Université de Montréal

Lifelong Topological Visual Navigation

par

Rey Reza Wiyatno

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

15 octobre 2021

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Lifelong Topological Visual Navigation

présenté par

Rey Reza Wiyatno

a été évalué par un jury composé des personnes suivantes :

Michel Boyer
(président-rapporteur)

Liam Paull
(directeur de recherche)

Sarath Chandar Anbil Parthipan
(membre du jury)

Résumé

La possibilité pour un robot de naviguer en utilisant uniquement la vision est attrayante en raison de sa simplicité. Les approches de navigation traditionnelles basées sur la vision nécessitent une étape préalable de construction de carte qui est ardue et sujette à l'échec, ou ne peuvent que suivre exactement des trajectoires précédemment exécutées. Les nouvelles techniques de navigation visuelle basées sur l'apprentissage réduisent la dépendance à l'égard d'une carte et apprennent plutôt directement des politiques de navigation à partir des images. Il existe actuellement deux paradigmes dominants : les approches de bout en bout qui renoncent entièrement à la représentation explicite de la carte, et les approches topologiques qui préservent toujours une certaine connectivité de l'espace. Cependant, alors que les méthodes de bout en bout ont tendance à éprouver des difficultés dans les tâches de navigation sur de longues distances, les solutions basées sur les cartes topologiques sont sujettes à des défaillances dues à des arêtes erronées dans le graphe.

Dans ce document, nous proposons une méthode de navigation visuelle topologique basée sur l'apprentissage, avec des stratégies de mise à jour du graphe, qui améliore les performances de navigation sur toute la durée de vie du robot. Nous nous inspirons des algorithmes de planification basés sur l'échantillonnage pour construire des graphes topologiques basés sur l'image, ce qui permet d'obtenir des graphes plus épars et d'améliorer les performances de navigation par rapport aux méthodes de base. En outre, contrairement aux contrôleurs qui apprennent à partir d'environnements d'entraînement fixes, nous montrons que notre modèle peut être affiné à l'aide d'un ensemble de données relativement petit provenant de l'environnement réel où le robot est déployé. Enfin, nous démontrons la forte performance du système dans des expériences de navigation de robots dans le monde réel. [1]

Mots-clés : navigation visuelle, apprentissage à vie, robotique, planification

1. Site internet du projet : https://montrealrobotics.ca/ltvn/

Abstract

The ability for a robot to navigate using vision only is appealing due to its simplicity. Traditional vision-based navigation approaches require a prior map-building step that was arduous and prone to failure, or could only exactly follow previously executed trajectories. Newer learning-based visual navigation techniques reduce the reliance on a map and instead directly learn policies from image inputs for navigation. There are currently two prevalent paradigms: end-to-end approaches forego the explicit map representation entirely, and topological approaches which still preserve some loose connectivity of the space. However, while end-to-end methods tend to struggle in long-distance navigation tasks, topological map-based solutions are prone to failure due to spurious edges in the graph.

In this work, we propose a learning-based topological visual navigation method with graph update strategies that improves lifelong navigation performance over time. We take inspiration from sampling-based planning algorithms to build image-based topological graphs, resulting in sparser graphs with higher navigation performance compared to baseline methods. Also, unlike controllers that learn from fixed training environments, we show that our model can be finetuned using a relatively small dataset from the real-world environment where the robot is deployed. Finally, we demonstrate strong system performance in real world robot navigation experiments. [2]

Keywords: visual navigation, lifelong learning, robotics, planning

2. Project page: https://montrealrobotics.ca/ltvn/

Contents

List of Tables

List of Figures

List of Acronyms and Abbreviations

ALVINN	*Autonomous Land Vehicle In a Neural Network*
BRIEF	*Binary Robust Independent Elementary Features*
BRM	*Bayesian Relational Memory*
CML	*Concurrent Mapping and Localization*
CNN	*Convolutional Neural Network*
CPC	*Constrastive Predictive Coding*
FAST	*Features from Accelerated Segment Test*
HTM	*Hallucinative Topological Memory*
ICP	*Iterative Closest Point*
LEAP	*Latent Embeddings for Abstracted Planning*
LQR	*Linear Quadratic Regulator*

MPC	*Model Predictive Control*
ORB	*Oriented FAST and Rotated BRIEF*
PID	*Proportional-Integral-Derivative*
PRM	*Probabilistic Roadmaps*
RGBD	*Red-Green-Blue-Depth*
RL	*Reinforcement Learning*
RRT	*Rapidly-exploring Random Tree*
$SE(2)$	*Special Euclidean group in two dimensions*
SGM	*Sparse Graphical Memory*
SIFT	*Scale-Invariant Image Features*
SLAM	*Simultaneous Localization and Mapping*
SoRB	*Search on Replay Buffer*
SPTM	*Semi-Parametric Topological Memory*

SURF	*Speeded Up Robust Features*
ViNG	*Visual Navigation with Goals*
VT&R	*Visual Teach and Repeat*

Remerciements

I would like to thank people that have helped me throughout my studies.

