features $\tilde{\mathbf{u}}(\mathcal{I}_k)$.

The Global Model updates the global features as follows:

- $\rho^{v\rightarrow u} : \mathbb{R}^{N\times 513} \rightarrow \mathbb{R}^{M+1\times 513}$ first averages the node features $\mathbf{X'}_V \in \mathbb{R}^{N\times 513}$ in $\overline{\mathbf{X}}'_V \in \mathbb{R}^{513}$ and then expands it by replicating $M+1$ times, resulting in $\hat{\mathbf{X}}'_V \in \mathbb{R}^{M+1\times 513}$.

- $\Phi^{u,v} : \mathcal{R}^{1026} \rightarrow \mathcal{R}^{513}$ is an MLP network with the same structure as Φ^{v} and $\Phi^{v,u}$ that combines the averaged node features $\hat{\mathbf{X}}'_V$ and the global features $\mathbf{u}(\mathcal{I}_k)$.

Finally, the updated node $\mathbf{X}'_V$ and global $\mathbf{u}'(\mathcal{I}_k)$ features are further processed by the Node Model to obtain the final node features matrix $\mathbf{X}''_V$. This is fed into the node prediction module to estimate the closest position node to the current location of the robot.

Node Prediction Module A probability value for each node $s_k(v_i) \in \mathbb{R}$ is estimated with a softmax classifier as follows:

$$\mathbf{s}_k = \text{Softmax}(\mathbf{W}\mathbf{Z}_k + \mathbf{b}) \in \mathbb{R}^N, \tag{4.10}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the weight matrix and the bias of a fully-connected layer, and N is the number of nodes in the graph. The node associated with the highest probability value is estimated as the closest to the current robot position.

4.3.2.3 Implementation details

Data collection for model training The experiments have been performed in simulated indoor environments generated in Unreal Engine (UE4) with a simulated ground robot equipped with a monocular camera. These rooms have been randomized with the Dungeon Architect[1], a procedural scenario generation system that allows to easily randomize environments. Given a set of objects and textures, the plugin first generates the simulated environment by randomizing the structure of the rooms (*e.g.*, the walls), the textures, and the position of the objects. Then, the robot navigates through the environment, collecting images

[1] `https://dungeonarchitect.dev/`

from its front-looking camera and associating them with the ground truth nodes. From each position, visible objects are taken into account to extract the global features. As for the graph building procedure, we check that objects are not hidden by obstacles, *i.e.*, walls. Indeed, UE4 allows to access different characteristics of the objects in the environment, including the 3D bounding box, and this helps the identification of the objects in the space. We first convert the objects' 3D bounding boxes into 2D bounding boxes and evaluate if they are actually visible in the agent FOV. Given the 3D bounding boxes of the walls, we exploit *plane-line* geometrical relationships to check if the object is in the line of sight of the agent or is hidden. Specifically, the robot and the object can be seen as the endpoints of a line, the wall can be described as a plane, and the object is considered to be visible if no *plane-line* intersections exist. For each generated scene, several trajectories are performed to collect the data in order to improve the generalization capabilities over different viewpoints and positions. In total, 47 graphs and 124 datasets have been collected to train the proposed framework.

Implementation and training details The robot front-view camera collects 512x512 images. Features are extracted by a ResNet-34 pre-trained on ImageNet dataset [91]. The structure of the GNN, described in Section 4.3.2.2, is implemented through the PyTorch MetaLayer[1]. The inputs for the network are the node features with shape (B, N, F), the edges with shape $(2, E)$, and the global features with shape $(B, M+1, F)$, where B is the batch size, N the number of nodes in a graph, F the number of features (*i.e.*, 513), E the number of edges and M the number of visible objects. The Node Model consists of an aggregation of global features and an update of node features. In particular, the MLPs that implement Φ^v and $\Phi^{v,u}$ of the Node Model include a hidden layer with 513 nodes

[1] `https://pytorch-geometric.readthedocs.io/en/latest/modules/nn.html#torch_geometric.nn.meta.MetaLayer`

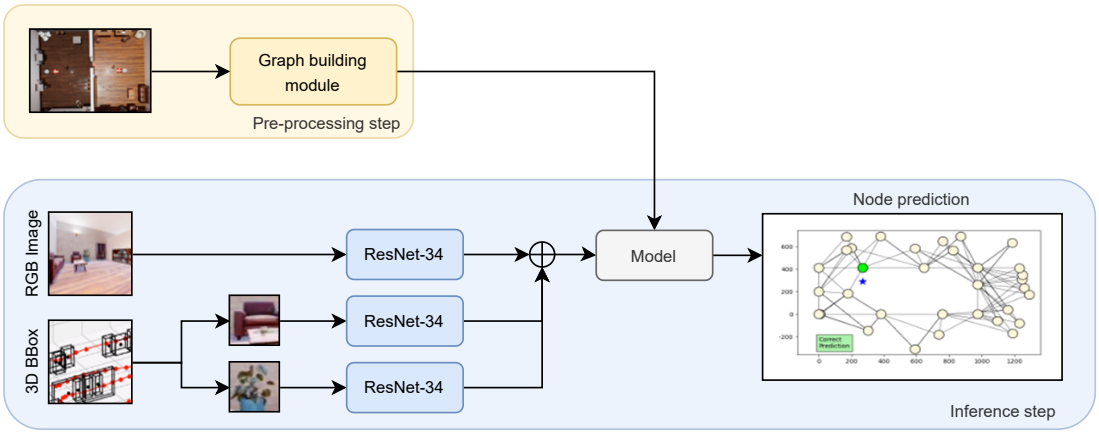


Figure 4.5: Inference process for the proposed topological localization approach. A pre-processing step is employed to build a graph representation of the environment. Then, a Graph Neural Network (GNN) processes the pre-built topological map and the current observation to predict the closest graph node.

followed by a ReLU activation and a dropout layer with dropout probability set to 0.2, and an output linear layer. The latter has 256 and 513 nodes in the case of Φ^v and $\Phi^{v,u}$, respectively. The same MLP structure is used for the function $\Phi^{u,v}$ of the Global Model, with both the hidden and output layer composed of 513 nodes. The best results are achieved with a batch size of 10, a weight decay of 10^{-4}, a dropout of 0.4, and a learning rate of 10^{-6}, which is decreased by a factor of 0.7 every 20 epochs.

Inference phase The test process is shown in Figure 4.5. The GNN model processes a pre-built topological map and the current observation of the robot. The node features $\mathbf{X}_V$ can be computed offline to reduce the computation time. Contrarily, the global features $\mathbf{u}(\mathcal{I}_k)$ need to be computed online at inference time. In addition, in this work, we focus on memory usage. Different from VO approaches, whose memory requirements tend to grow with the increase of the path length, a topological localization approach allocates a fixed amount of memory, *i.e.*, required for storing the graph map and for the inference step.

Specifically, this results in a significant memory consumption saving, mainly due to the low amount of memory required by GNNs to process graph-structured data.

4.3.3 Experimental results

Evaluation Environments To evaluate the performance of the proposed approach, we generate 11 additional scenes with different characteristics with respect to the training ones. The aforementioned evaluation graphs and the datasets are split into:

1. *Training-graphs Validation set*: A set of trajectories different from the training ones, but collected on 5 graphs that were also used during training.
2. *Training-scenarios Validation set*: A set of trajectories collected in 3 graphs, built from environments with a similar appearance to the training ones.
3. *Test set*: A set of trajectories collected in 3 graphs, built from scenes significantly different from the training ones.

The topological maps are generated as detailed in Section 4.3.2.1. Some examples are shown in Figure 4.6.

Baselines We select the monocular vision-only version of ORB-SLAM3 (OS3) [37] as the baseline since it is one of the most popular and recent VO algorithms. It has three main parallel threads: (i) the tracking thread to localize the camera with every frame by finding feature matches to the local map, minimizing the reprojection error via motion-only BA and detecting new keyframes; (ii) the local mapping thread to manage the local map and optimize it, inserting and deleting keyframes and performing local BA; (iii) the loop closing thread to detect large loops and correct the accumulated drift by performing a pose-graph optimization.

Figure 4.6: Example of indoor environments generated for the training, validation and test sets. The first row shows the *top-view* of the rooms, while the second row the robot *front-view* camera.

This thread launches a fourth thread to perform full BA after the pose-graph optimization, to compute the optimal structure and motion solution. For the comparison, we adapted OS3 to solve a topological localization task, *i.e.*, each metric position is associated with the closest position node in the graph. In detail, firstly, the output trajectory is aligned with the Umeyama [125] matrix and with the scale correction. Then, the association between OS3 metrics positions and graph nodes is performed, but only if the distance between the metric position of the graph node and the OS3 output position is lower than a given threshold (approximately 5 m). We use this threshold in order to take into account when the VO algorithm tends to diverge.

Metrics We consider different metrics to evaluate the performance of our approach and compare it with the *state-of-the-art* VO approach. Firstly, accuracy is taken into account to evaluate whether the robot estimates correctly its position in the space. An estimation is considered to be correct if the estimated node and

Training-scenario				
	Accuracy	Accuracy 1-hop	Accuracy 2-hop	Memory
T-OS3	34.4%	**80.9**%	**89.0**%	929.7 MB
GOLN	**50.0**%	75.0%	85.0%	**116.8** MB
Test-scenario (room 1)				
	Accuracy	Accuracy 1-hop	Accuracy 2-hop	Memory
T-OS3	20.7%	**92.1**%	**100**%	718.8 MB
GOLN	**25.9**%	77.8%	92.6%	**116.8** MB
Test-scenario (room 2)				
	Accuracy	Accuracy 1-hop	Accuracy 2-hop	Memory
T-OS3	4.8%	59.4%	63.2%	669.6 MB
GOLN	**21.7**%	**69.6**%	**87.0**%	**116.8** MB
Test-scenario (room 3)				
	Accuracy	Accuracy 1-hop	Accuracy 2-hop	Memory
T-OS3	28.1%	**70.6**%	**85.3**%	684.7 MB
GOLN	**37.5**%	48.3%	59.9%	**116.8** MB

Table 4.1: Performance comparison in online localization experiments between GOLN and monocular vision-only ORB-SLAM3.

the ground-truth node belong to the same group of four overlapping nodes (due to the graph map structure, composed of four nodes per position as detailed in Section 4.3.2.1). Then, *1-hop* and *2-hop* accuracies are considered. In these cases, the estimation is considered correct if it is distant from the ground-truth node at most, respectively, one hop (*i.e.*, there exists an edge between them) and two hops (*i.e.*, they have at least a shared neighbor node). Finally, memory consumption is evaluated as the last metrics. In our experiments, this storage requirement has been quantified as the fixed GPU memory allocated by the framework. This depends on several forward steps of the ResNet, one for the overall scene and one per each object, and the forward step of the GNN model. Instead, we report the peak of the RAM consumption for the memory required by OS3.

Comparative results In Table 4.1 we compare the proposed approach with a modified version of OS3, adapted to solve a topological localization task (*i.e.*,

Figure 4.7: Test-scenario rooms employed for online localization experiments. The first row comprises the *top-view* of the environments, while the second one the robot *front-view* camera.

named T-OS3). We compare the accuracy results and the memory requirements over the same paths, collected in four different simulated scenarios, as illustrated in Figure 4.7. In the experiments, the first scenario (*i.e.*, training-scenario) is one of the environments used for the model optimization, but the trajectory performed by the robot is totally different. Instead, the others (*i.e.*, the test-scenarios) includes trajectories performed in significantly different environments. It should be specified that these test scenarios are not part of the test set previously described. Instead, they have been generated only for online experiments. These experiments show that, when OS3 succeeds in its task, it achieves remarkable performance. Nevertheless, in challenging environments it can fail in estimating the trajectory, bringing a very low accuracy in our topological problem (*e.g.*, in room 2). Regardless, all the experiments emphasize that our approach results in a significant memory consumption saving. In fact, it was observed that, especially with the increase of the path length, memory requirements for VO algorithms

tend to grow (approaching a peak of almost 1 GB), while our approach allocates a fixed amount of memory (*i.e.*, approximately 117 MB). This saving is mainly due to the low amount of memory required by the GNN employed in order to process the topological map of an indoor environment along with the visual path embeddings.

Results In Table 4.2 we report the results obtained for the three sets described above, *i.e.*, training-graph validation set, training-scenarios validation set, and test set. The experimental results show that the proposed approach is able to provide an accurate estimation in the same environments used for training, and also to generalize in environments with a similar appearance. As expected, the main challenge is represented by environments completely different from the training ones. In addition to the accuracy value, the 1- and 2-hop accuracies are also considered since connected nodes describe similar scenes. Indeed, in indoor scenarios, they correspond to close locations. The experimental results reveal that, even if the estimation on the test set is not accurate, the accuracy 1-hop and accuracy 2-hop are very satisfactory demonstrating that the algorithm is able to perform a high-level understanding of the scene and a sufficiently accurate localization.

	Accuracy	Accuracy 1-hop	Accuracy 2-hop
Train-graphs Validation	81.1%	91.6%	95.8%
Train-scenarios Validation	77.6%	91.4%	96.6%
Test	55.1%	85.7%	93.9%

Table 4.2: Accuracy performance of GOLN framework in several environments, either similar to the training set or totally different.

4.3.4 Discussion and limitations

We propose a novel framework, named GOLN, for a low-memory consumption topological localization by explicitly employing the objects present in an environment. In this approach, the positions of the robot and the objects in the scene are modeled as a graph in order to build a simple but representative map. The algorithm requires a previously-built graph map and the front-view camera installed on the robot. From the image captured by the monocular camera, the framework extracts visual features of the overall scene and of the objects. Experiments have been performed in simulated environments with a ground robot. Experimental results show that accuracy performance is comparable to a *state-of-the-art* approach but the memory required is far below. The limitation of this approach is twofold:

- The framework requires an object detector specifically trained for the indoor scenario. Since they are strictly coupled, the performance of the localization task may drop if the former fails to detect objects.
- Temporal dependencies between successive frames are not taken into account and, therefore, consecutive predictions are completely uncorrelated. However, temporal dependencies might help in enhancing the performance and preventing spurious predictions.

4.4 Aerial platforms

Autonomous Micro Aerial Vehicles (MAVs) are becoming increasingly pervasive in a wide range of applications including surveillance, commercial delivery, and search and rescue. Their popularity stems from the advantages they exhibit over other platforms: i) compact size, ii) agility, and iii) low cost. Enabling them

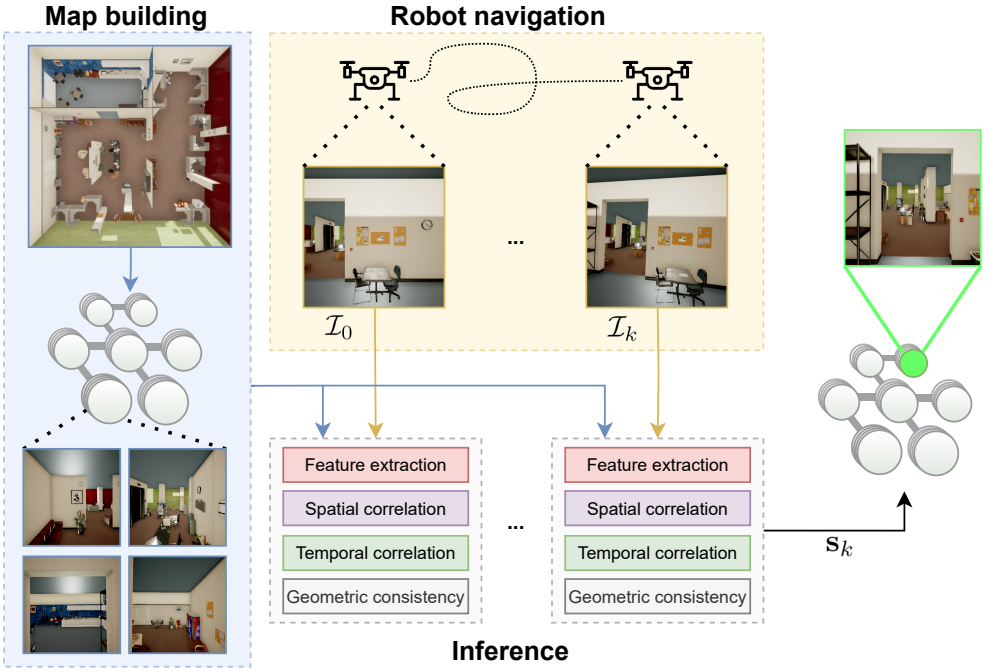


Figure 4.8: Overview of the approach. The proposed strategy processes the images collected by the MAV and predicts the closest node in the topological graph that represents the environment.

with autonomous navigation capabilities requires different modules, among which localization plays a central role in the operation of the overall system. Inspired by these considerations, a novel vision-based topological localization approach designed for MAV applications is presented in the following. An overview of the proposed framework is reported in Figure 4.8.

4.4.1 Contributions

In this study, we focus on vision-based topological localization in indoor scenarios, which present important difficulties due to the high similarity between different locations, *e.g.*, corridors or office rooms. The spatio-temporal relationships that are inherently induced during the robot navigation are considered only by a few

approaches in the literature [69; 88; 121], although only ground robot platforms are taken into account. MAV scenarios, instead, are significantly harder. An MAV is free to move in the 3D space, which increases the number of locations to be modeled in the semantic map and, thus, the chance of perceptual aliasing. In this work, we aim to take a further step in the research on vision-based topological localization for MAVs.
In particular:

- We propose a novel Graph Recurrent Neural Network (GRNN) framework for topological localization in MAV contexts. Our algorithm processes a graph-based map and the sequence of images gathered by the MAV to estimate its position within the topological map.
- Our strategy models spatial relationships between adjacent locations and considers temporal dependencies of the image sequence collected during the navigation to enforce consistency among consecutive predictions.
- To enhance the generalization capabilities of our approach, we develop a new procedural script to generate photorealistic environments and create a dataset composed of several scenarios with randomized characteristics.
- We evaluate our approach on environments significantly different from the training ones to further prove the generalization capabilities of our strategy and compare it against state-of-the-art baselines.

4.4.2 Proposed approach

The objective of this work is to localize an MAV free to fly in the 3D space with respect to a topological map that describes the environment. The topological map is structured as an undirect graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and

edges, respectively. Different from our previous work [121], object information is not explicitly included in the graph nodes, since the requirement of a 3D object detector specifically trained for the evaluation environment is a strict assumption that might not be valid in a wide range of scenarios, limiting the applicability of the proposed approach. Thus, in this study, each graph node $v_i \in \mathcal{V}$ represents a location in the environment and has associated a visual observation of the surrounding scene, captured by the robot front-looking camera. The observation is encoded by using *node features* extracted by a Convolutional Neural Network (CNN) on the images captured at that location during the graph construction phase. An edge $e_{ij} \in \mathcal{E}$, instead, connects two nodes (v_i, v_j) if there exists an obstacle-free straight path between them.

At each timestep, our method processes the topological graph of the scene and the images captured by the MAV during the navigation and predicts the closest location node in the graph. In particular, the features of the current MAV observation and the topological map are provided as inputs to a Graph Recurrent Neural Network (GRNN) model to achieve location (node) prediction. The model takes inspiration from [122] and [126] and exploits the message-passing paradigm to propagate information between the nodes of the graph to capture spatial relations, and through multiple timesteps to model temporal dependencies.

4.4.2.1 Map building

In this Section, we describe the automatic graph construction procedure used to initialize the graph data. We follow the intuitions of [127] to compute a one-voxel-thin skeleton diagram that acts as the graph representation of the environment. Starting from the 3D voxel grid (Figure 4.9a), we obtain the skeleton of the grid (Figure 4.9b) through a thinning algorithm [128; 129]. The skeleton is composed of the set of voxels that locally maximizes the distance from the obstacles. We

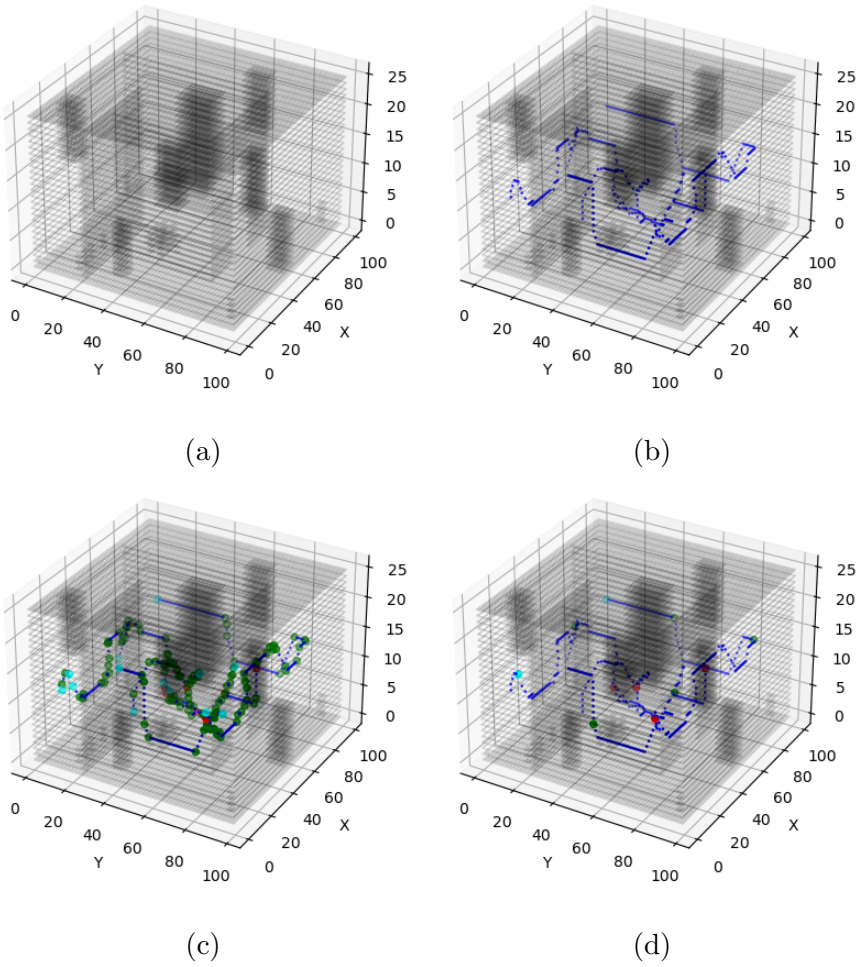


Figure 4.9: 3D topological graph building process. 4.9a and 4.9b represent the voxel grid and the *skeletonization* of the environment, respectively. The following steps 4.9c-4.9d create the nodes and merge the closest ones to induce sparsity on the graph. A specific node color represents the type node according to its number of neighbors, *i.e.*, end-point (in light blue), corner (in green), or branch (in red).

classify each voxel according to its number of neighbors as either an *end*, *branch*, or *corner* node (see Figure 4.9c). The former has one neighbor and represents a boundary in the grid. A branch node, instead, has more than three neighbors, *i.e.*, it is the source of two or more branches. Finally, a corner node is a voxel that has two neighbors, and these three define a curve of 90°, 135°, 225°, or 270° in the

skeleton. Since, after this phase, the number of end, branch, and corner nodes might be significant, we induce sparsity by merging clusters of nodes (see Figure 4.9d). The output of this last phase is the graph structure of the environment.

Finally, we associate visual observations with each node of the graph. However, due to viewpoint variations, different parts of the scene might be observed from the same location depending on the heading of the front-looking camera. Hence, changing the camera point-of-view may lead to a strongly different appearance of the captured images. To overcome this issue, we need to encode the visual information from different viewpoints of each location and, thus, we decide to consider two sets of nodes: *macro nodes* and *micro nodes*. Specifically, for each node v_i, which we refer to as macro node, we capture images by rotating the camera with respect to the yaw angle (*i.e.*, the heading) four times by 90 degrees. Each image is processed by the CNN extractor to compute the node features, which represent the encoding of the scene appearance captured from that orientation. This visual observation is referred to as *micro node* and the collection of the four micro nodes composes the macro node. We indicate the micro nodes with $v_{i,j}$, where the index $j = \{1, 2, 3, 4\}$ indicates the four camera orientations at the location represented by the macro node v_i. Thus, a graph contains M macro nodes, and $N = 4M$ micro nodes.

4.4.2.2 Localization procedure

Given a pre-built topological graph, we aim to localize the MAV during navigation by predicting the graph node closest to its actual position. The overall architecture is shown in Figure 4.10 and is composed of different modules:

- **Features extraction module:** given the input image, computes the associated *global features*. This module is also used during the graph building phase to compute the *node features*;

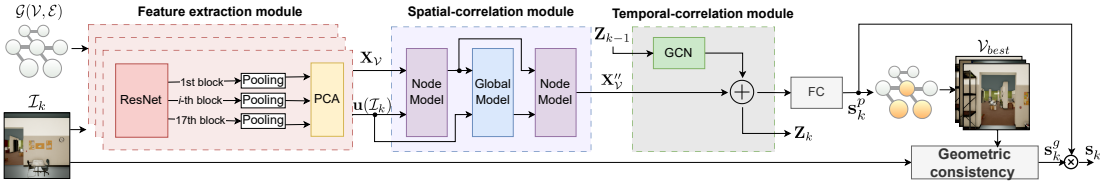


Figure 4.10: Overview of the proposed architecture. The feature extraction module combines the outputs of different convolutional stages of ResNet-50 and processes it with PCA to obtain a 512-dimensional vector. This vector is combined with the features of the neighbor nodes by the spatial-correlation module. The temporal-correlation module, instead, propagates information from past graphs through time. Finally, the node scores computed by the FC layer are refined by a geometric consistency check that re-ranks the top k node candidates.

- **Spatial correlation module:** combines the global features from the current observation with those of graph nodes;
- **Temporal correlation module:** propagates the spatial information through time to capture the temporal correlation that is naturally induced by the sequence of images;
- **Node prediction module:** after spatial and temporal information are integrated into the graph, this module estimates for each node the probability of being the one that corresponds to the current robot location;
- **Geometric consistency module:** weighs the scores from the node prediction module according to a geometric consistency check between the node images and the current observation.

In the following, we describe each module in detail.

Features Extraction Module Using only features extracted from the last layers of deep networks often implies that important information of the input image is lost [23], as we also show in the ablation study (see Table 4.4). Hence,

we propose to use feature maps from multiple levels to exploit different information. Similarly to [90], we combine the output of the 17 convolutional blocks of ResNet-50 [72] to compute the image features. For each feature block, we apply a 2D average pooling to downsample the dimension into a fixed size (*i.e.*, (4, 4)). Feature maps are then flattened, concatenated, and Principal Component Analysis (PCA) is applied to obtain a 512-dimensional feature vector. This procedure is used to obtain both the *global features* $\mathbf{u}(\mathcal{I}_k) \in \mathbb{R}^{512}$ of the image $\mathcal{I}_k$ captured during the MAV navigation, and the micro *node features* $\mathbf{x}(v_{i,j}) \in \mathbb{R}^{512}$, associated with the image of the micro node j (and macro node v_i) collected during the graph building phase. We indicate with $\mathbf{X}_\mathcal{V} \in \mathbb{R}^{M \times 4 \times 512}$ the matrix containing all the micro node features of the graph, such that:

$$\mathbf{X}_\mathcal{V}[i, j, :] = \mathbf{x}(v_{i,j}),\ \forall i = 1, \ldots, M, j = 1, \ldots, 4 \tag{4.11}$$

Algorithm 4 Spatial-correlation module

function $SpatialCorrelationModule(\mathbf{X}_\mathcal{V}, \mathcal{E}, \mathbf{u}(\mathcal{I}_k))$
  $\mathbf{X}'_\mathcal{V} = NodeModel(\mathbf{X}_\mathcal{V}, \mathcal{E}, \mathbf{u}(\mathcal{I}_k))$
  $\mathbf{u}' = GlobalModel(\mathbf{X}'_\mathcal{V}, \mathbf{u}(\mathcal{I}_k))$
  $\mathbf{X}''_\mathcal{V} = NodeModel(\mathbf{X}'_\mathcal{V}, \mathcal{E}, \mathbf{u}'(\mathcal{I}_k))$
  return $\mathbf{X}''_\mathcal{V}$
end function

Spatial correlation The global features of the current observation $\mathbf{u}(\mathcal{I}_k)$ and the node features $\mathbf{X}_\mathcal{V}$ are combined with a Graph Neural Network (GNN), taking advantage of the message-passing framework based on [122]. It propagates $\mathbf{u}(\mathcal{I}_k)$ through the graph and exploits the spatial relationships between graph nodes. The GNN is composed of a combination of computational blocks, as shown in Algorithm 4: the Node Model and Global Model. The former (see Algorithm 5) updates each node's features by combining them with those of its neighbors and

Algorithm 5 Node Model

function $NodeModel(\mathbf{X}_{\mathcal{V}}, \mathcal{E}, \mathbf{u}(\mathcal{I}_k))$
  for $i \in \{1...M\}$
    for $j \in \{1, 2, 3, 4\}$
      $\mathcal{N}_i = \mathbf{X}_{\mathcal{V}}[z, :, :],\ \forall z : \mathbf{e}_{iz} \in \mathcal{E}$
      $\tilde{\mathbf{X}}_{\mathcal{V}}[i, j, :] \leftarrow \Phi^v(\mathbf{X}_{\mathcal{V}}[i, j, :], \mathcal{N}_i)$
    end for
  end for
  $\tilde{\mathbf{u}}(\mathcal{I}_k) \leftarrow \rho^{u \to v}(\mathbf{u}(\mathcal{I}_k))$
  $\mathbf{X}'_{\mathcal{V}} \rightarrow \Phi^{v,u}(\mathbf{X}_{\mathcal{V}}, \tilde{\mathbf{X}}_{\mathcal{V}}, \tilde{\mathbf{u}}(\mathcal{I}_k))$
  return $\mathbf{X}'_{\mathcal{V}}$
end function

Algorithm 6 Global Model

function $GlobalModel(\mathbf{X}'_{\mathcal{V}}, \mathbf{u}(\mathcal{I}_k))$
  $\hat{\mathbf{X}}'_{\mathcal{V}} \leftarrow \rho^{v \to u}(\mathbf{X}'_{\mathcal{V}})$
  $\mathbf{u}'(\mathcal{I}_k) \leftarrow \Phi^{u,v}(\hat{\mathbf{X}}'_{\mathcal{V}}, \mathbf{u}(\mathcal{I}_k))$
  return $\mathbf{u}'(\mathcal{I}_k)$
end function

the global features of the current observation. The Global Model (see Algorithm 6) integrates the updated node features with the global features.

The Node Model uses the following functions:

- $\Phi^v : \mathbb{R}^{512} \rightarrow \mathbb{R}^{256}$ implements the message-passing framework. First, the features of a micro node $\mathbf{X}_{\mathcal{V}}[i, j, :]$ and those of each of its neighbors $\mathcal{N}_i$ are separately fed into a Multi-Layer Perceptron (MLP) network to obtain the associated encodings. These are then averaged into a 256-element vector that is stored into $\tilde{\mathbf{X}}_{\mathcal{V}}[i, j, :]$.

- $\rho^{u \to v} : \mathbb{R}^{512} \rightarrow \mathbb{R}^{M \times 4 \times 512}$ expands the global features $\mathbf{u}(\mathcal{I}_k)$ by replicating them $M \times 4$ times, *i.e.*, $\tilde{\mathbf{u}}(\mathcal{I}_k) \in \mathbb{R}^{M \times 4 \times 512}$.

- $\Phi^{v,u} : \mathbb{R}^{M \times 4 \times 1280} \rightarrow \mathbb{R}^{M \times 4 \times 512}$ is an MLP network with the same structure as Φ^v that combines $\tilde{\mathbf{u}}(\mathcal{I}_k)$, the node features matrix $\mathbf{X}_{\mathcal{V}}$ before message passing, and the aggregated node features $\tilde{\mathbf{X}}_{\mathcal{V}}$. Specifically, the three matrices

are first concatenated and then fed into the MLP to compute the updated node features $\mathbf{X}'_{\mathcal{V}}$.

The Global Model updates the global features as follows:

- $\rho^{v\to u} : \mathbb{R}^{M\times 4\times 512}\to\mathbb{R}^{512}$ averages $\mathbf{X}'_{\mathcal{V}}$ over the nodes to obtain $\hat{\mathbf{X}}'_{\mathcal{V}} \in \mathbb{R}^{512}$.
- $\Phi^{u,v} : \mathbb{R}^{1024}\to\mathbb{R}^{512}$ processes the concatenation of the global features $\mathbf{u}(\mathcal{I}_k)$ and $\hat{\mathbf{X}}'_{\mathcal{V}}$ and updates the global feature vector $\mathbf{u}'(\mathcal{I}_k) \in \mathbb{R}^{512}$. This function is implemented using the same MLP structure as in Φ^v and $\Phi^{v,u}$.

Finally, the updated node $\mathbf{X}'_{\mathcal{V}}$ and global $\mathbf{u}'(\mathcal{I}_k)$ features are further processed by the Node Model to obtain the final node features matrix $\mathbf{X}''_{\mathcal{V}}$. This is fed into the temporal-correlation module to combine it with information from past timesteps and compute the current graph state.

Temporal correlation To model temporal dependencies between consecutive observations, a graph state is generated and propagated through time. The graph state at timestep k is represented by a node feature matrix $\mathbf{Z}_k \in \mathbb{R}^{M\times 4\times 512}$, computed similarly to [126], as follows:

$$\mathbf{Z}_k = \mathrm{ReLU}\left(\mathbf{X}''_{\mathcal{V}} + \mathrm{GCN}\left(\mathbf{Z}_{k-1}, \mathcal{E}\right)\right), \tag{4.12}$$

where $\mathbf{X}''_{\mathcal{V}}$ is the node feature matrix computed by the spatial-correlation module at timestep k, and GCN is a Graph Convolutional Network [130] that selects the information to propagate through time from the previous graph state $\mathbf{Z}_{k-1}$ (see Section 4.4.2.3 for implementation details).

Node prediction The graph state at timestep k is used to compute the score $s^p_k(v_{i,j}) \in \mathbb{R}$, which encodes the probability that the robot location at timestep k

is at the j-th micro node of the i-th macro node. The score vector, in particular, is computed as follows:

$$\mathbf{s}_k^p = \text{Softmax}\left(\mathbf{W}\mathbf{Z}_k + \mathbf{b}\right), \tag{4.13}$$

where $\mathbf{W}$ and $\mathbf{b}$ are the weight matrix and the bias.

Re-ranking via geometric consistency The best location candidates are re-ranked with a geometric consistency check by following the procedure proposed in [78]. We extract patches at multiple scales and compute VLAD descriptors [77] on the images of the top-5 candidate micro nodes $\mathcal{V}_{best}$ (according to the scores in $\mathbf{s}_k^p$), and on the current observation. The VLAD descriptors are then compared to detect mutual matches between the candidate node images and $\mathcal{I}_k$. The matches are used to perform a geometric consistency check through a RANSAC-based scoring procedure (namely GC). Alternatively, a lighter and faster version (namely s-GC) computes the score directly on the matched feature pairs without requiring sampling. The consistency scores $\mathbf{s}_k^g = s_k^g(v_{i,j})$, $\forall v_{i,j} \in \mathcal{V}_{best}$ are then multiplied by the prediction probabilities $s_k^p(v_{i,j})$, and the nodes are re-ranked accordingly to compute the final score vector $\mathbf{s}_k$.

4.4.2.3 Implementation details

Data collection for model training To train the location model we need scenes with varying characteristics and appearance, and with ground truth 3D occupancy maps for graph initialization. Collecting these data from real scenarios is impractical and burdensome, therefore, we decide to gather training samples by exploiting the Unreal Engine (UE4) simulator to generate photorealistic en-

vironments. For this purpose, we develop a procedural script[1] to build indoor scenes with randomized characteristics. Specifically, each scenario is created by first randomizing the shape and the size of rooms and corridors, and then adding walls and floor textures, objects, and furniture with random layouts and characteristics. For each generated scene, the topological graph is first built as described in Section 4.4.2.1. Afterward, the training set is generated by flying an MAV simulated with the AirSim plugin [96] in the scene, collecting 512×512 images from the front-looking camera and associating them with the nearest ground truth location node. To favor generalization over different viewpoints and positions, for each scene multiple random trajectories are sampled and executed. Each one is generated by sampling a starting and an end position, and using a 3D implementation of A* to obtain a collision-free path between those points (if it exists). For model optimization, we generate 400 indoor scenes and collect a total of 156820 images with ground truth node associations.

Implementation and training details As detailed in Section 4.4.2.2, both global and node features are extracted by a CNN with a ResNet-50 backbone pretrained on the ImageNet dataset [91]. The features extracted from the ResNet are flattened and concatenated into a 242688 element vector, which is then processed by PCA to reduce the dimensionality to 512. PCA is fitted only on the node features of the training graphs. The MLPs that implement Φ^v and $\Phi^{v,u}$ of the Node Model include a hidden layer with 512 nodes followed by a ReLU activation and a dropout layer with dropout probability set to 0.2, and an output linear layer. The latter has 128 and 512 nodes in the case of Φ^v and $\Phi^{v,u}$, respectively. The same MLP structure is used for the function $\Phi^{u,v}$ of the Global Model, with both the hidden and output layer composed of 512 nodes. The GCN in

[1] `https://github.com/isarlab-department-engineering/UE4-environment-randomization`

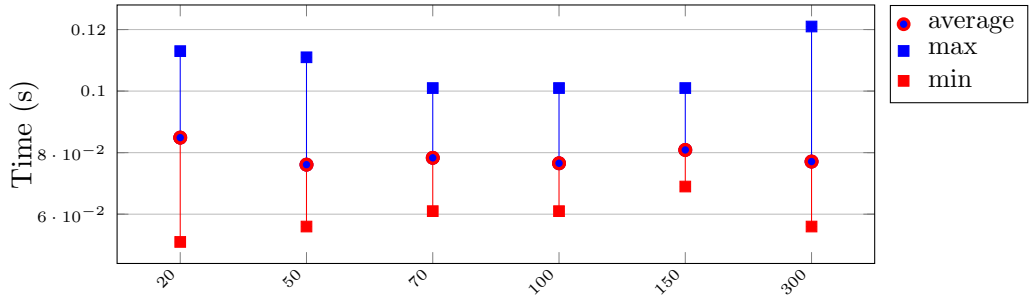


Figure 4.11: Analysis of the computational time as a function of the number of nodes in the graph.

the temporal correlation module includes a single layer that outputs a vector with 512 elements. The GRNN model (*i.e.*, the spatial correlation, temporal correlation, and classification modules) and the GC module (*i.e.*, the CNN that extracts the VLAD descriptor) are trained separately. The latter is trained as detailed in [78], while the GRNN model is optimized by setting the batch size to 32 sequences, and the weight decay to 0.001. The learning rate is initially set to 0.0001 and decreased by a factor of 0.7 every 20 epochs. The training process takes approximately 8 hours and 80 epochs to converge on a workstation equipped with an NVIDIA RTX A6000 48GB.

Inference phase The node features $\mathbf{X}_{\mathcal{V}}$, their embeddings $\tilde{\mathbf{X}}_{\mathcal{V}}$, and the VLAD features of the graph nodes can be computed offline to reduce the inference time as they do not change. Therefore, during inference, only global features for the current observation $\mathbf{u}(\mathcal{I}_k)$ are computed. This step, which involves the spatial correlation, temporal correlation, and node prediction modules, takes approximately 0.08 ± 0.04 seconds. This computational time is nearly unaffected by the graph size, *i.e.*, our approach efficiently scales with the number of nodes. This can be observed in Figure 4.11, where we analyze the maximum inference time

as a function of the graph size. Specifically, we considered six graphs, with an increasing number of nodes (from 20 to 300), and we perform nine runs for each set to compute average, minimum and maximum values (geometric check is not considered for this study since its execution time does not depend on the number of nodes in the graph). If also the GC is performed, the total time increases by 0.11 seconds (0.09 for the faster version) for each image candidate. Hence, if the top 5 candidates are considered for the GC, the total inference time is approximately 0.57 ± 0.08 seconds per prediction.

4.4.3 Experimental results

Evaluation Environments To assess the performance of our approach, we generate and execute additional trajectories different from those used for training. To further evaluate the generalization capabilities, we also generate additional scenes with different characteristics with respect to the training environments. Specifically, we build four evaluation sets as follows:

- **Tr-T**: A set of trajectories different from the training ones, but collected on the same scenes used during optimization (*i.e.*, by using the same topological graph). It contains 20 graphs and 8244 observations collected during the execution of different trajectories.

- **Tr-A**: A set of trajectories collected on environments different from those used during training, although with a similar appearance. It contains 20 graphs and 5958 observations.

- **Test**: A set of trajectories collected on scenes significantly different from the training ones (*i.e.*, different layouts, textures, furniture). It contains 20 graphs and 6286 observations.

(a) (b) (c) (d)

Figure 4.12: Examples of the evaluation environments: **Tr-T** set (4.12a); **Tr-A** set (4.12b); **Test** set (4.12c); **Test-Ph** set (4.12d).

- **Test-Ph**: The last set contains 1 graph and 356 observations collected in a photorealistic office-like environment and is, therefore, the most challenging due to the significant domain shift with respect to the training scenes.

On average, the graphs of **Tr-T**, **Tr-A**, and **Test** have 70 micro nodes (± 10), while the graph of **Test-Ph** has 48 micro nodes. The topological maps for the scenes of **Tr-A**, **Test** and **Test-Ph** are built as detailed in Section 4.4.2.1. Some examples of the scenes are shown in Figure 4.12.

Baselines We consider different state-of-the-art approaches in the experimental study. All the baselines are trained on the same training set used to optimize our strategy (see Section 4.4.2.3):

- **GOLN** [121]. This framework is based on the GNN proposed by [122] and exploits the spatial correlation between close locations in the topological map, as detailed in Section 4.3. Differently from the original paper, we do not use the object-based map for the comparison since, in this case, we do not assume the availability of a 3D object detector.

- **ST-GLN** [88]. The approach exploits both spatial correlation and temporal correlation for topological localization through a Graph Convolutional

LSTM (GCLSTM) based on [131]. For each frame collected by a ground robot, the framework returns the closest node in the graph.

- **NetVLAD** [76]. This approach models the environment with a database of uncorrelated images, against which the current query image is matched to obtain the most similar ones. We train the model by using the training graph images as the database images, and the training observations collected by the MAV as the query images.

- **Patch-NetVLAD** [78]. The improved version of [76], where each image is described through patch-level descriptors. The localization performance is enhanced through a geometric consistency check between the query frame and the map images. As for [76], we train the model by employing the training graph images as database images and training observations as query images. Since in our approach the GC is applied on the top 5 candidates, for comparison purposes we run Patch-NetVLAD (referred to as P-NetVLAD) by both considering all the node images and only the top-5 candidates in its GC step (namely P-NetVLAD-5). The lightweight version is instead referred to as P-NetVLAD-s.

Evaluation metrics At test time, our approach processes image observations and predicts the micro node in the topological graph encoding the robot location. To evaluate and compare its performance against the baselines, we introduce three metrics:

- **Macro Node Accuracy (MNA)**: the accuracy score computed at macro node level. In particular, the prediction is considered to be correct if any of the four *micro nodes* associated with the ground truth macro node is predicted.

- **One Hop Accuracy (OHA)**: the predicted node is considered correct if it corresponds to the ground truth macro node or one of its neighbors at distance one in the graph (*i.e.*, with an edge connecting them to the ground truth macro node).

- **Recall@N (R@N)**: a metric commonly used in VPR tasks adapted to our setting. Usually, a metric distance is set as a threshold (*e.g.*, 25 meters [78]), but this is not suitable for indoor scenarios since two locations might be close in the metric space while belonging to two different rooms, due to the presence of walls. A prediction is considered correct if at least one of the top N predicted nodes is at most one hop distant from the ground-truth micro node in the graph map. It should be noted that R@1 corresponds to OHA.

For each evaluation set, we first compute the MNA and OHA metrics on each environment separately and then their overall mean and standard deviation. For the R@N we show the comparative results between the proposed approach and the baselines obtained on the **Test-Ph** environment.

Comparative results In Table 4.3 we compare the Macro Node Accuracy (MNA) and the One Hop Accuracy (OHA) of the proposed approach with those

	Tr-T		Tr-A		Test		Test-Ph	
	MNA	OHA	MNA	OHA	MNA	OHA	MNA	OHA
GOLN [121]	0.64 ± 0.04	0.84 ± 0.04	0.49 ± 0.05	0.72 ± 0.07	0.38 ± 0.06	0.62 ± 0.08	0.43 ± 0.00	0.72 ± 0.00
ST-GLN [88]	0.72 ± 0.04	0.89 ± 0.03	0.64 ± 0.04	0.83 ± 0.05	0.51 ± 0.07	0.73 ± 0.09	0.46 ± 0.00	0.71 ± 0.00
NetVLAD [76]	0.64 ± 0.07	0.81 ± 0.08	0.66 ± 0.07	0.85 ± 0.05	0.53 ± 0.06	0.76 ± 0.05	0.59 ± 0.00	0.76 ± 0.00
P-NetVLAD-5 [78]	0.76 ± 0.05	0.88 ± 0.04	0.76 ± 0.06	0.90 ± 0.04	0.71 ± 0.06	0.88 ± 0.05	0.67 ± 0.00	0.85 ± 0.00
P-NetVLAD-s [78]	0.77 ± 0.04	0.89 ± 0.02	0.75 ± 0.06	0.90 ± 0.03	0.70 ± 0.05	0.87 ± 0.04	0.68 ± 0.00	0.87 ± 0.00
P-NetVLAD [78]	0.78 ± 0.04	0.89 ± 0.04	$\mathbf{0.77} \pm 0.06$	0.90 ± 0.04	0.77 ± 0.04	0.91 ± 0.03	0.69 ± 0.00	0.87 ± 0.00
Ours (w/o GC)	0.84 ± 0.03	0.96 ± 0.02	0.75 ± 0.05	0.93 ± 0.04	0.65 ± 0.07	0.87 ± 0.07	0.58 ± 0.00	0.85 ± 0.00
Ours (w/ s-GC)	0.85 ± 0.02	0.97 ± 0.01	$\mathbf{0.77} \pm 0.01$	$\mathbf{0.94} \pm 0.01$	0.71 ± 0.01	0.89 ± 0.02	0.73 ± 0.00	0.91 ± 0.00
Ours (w/ GC)	$\mathbf{0.86} \pm 0.03$	$\mathbf{0.97} \pm 0.02$	0.76 ± 0.05	$\mathbf{0.94} \pm 0.04$	$\mathbf{0.79} \pm 0.05$	$\mathbf{0.94} \pm 0.04$	$\mathbf{0.74} \pm 0.00$	$\mathbf{0.92} \pm 0.00$

Table 4.3: The Table reports mean and standard deviation for MNA and OHA computed on the environments of each evaluation set.

of the baselines introduced in Section 4.4.3. For our approach, we reported the results obtained with both the RANSAC-based version (GC) and the faster version (s-GC) of the geometric consistency module described in Section 4.4.2.2. In addition, we report the results obtained without (w/o GC), where the predicted node is the one with the highest probability value returned by the GRNN.

Results show that the proposed strategy outperforms NetVLAD. As an instance, in the Test evaluation set MNA and OHA are respectively 12% and 11% higher for our approach without the geometric check than NetVLAD. If the consistency check is included, the improvement over this baseline is even more evident (15% higher on MNA and 16% on OHA). This proves that modeling spatiotemporal dependencies is beneficial for localization.

Our strategy outperforms also the other topological works, *i.e.*, GOLN [121] and ST-GLN [88], demonstrating its superiority in managing localization with MAV platforms. On the photorealistic set **Test-Ph**, the accuracy improvement with respect to these two baselines is up to 30% and 20% with respect to MNA and OHA, respectively.

Among the baselines, Patch-NetVLAD [78] is certainly the one achieving the best results, mainly thanks to its geometric consistency check. Reducing the number of candidates to be re-ranked does not have a significant impact on its performance. We can, indeed, observe a drop of only approximately 2% in the metrics when only 5 candidates are considered.

When compared with our version without the geometric consistency check, Patch-NetVLAD performance is on par and, in some cases, slightly better. However, when the geometric re-ranking is included, our approach outperforms Patch-NetVLAD in the most challenging scenarios, *i.e.*, the **Test** and the **Test-Ph** sets, which differ from the scenes used for training. In the latter set, the improvement is up to 5% with respect to both metrics. These results highlight two impor-

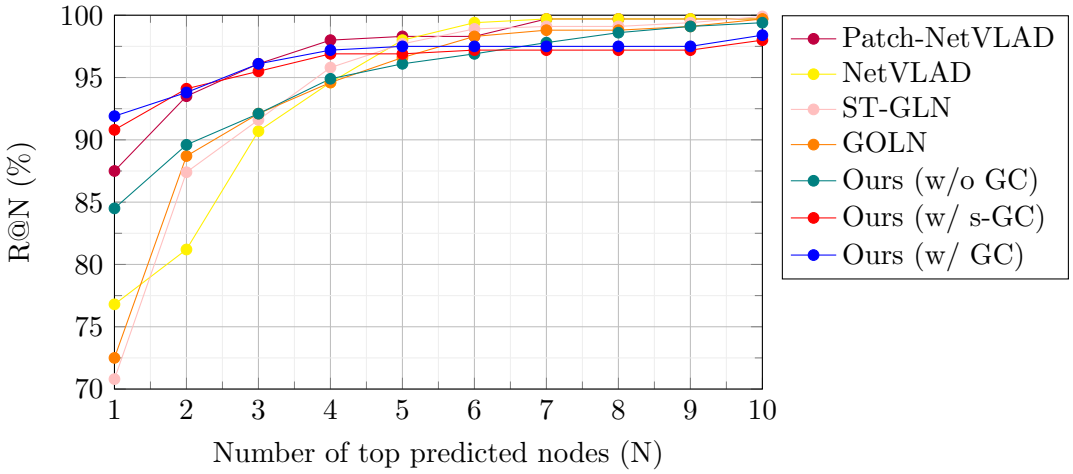


Figure 4.13: Comparison of the R@N metric on the Test-Ph environment.

tant aspects. First, they show the generalization capabilities of the strategy we propose. Second, since our GC is similar to that of Patch-NetVLAD, this demonstrates that the node probability scores computed by our GRNN model are more accurate than those of the baseline.

Figure 4.13 shows the R@N performance of our approach compared to the state-of-the-art, considering the top N predicted nodes (from 1 to 10). Our approach (with GC included) outperforms the baselines for $N < 4$, proving that it requires few candidates to correctly predict the location.

Ablation study We also perform an ablation study to show the impact of the components of the proposed framework on the performance. Specifically, we analyze the effect of the following components:

- **Multi-level Features (MLF)**: fusing multi-level information during the feature extraction phase;
- **Temporal Correlation (TC)**: using the temporal correlation module;
- **Geometric Consistency (GC)**: re-ranking the predictions through the

MLF	TC	GC	Tr-T		Tr-A		Test		Test-Ph	
			MNA	OHA	MNA	OHA	MNA	OHA	MNA	OHA
-	-	-	0.64 ± 0.04	0.84 ± 0.04	0.49 ± 0.05	0.72 ± 0.07	0.38 ± 0.06	0.62 ± 0.08	0.43 ± 0.00	0.72 ± 0.00
-	✓	-	0.73 ± 0.05	0.92 ± 0.04	0.61 ± 0.05	0.88 ± 0.06	0.47 ± 0.07	0.80 ± 0.08	0.42 ± 0.00	0.81 ± 0.00
✓	-	-	0.80 ± 0.04	0.94 ± 0.02	0.64 ± 0.05	0.85 ± 0.04	0.51 ± 0.07	0.77 ± 0.06	0.47 ± 0.00	0.72 ± 0.00
✓	✓	-	0.84 ± 0.03	0.96 ± 0.02	0.75 ± 0.05	0.93 ± 0.04	0.65 ± 0.07	0.87 ± 0.07	0.58 ± 0.00	0.85 ± 0.00
✓	✓	✓	$\mathbf{0.86} \pm 0.03$	$\mathbf{0.97} \pm 0.02$	$\mathbf{0.76} \pm 0.05$	$\mathbf{0.94} \pm 0.04$	$\mathbf{0.79} \pm 0.05$	$\mathbf{0.94} \pm 0.04$	$\mathbf{0.74} \pm 0.00$	$\mathbf{0.92} \pm 0.00$

Table 4.4: Ablation study on the main components of the proposed approach. Check marks (✓) indicate if a particular component has been enabled. For both the metrics, we report the mean and the standard deviation computed on the environments of each evaluation set.

geometric consistency check.

The quantitative results are reported in Table 4.4. As expected, a large performance drop is observed when the three components are absent. An accuracy gain (up to 12% and 18% on MNA and OHA, respectively) is experienced when the TC module is included. Modeling temporal dependencies is, indeed, crucial to improve the robustness to perceptual aliasing. This, in practice, allows the approach to avoid predicting nodes that exhibit a similar appearance to the current image but are inconsistent with past observations.

The improvement is, in some cases, even more significant when the MLF module is added. This proves that aggregating features from different layers of ResNet is beneficial for the localization task since a richer semantic representation of the scene is provided to the spatial and temporal modules. When both MLF and TC modules are enabled, the accuracy is further improved (up to 18% on MNA and 13% on OHA with respect to the versions that use them separately).

Results indicate that incorporating the Geometric Consistency (GC) module is particularly effective in environments that differ from the training data, such as Tr-A and Test. This is expected due to domain shifts and perceptual aliasing issues, which can cause the node with the highest estimated score to be incorrect. The GC module can re-rank nodes based on geometric consistency, filtering out spurious associations and improving performance.

4.4.4 Discussion and limitations

Backbones of GRNN and GC the final setup proposed encompasses a combination of two different backbones for the features extraction in the GRNN network and that used to compute VLAD, respectively ResNet-50 and VGG-16. We also investigated the use of the same backbone (either ResNet or VGG) for both the feature computations to reduce the execution time during inference. However, this choice resulted in a performance drop and, therefore, we opted for the first strategy (ResNet+VGG), as it maximizes the trade-off between accuracy and time.

Deployment on real MAVs reducing the computational cost is a major aspect to deploy our approach on a real platform. Low-level optimizations (*e.g.*, TensorRT) and distillation approaches to reduce the execution time of the deep networks without affecting the localization accuracy can be considered. In addition, different configurations that combine onboard computations (*e.g.*, on Nvidia NX boards) and offboard ones on dedicated workstations will be investigated, also considering the roundtrip delay induced by the communication.

Generalization on real environments an important challenge lies in the domain gap between simulated and real-world images. Besides fine-tuning with real-world images, we are considering different approaches to address this problem and maximize the generalization capabilities of our network. One approach that is under investigation, is to take advantage of Generative Adversarial Networks (GANs) to reduce the gap between the two domains.

Pre-built graph Our approach performs localization on a graph that is pre-built offline and does not change during the inference phase. This assumption

could be removed by extending our solution with online graph building strategies that build and update the topological map as the robot navigates in the scene.

Chapter 5

Conclusions

The presented Thesis aims to investigate novel solutions in the vision-based robotic localization field. They are particularly useful in GPS-denied environments, such as in indoor scenarios, where external sources of information are not available and the success of autonomous navigation pipelines depends on the onboard sensors. Since vision-based approaches are mainly grounded on VO and SLAM frameworks, benchmarks and comparative analyses result to be crucial for an in-depth understanding of the limitations present in different settings and scenarios. Thus, two benchmark studies are presented to help researchers and practitioners in selecting the best approach for a specific use case and to highlight the drawbacks of each method. The main contributions are then related to two topological localization solutions for ground and aerial vehicles, which aim to trade off precision with a more robust and higher-level positioning system. The latter presents a vision-based topological approach to localize autonomous MAVs in a pre-built topological map. Spatio-temporal dependencies between location and consecutive observations are modeled by a graph recurrent neural network, and its estimates are refined by a consistency module that re-ranks the top node candidates according to a geometric check. Experiments show that this strategy outperforms other topological localization and visual place recognition baselines,

and is able to generalize on environments that differ from those used during the training phase.

Limitations are mainly related to the simulation-to-real domain gap. First, the deployment on a real robotic platform requires low-level optimizations for reducing the computational cost and running the inference process directly on embedded boards, *e.g.*, Nvidia Xavier NX boards. Future work will also integrate domain adaptation techniques to favor the deployment in real scenarios, both indoor and outdoor, without the need for further optimization steps. For instance, the domain adaptation capabilities of GANs can help in reducing the gap between the two domains in real scenarios applications. Furthermore, online graph-building strategies will be considered to avoid the map pre-initialization phase. Indeed, the capability of a mobile robot to generate, update, and maintain a representation of the space during navigation would represent a further step toward robot autonomy.

Bibliography

[1] S. Huang and G. Dissanayake, "Robot localization: An introduction," in *Wiley Encyclopedia of Electrical and Electronics Engineering*, pp. 1–10, 2016. 11

[2] J. Almeida and V. M. Santos, "Real time egomotion of a nonholonomic vehicle using lidar measurements," *Journal of Field Robotics*, vol. 30, pp. 129–141, 01 2013. 11

[3] J. Hyun and H. Myung, "Nr-uio: Nlos-robust uwb-inertial odometry based on interacting multiple model and nlos factor estimation," *Sensors*, vol. 21, p. 7886, 11 2021. 11

[4] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *IEEE robotics & automation magazine*, vol. 18, no. 4, pp. 80–92, 2011. 11

[5] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. 11

[6] G. Bresson, Z. Alsayed, L. Yu, and S. Glaser, "Simultaneous localization and mapping: A survey of current trends in autonomous driving," *IEEE Transactions on Intelligent Vehicles*, vol. 2, no. 3, pp. 194–220, 2017. 11

[7] J. Delmerico and D. Scaramuzza, "A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots," in *ICRA*, 2018. 11, 22, 24, 65

[8] H. P. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover.," tech. rep., Stanford Univ CA Dept of Computer Science, 1980. 11

[9] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, vol. 1, pp. I–I, 2004. 11

[10] A. Geiger, J. Ziegler, and C. Stiller, "Stereoscan: Dense 3d reconstruction in real-time," in *Intelligent Vehicles Symposium (IV)*, 2011. 12, 23

[11] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, "ORB-SLAM: a versatile and accurate monocular slam system," *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, 2015. 12, 19

[12] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision (ECCV)*, September 2014. 12, 22

[13] J. Engel, J. Stückler, and D. Cremers, "Large-scale direct slam with stereo cameras," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1935–1942, IEEE, 2015. 12

[14] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, mar 2018. 12, 17, 22, 39, 59

[15] C. Forster, M. Pizzoli, and D. Scaramuzza, "Svo: Fast semi-direct monocular visual odometry," in *2014 IEEE international conference on robotics and automation (ICRA)*, pp. 15–22, IEEE, 2014. 12, 22

[16] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 298–304, IEEE, 2015. 12, 20, 39

[17] T. Qin, P. Li, and S. Shen, "Vins-mono: A robust and versatile monocular visual-inertial state estimator," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, 2018. 12, 18, 22, 39, 59

[18] P. Geneva, K. Eckenhoff, W. Lee, Y. Yang, and G. Huang, "Openvins: A research platform for visual-inertial estimation," in *Proc. of the IEEE International Conference on Robotics and Automation*, (Paris, France), 2020. 12, 18, 40, 59

[19] G. Costante and T. A. Ciarfuglia, "Ls-vo: Learning dense optical subspace for robust visual odometry estimation," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1735–1742, 2018. 13, 17

[20] G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia, "Exploring Representation Learning with CNNs for Frame-to-Frame Ego-Motion Estimation," *Robotics and Automation Letters, IEEE*, vol. 1, no. 1, pp. 18–25, 2016. 13, 17

[21] E. C. Tolman, "Cognitive maps in rats and men," in *Psychological Review*, vol. 55(4), pp. 770–778, 1948. 14, 66

[22] X. Zhang, L. Wang, and Y. Su, "Visual place recognition: A survey from

deep learning perspective," *Pattern Recognition*, vol. 113, p. 107760, 2021. 14, 26, 29, 66

[23] S. Garg, T. Fischer, and M. Milford, "Where is your place, visual place recognition?," *International Joint Conferences on Artificial Intelligence*, 03 2021. 14, 26, 66, 94

[24] T. Barros, R. Pereira, L. Garrote, C. Premebida, and U. J. Nunes, "Place recognition survey: An update on deep learning approaches," *ArXiv*, vol. abs/2106.10458, 2021. 14, 26, 30, 66

[25] B. J. Kuipers, "Modeling spatial knowledge," in *Cognitive Science*, vol. 2, pp. 129 – 153, 1978. 14, 30

[26] V. Guizilini and F. Ramos, "Semi-parametric models for visual odometry," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 3482–3489, IEEE, 2012. 17

[27] T. A. Ciarfuglia, G. Costante, P. Valigi, and E. Ricci, "Evaluation of non-geometric methods for visual odometry," *Robotics and Autonomous Systems*, vol. 62, no. 12, pp. 1717 – 1730, 2014. 17

[28] S. Wang, R. Clark, H. Wen, and N. Trigoni, "DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2043–2050, May 2017. 17

[29] G. Costante and M. Mancini, "Uncertainty estimation for data-driven visual odometry," *IEEE Transactions on Robotics*, vol. 36, no. 6, pp. 1738–1757, 2020. 17

[30] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*, pp. 430–443, Springer, 2006. 18

[31] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision (ijcai)," *[No source information available]*, vol. 81, 04 1981. 18

[32] J. Shi *et al.*, "Good features to track," in *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pp. 593–600, IEEE, 1994. 18

[33] D. Gálvez-López and J. D. Tardós, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, pp. 1188–1197, October 2012. 18

[34] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*, pp. 2564–2571, Ieee, 2011. 19

[35] R. Mur-Artal and J. D. Tardós, "Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017. 19, 59

[36] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, 2017. 19

[37] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, pp. 1–17, 2021. 19, 39, 40, 83

[38] M. Bloesch, M. Burri, S. Omari, M. Hutter, and R. Siegwart, "Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback," *The International Journal of Robotics Research*, vol. 36, no. 10, pp. 1053–1072, 2017. 20

[39] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, "Unsupervised learning of depth and ego-motion from video," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6612–6619, 2017. 20, 40

[40] A. Ranjan, V. Jampani, L. Balles, K. Kim, D. Sun, J. Wulff, and M. J. Black, "Competitive collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation," *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12232–12241, 2019. 21, 40

[41] C. Godard, O. Mac Aodha, M. Firman, and G. J. Brostow, "Digging into self-supervised monocular depth estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3828–3838, 2019. 21, 40

[42] V. Guizilini, R. Ambrus, S. Pillai, A. Raventos, and A. Gaidon, "3d packing for self-supervised monocular depth estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2485–2494, 2020. 22, 40

[43] R. Ambrus, V. Guizilini, J. Li, S. Pillai, and A. Gaidon, "Two Stream Networks for Self-Supervised Ego-Motion Estimation," in *Conference on Robot Learning (CoRL)*, October 2019. 22

[44] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving?

the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 22, 23, 24, 35

[45] R. Giubilato, S. Chiodini, M. Pertile, and S. Debei, "An evaluation of ros-compatible stereo visual slam methods on a nvidia jetson tx2," *Measurement*, vol. 140, pp. 161–170, 2019. 22, 23

[46] J. Jeon, S. Jung, E. Lee, D. Choi, and H. Myung, "Run your visual-inertial odometry on nvidia jetson: Benchmark tests on a micro aerial vehicle," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5332–5339, 2021. 22

[47] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual–inertial odometry using nonlinear optimization," *The International Journal of Robotics Research*, vol. 34, no. 3, pp. 314–334, 2015. 22

[48] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016. 22, 23, 24, 36

[49] R. Ambrus, V. Guizilini, J. Li, and S. P. A. Gaidon, "Two stream networks for self-supervised ego-motion estimation," in *Conference on Robot Learning*, pp. 1052–1061, PMLR, 2020. 23

[50] T. Pire, T. Fischer, J. Civera, P. De Cristóforis, and J. J. Berlles, "Stereo parallel tracking and mapping for robot localization," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1373–1378, IEEE, 2015. 23

[51] M. Labbe and F. Michaud, "Appearance-based loop closure detection for online large-scale and long-term operation," *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 734–745, 2013. 23

[52] A. Alapetite, Z. Wang, J. P. Hansen, M. Zajaczkowski, and M. Patalan, "Comparison of three off-the-shelf visual odometry systems," *Robotics*, vol. 9, no. 3, p. 56, 2020. 23

[53] E. Mingachev, R. Lavrenov, T. Tsoy, F. Matsuno, M. Svinin, J. Suthakorn, and E. Magid, "Comparison of ros-based monocular visual slam methods: Dso, ldso, orb-slam2 and dynaslam," in *International Conference on Interactive Collaborative Robotics*, pp. 222–233, Springer, 2020. 23

[54] X. Gao, R. Wang, N. Demmel, and D. Cremers, "Ldso: Direct sparse odometry with loop closure," in *iros*, October 2018. 23

[55] B. Bescos, J. M. Fácil, J. Civera, and J. Neira, "Dynaslam: Tracking, mapping, and inpainting in dynamic scenes," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 4076–4083, 2018. 23

[56] D. Schubert, T. Goll, N. Demmel, V. Usenko, J. Stueckler, and D. Cremers, "The tum vi benchmark for evaluating visual-inertial odometry," in *International Conference on Intelligent Robots and Systems (IROS)*, October 2018. 23, 24, 36

[57] G. Huang, "Visual-inertial navigation: A concise review," in *2019 international conference on robotics and automation (ICRA)*, pp. 9572–9582, IEEE, 2019. 23

[58] L. Jinyu, Y. Bangbang, C. Danpeng, W. Nan, Z. Guofeng, and B. Hujun, "Survey and evaluation of monocular visual-inertial slam algorithms for

augmented reality," *Virtual Reality & Intelligent Hardware*, vol. 1, no. 4, pp. 386–410, 2019. 23

[59] B. Bodin, H. Wagstaff, S. Saecdi, L. Nardi, E. Vespa, J. Mawer, A. Nisbet, M. Luján, S. Furber, A. J. Davison, *et al.*, "Slambench2: Multi-objective head-to-head benchmarking for visual slam," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3637–3644, IEEE, 2018. 24

[60] M. Bujanca, P. Gafton, S. Saeedi, A. Nisbet, B. Bodin, M. F. O'Boyle, A. J. Davison, P. H. Kelly, G. Riley, B. Lennox, *et al.*, "Slambench 3.0: Systematic automated reproducible evaluation of slam systems for robot vision challenges and scene understanding," in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 6351–6358, IEEE, 2019. 24

[61] J. Jeon, S. Jung, E. Lee, D. Choi, and H. Myung, "Run your visual-inertial odometry on nvidia jetson: Benchmark tests on a micro aerial vehicle," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5332–5339, 2021. 24

[62] T. Qin, J. Pan, S. Cao, and S. Shen, "A general optimization-based framework for local odometry estimation with multiple sensors," 2019. 24

[63] Y. Lu and S. Young, "A survey of public datasets for computer vision tasks in precision agriculture," *Computers and Electronics in Agriculture*, vol. 178, p. 105760, 2020. 25

[64] C. Potena, R. F. Carpio, N. Pietroni, J. Maiolini, G. Ulivi, E. Garone, and A. Gasparri, "Suckers emission detection and volume estimation for the precision farming of hazelnut orchards," in *2020 IEEE Conference on Control Technology and Applications (CCTA)*, pp. 285–290, IEEE, 2020. 25

[65] E. Bellocchio, G. Costante, S. Cascianelli, M. L. Fravolini, and P. Valigi, "Combining domain adaptation and spatial consistency for unseen fruits counting: a quasi-unsupervised approach," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1079–1086, 2020. 25

[66] E. Bellocchio, F. Crocetti, G. Costante, M. L. Fravolini, and P. Valigi, "A novel vision-based weakly supervised framework for autonomous yield estimation in agricultural applications," *Engineering Applications of Artificial Intelligence*, vol. 109, p. 104615, 2022. 25

[67] K. Tsintotas, L. Bampis, and A. Gasteratos, "The revisiting problem in simultaneous localization and mapping: A survey on visual loop closure detection," in *IEEE Transactions on Intelligent Transportation Systems*, 2022. 26

[68] F. Maffra, Z. Chen, and M. Chli, "Viewpoint-tolerant place recognition combining 2d and 3d information for uav navigation," in *ICRA*, 2018. 26, 29

[69] K. Chen, J. P. de Vicente, G. Sepulveda, F. Xia, A. Soto, M. Vazquez, and S. Savarese, "A behavioral approach to visual navigation with graph localization networks," in *Proceedings of Robotics: Science and Systems*, (FreiburgimBreisgau, Germany), 2019. 26, 30, 90

[70] D. S. Chaplot, R. Salakhutdinov, A. Gupta, and S. Gupta, "Neural topological slam for visual navigation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 26

[71] R. R. Wiyatno, A. Xu, and L. Paull, "Lifelong topological visual navigation," *IEEE Robotics and Automation Letters*, vol. 7, pp. 9271–9278, 2021. 26

[72] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. 27, 32, 77, 95

[73] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015. 27

[74] Z. Chen, O. Lam, A. Jacobson, and M. Milford, "Convolutional neural network-based place recognition," *ArXiv*, vol. abs/1411.1509, 2014. 27

[75] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: An astounding baseline for recognition," *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 512–519, 2014. 27

[76] R. Arandjelović, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 27, 28, 103, 104

[77] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 27, 98

[78] S. Hausler, S. Garg, M. Xu, M. Milford, and T. Fischer, "Patch-NetVLAD: Multi-scale fusion of locally-global descriptors for place recognition," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 28, 98, 100, 103, 104, 105

[79] M. J. Milford and G. F. Wyeth, "Seqslam: Visual route-based navigation for

sunny summer days and stormy winter nights," in *2012 IEEE International Conference on Robotics and Automation*, pp. 1643–1649, 2012. 29

[80] R. Mereu, G. Trivigno, G. Berton, C. Masone, and B. Caputo, "Learning sequential descriptors for sequence-based visual place recognition," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 10383–10390, 2022. 29

[81] F. Lu, B. Chen, X.-D. Zhou, and D. Song, "Sta-vpr: Spatio-temporal alignment for visual place recognition," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4297–4304, 2021. 29

[82] K. Tsintotas, L. Bampis, and A. Gasteratos, "Assigning visual words to places for loop closure detection," in *IEEE Intl. Conf. on Robotics and Automation, ICRA*, pp. 5979–5985, 2018. 29

[83] K. Tsintotas, L. Bampis, and A. Gasteratos, "Probabilistic appearance-based place recognition through bag of tracked words," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1737–1744, 2019. 29

[84] K. Tsintotas, L. Bampis, and A. Gasteratos, "Modest-vocabulary loop-closure detection with incremental bag of tracked words," *Robotics and Autonomous Systems*, vol. 141, p. 103782, 2021. 29

[85] E. Garcia-Fidalgo and A. Ortiz, "Vision-based topological mapping and localization methods: A survey," *Robotics and Autonomous Systems*, vol. 64, pp. 1–20, 2015. 30

[86] I. Ulrich and I. Nourbakhsh, "Appearance-based place recognition for topological localization," in *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2, pp. 1023–1029, 2000. 30

[87] Z. Hong, Y. Petillot, D. Lane, Y. Miao, and S. Wang, "TextPlace: Visual place recognition and topological localization through reading scene texts," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2861–2870, 2019. 30, 66

[88] T. Niwa, S. Taguchi, and N. Hirose, "Spatio-temporal graph localization networks for image-based navigation," in *IROS*, 2022. 31, 90, 102, 104, 105

[89] F. Xue, X. Wu, S. Cai, and J. Wang, "Learning multi-view camera relocalization with graph neural networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 32

[90] A. Bojko, R. Dupont, M. Tamaazousti, and H. Le Borgne, "Self-improving SLAM in dynamic environments: Learning when to mask," in *British Machine Vision Conference (BMVC)*, 2022. 32, 95

[91] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009. 32, 81, 99

[92] M. Legittimo, S. Felicioni, F. Bagni, A. Tagliavini, A. Dionigi, F. Gatti, M. Verucchi, G. Costante, and M. Bertogna, "A benchmark analysis of data-driven and geometric approaches for robot ego-motion estimation," *Journal of Field Robotics*, pp. 1–29, 2023. 34, 35, 45, 58, 65, 72

[93] F. Crocetti, E. Bellocchio, A. Dionigi, S. Felicioni, G. Costante, M. L. Fravolini, and P. Valigi, "Ard-vo: Agricultural robot data set of vineyards and olive groves," *Journal of Field Robotics*, 2023. 34, 54

[94] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012. 35

[95] A. Handa, T. Whelan, J. McDonald, and A. Davison, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *IEEE Intl. Conf. on Robotics and Automation, ICRA*, (Hong Kong, China), May 2014. 35

[96] S. Shah, D. Dey, C. Lovett, and A. Kapoor, "Airsim: High-fidelity visual and physical simulation for autonomous vehicles," in *Field and Service Robotics*, 2017. 37, 99

[97] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," in *Autonomous Robots 4(4)*, p. 333–349, 1997. 40

[98] M. Esti, R. L. G. Airola, E. Moneta, M. Paperaio, and F. Sinesio, "Qualitative data analysis for an exploratory sensory study of grechetto wine," *Analytica Chimica Acta*, vol. 660, no. 1-2, pp. 63–67, 2010. 56

[99] A. Barriguinha, M. de Castro Neto, and A. Gil, "Vineyard yield estimation, prediction, and forecasting: A systematic literature review," *Agronomy*, vol. 11, no. 9, p. 1789, 2021. 56

[100] L. Sanchez, B. Sams, M. Alsina, N. Hinds, L. Klein, and N. Dokoozlian, "Improving vineyard water use efficiency and yield with variable rate irrigation in california," *Advances in animal biosciences*, vol. 8, no. 2, pp. 574–577, 2017. 56

[101] M. Kazmierski, P. Glémas, J. Rousseau, and B. Tisseyre, "Temporal stability of within-field patterns of ndvi in non irrigated mediterranean vineyards," *Oeno One*, vol. 45, no. 2, pp. 61–73, 2011. 56

[102] T. Frioni, J. VanderWeide, A. Palliotti, S. Tombesi, S. Poni, and P. Sabbatini, "Foliar vs. soil application of ascophyllum nodosum extracts to im-

prove grapevine water stress tolerance," *Scientia Horticulturae*, vol. 277, p. 109807, 2021. 56

[103] M. A. Hassan, M. Yang, A. Rasheed, G. Yang, M. Reynolds, X. Xia, Y. Xiao, and Z. He, "A rapid monitoring of ndvi across the wheat growth cycle for grain yield prediction using a multi-spectral uav platform," *Plant science*, vol. 282, pp. 95–103, 2019. 56

[104] A. Cicatelli, T. Fortunati, I. De Feis, and S. Castiglione, "Oil composition and genetic biodiversity of ancient and new olive (olea europea l.) varieties and accessions of southern italy," *Plant science*, vol. 210, pp. 82–92, 2013. 57

[105] F. Nardi, A. Carapelli, R. Dallai, G. K. Roderick, and F. Frati, "Population structure and colonization history of the olive fly, bactrocera oleae (diptera, tephritidae)," *Molecular ecology*, vol. 14, no. 9, pp. 2729–2738, 2005. 57

[106] R. Malheiro, S. Casal, P. Baptista, and J. A. Pereira, "A review of bactrocera oleae (rossi) impact in olive products: From the tree to the table," *Trends in Food Science & Technology*, vol. 44, no. 2, pp. 226–242, 2015. 57

[107] D. Marchini, R. Petacchi, S. Marchi, *et al.*, "Bactrocera oleae reproductive biology: new evidence on wintering wild populations in olive groves of tuscany (italy)," *Bulletin of Insectology*, vol. 70, no. 1, pp. 121–128, 2017. 57

[108] F. Sanz-Cortés, J. Martinez-Calvo, M. L. Badenes, H. Bleiholder, H. Hack, and U. Llacer, G.; Meier, "Phenological growth stages of olive trees (olea europaea)," *Annals of Applied Biology*, vol. 140, no. 2, pp. 151–157, 2002. 57

[109] F. Huang, W. Wen, J. Zhang, and L.-T. Hsu, "Point wise or feature wise? a benchmark comparison of publicly available lidar odometry algorithms in urban canyons," *IEEE Intelligent Transportation Systems Magazine*, vol. 14, no. 6, pp. 155–173, 2022. 59

[110] N. Jonnavithula, Y. Lyu, Z. Zhang, Q. Liu, and Z. Li, "Lidar odometry methodologies for autonomous driving: A survey," *arXiv preprint arXiv:2109.06120*, 2021. 59

[111] H. Wang, C. Wang, C. Chen, and L. Xie, "F-loam : Fast lidar odometry and mapping," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2020. 59

[112] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758–4765, IEEE, 2018. 59

[113] X. Guo, L. F. Polania, B. Zhu, C. Boncelet, and K. E. Barner, "Graph neural networks for image understanding based on multiple cues: Group emotion recognition and event recognition as use cases," in *2019 22nd International Conference on*, pp. 4287–4292, IEEE, 2014. 68

[114] X. Liu, W. Liu, M. Zhang, J. Chen, L. Gao, C. Yan, and T. Mei, "Social relation recognition from videos via multi-scale spatial-temporal reasoning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3566–3574, IEEE, 2019. 68

[115] S. Felicioni and M. Dimiccoli, "Interaction-gcn: A graph convolutional network based framework for social interaction recognition in egocen-

tric videos," in *2021 IEEE International Conference on Image Processing (ICIP)*, pp. 2348–2352, 2021. 68

[116] D. Ghosal, N. Majumder, S. Poria, N. Chhaya, and A. Gelbukh, "Dialoguegcn: A graph convolutional neural network for emotion recognition in conversation," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019. 68

[117] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, 2021. 68

[118] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, p. 729–734, 2005. 69

[119] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," in *IEEE Transactions on Neural Networks*, p. 61–80, IEEE, 2009. 69

[120] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *Proceedings of the International Conference on Learning Representations*, 2016. 72

[121] S. Felicioni, M. Legittimo, M. L. Fravolini, and G. Costante, "Goln: Graph object-based localization network," in *2021 20th International Conference on Advanced Robotics (ICAR)*, pp. 849–856, 2021. 73, 90, 91, 102, 104, 105

[122] P. Battaglia, J. B. C. Hamrick, V. Bapst, A. Sanchez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre,

F. Song, A. Ballard, J. Gilmer, G. E. Dahl, A. Vaswani, K. Allen, C. Nash, V. J. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, and R. Pascanu, "Relational inductive biases, deep learning, and graph networks," *arXiv*, 2018. 74, 76, 78, 91, 95, 102

[123] C. Gomez, A. C. Hernandez, E. Derner, R. Barber, and R. Babuška, "Object-based pose graph for dynamic indoor environments," in *IEEE Robotics and Automation Letters*, pp. 5401 – 5408, IEEE, 2020. 75

[124] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi, "Visual semantic navigation using scene priors," *7th International Conference on Learning Representations, ICLR*, 2019. 75

[125] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 13, pp. 376–380, 1991. 84

[126] L. Ruiz, F. Gama, and A. Ribeiro, "Gated graph recurrent neural networks," in *2019 27th European Signal Processing Conference (EUSIPCO)*, vol. 68, pp. 6303–6318, 2020. 91, 97

[127] H. Oleynikova, Z. Taylor, R. Y. Siegwart, and J. I. Nieto, "Sparse 3d topological graphs for micro-aerial vehicle planning," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–9, 2018. 91

[128] T. Lee, R. Kashyap, and C. Chu, "Building skeleton models via 3-d medial surface axis thinning algorithms," *CVGIP: Graphical Models and Image Processing*, vol. 56, no. 6, pp. 462–478, 1994. 91

[129] H. Homann, "Implementation of a 3d thinning algorithm," *The Insight Journal*, 2007. 91

[130] B. Jiang, Z. Zhang, D. Lin, J. Tang, and B. Luo, "Semi-Supervised Learning With Graph Learning-Convolutional Networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 97

[131] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured Sequence Modeling with Graph Convolutional Recurrent Networks," *ICONIP*, p. 362–373, 2018. 103

Appendix A

Scientific Publications

Publications

Felicioni, S. and Dimiccoli, M. (2021). *Interaction-GCN: A Graph Convolutional Network Based Framework for Social Interaction Recognition in Egocentric Videos*. In 2021 IEEE International Conference on Image Processing (ICIP).

Felicioni, S., Legittimo, M., Fravolini M. L. and Costante, G. (2021). *GOLN: Graph Object-based Localization Network*. In 2021 International Conference on Advanced Robotics (ICAR).

Legittimo, M., **Felicioni, S.**, Bagni, F., Tagliavini, A., Dionigi, A., Gatti, F., Verucchi, M., Costante, G. and Bertogna, M. (2023). *A Benchmark Analysis of Data-driven and Geometric Approaches for Visual Odometry*. In Journal of Field Robotics (JFR).

Mollica, G., **Felicioni, S.**, Legittimo, M., Meli, L., Costante, G. and Valigi, P. (2023). *MA-VIED: A Multisensor Automotive Visual Inertial Event Dataset*. In IEEE Intelligent Transportation Systems Transactions (T-ITS).

Crocetti, F., Bellocchio, E., Dionigi, A., **Felicioni, S.**, Costante, G., Fravolini, M.L. and Valigi, P. (2023). *ARD-VO: Agricultural Robot Dataset of Vineyards and Olive groves.* In Journal of Field Robotics (JFR).

Felicioni, S., Rizzo, B., Tortorici, C., Costante, G. (2024). *Vision-based Topological Localization for MAVs.* In IEEE Robotics and Automation Letters (RA-L).

Dionigi, A., **Felicioni, S.**, Leomanni, M., Costante, G. (2024). *D-VAT: End-to-End Visual Active Tracking for Micro Aerial Vehicles.* In IEEE Robotics and Automation Letters (RA-L).

Under-review Papers

Felicioni, S., Burani, E., Leomanni, M., Fravolini, M.L., Valigi, P., Costante, G. (2024). *Integrating Occupancy Grid with Semantic Road Information for Autonomous Navigation in Urban Scenarios: A Benchmark Study.* Submitted to 2024 IEEE International Conference on Automation Science and Engineering (CASE).

Acknowledgements

First, I would like to express my gratitude to Prof. Gabriele Costante for his constant support and valuable advice during my studies. Then, to Prof. Paolo Valigi and Prof. Mario Luca Fravolini for their invaluable experience and guidance.

A sincere thanks to the Intelligent Systems, Automation and Robotics Laboratory (ISARLab) team, in particular to Alberto, Marco, Francesco, Raffaele, Alessandro, Elena, Valerio, and Folco, for being colleagues and friends, and for all the special moments spent together.

I would also like to extend my deepest gratitude to Dr. Claudio Tortorici for giving me the opportunity to join the Technology Innovation Institute (TII) in Abu Dhabi, and to all the Autonomous Robotics Research Center (ARRC) members, especially Rafael, Reem, and Adrian, for having enriched my experience.

This journey would not have been possible without the dedicated and unconditional support of my family. Thanks to my parents, Sabrina and Narciso, for having always helped me in realizing my dreams, to Alessio and Sara, to my aunt Daniela and uncle Giampiero, to Sara, to my grandmothers Antonia and Adele. Finally, to my grandparents, Luigi and Iso: I know they would have been proud of me.

To all my friends, in particular to Donato, Saverio, Federico, Mehdi, Giacomo, and Giuseppe, since these years would have been really different without you.

Last, but not least, to Marta, for your patience, your care, your love, and for always being by my side. With you, I am not afraid of the changes and challenges of life.