First and foremost, I thank my advisor, Liam Paull, for the opportunity to do research in his group. Thank you for the mentorship and for the countless discussions that led me to the completion of my research. Thank you for the freedom to explore various research topics before finding a suitable one. I am also grateful for the time when you advised me to take a step back from coding, and to instead look at the problem from the first principle. Thank you for always encouraging me to experiment with a real robot despite its time consuming nature. I still love to watch our work in action in real life. All in all, you have helped me to develop a curious mind and become a better scientist.

To Anqi Xu, who has been a great mentor and a close collaborator in this project. Thank you for spending numerous hours to discuss about every small details in this project. Thank you for also challenging my ideas. You have helped me to develop an attitude to critically evaluate new ideas and becoming more confident in presenting them.

To Mila, IVADO, Mitacs, Element AI, and Microsoft for providing me with financial support throughout my studies. I am so grateful for them, and I hope this work can greatly benefit the scientific community.

To Simon Lacoste-Julien and Leslie Pack Kaelbling who have shared their wisdom in the pursuit of scientific research. Your mindset inspired me to maintain an approach to do good science that comes from an honest place.

To Jeff Orchard for being the first person that introduced me to research, and for teaching me what it means to do science. I am forever grateful for you taking a chance on me when I did not have any experience whatsoever in research.

To the members of the Robotics and Embodied AI Lab (REAL) for all the discussions that we have during group meetings. This goes especially to Krishna for helpful discussions throughout this project, as well as Vincent for being generally helpful and the kindest person.

To Vulfpeck, Cory Wong, Julian Lage, Jon Batiste, Tomo Fujita, Jacob Collier, and the late Claude Debussy. Your music have been playing on repeat as I work on this project, so much that my brain associates the whole graduate school experience with your music.

To all the health workers. The majority of my studies have been conducted during the pandemic, which would have been a chaotic experience without your sacrifices. I am forever grateful for all of you.

To my parents, grandmother, uncles, and aunts who have always supported me in my studies, even when you do not even know what I work on. This goes especially to my uncle and grandmother, who passed away during my studies. I wish I had the chance to show off this work to you both. This work is for you.

To my beloved wife, Irfani. For the tremendous support and patience. For always believing in me more than I believe myself. For always reminding quality triumphs quantity. You are the one person that I want to make the most proud of.

Chapter 1

Introduction

Regardless of the application, the ability to navigate within unstructured environments is essential for mobile robots. For instance, this ability can enable robots to facilitate work in warehouses, and to assist healthcare professionals in hospitals. Despite the progress over the past few decades, robots have yet to be able to reliably navigate using only visual sensors. This is unfortunate because this ability can reduce the overall cost of robots by eliminating the needs of other more costly sensors, which will make them more accessible. We thus focus on developing efficient methods to solve the semantic visual navigation task for mobile robots. In particular, we propose a topological visual navigation method that can improve lifelong navigation performance. [1]

One key challenge for general-purpose autonomous mobile robots is the arduous setup requirements for operating in a new environment. Classical navigation methods combine mapping with separate planning and control modules [**104, 103, 89, 34**]. Traditionally, a robot must first be expertly piloted to collect information from the environment, and build a metric representation of the space, for example using Simultaneous Localization and Mapping (SLAM) and Concurrent Mapping and Localization (CML) [**63, 100**]. However, this often requires costly equipment such as light detection and ranging sensors. To overcome these limitations, one appealing strategy is to forego the global metric map and instead maintain a *topological* representation (i.e., a graph) of the space [**60, 104**]. In this class of approaches, the connectivity of the graph encodes the traversability of the environment, and it is assumed that some local controller is responsible for actually navigating each of the edges.

Another limitation of metric-based navigation systems is that specifying goals in metric space, such as a point coordinate, is not intuitive to humans. Ideally, we need a system where we can specify the navigation goals in some semantic representation that humans can easily comprehend, such as images of target objects or locations. However, teaching a machine to understand semantic representations is also a challenging problem by itself. Recently, we see

1. This work is under review at the *International Conference on Robotics and Automation 2022.*

(a) Image sequence (b) Subgoal poses

Figure 1.1. A sample plan produced with our method to move from the top-left to the bottom-right image. The intermediary subgoal images transition smoothly, which helps the robot's image-based controller easily navigate to the goal.

the emergence of learning-based methods that directly infer actions from images or natural language queries [**123, 20, 2, 112**]. However, these policies tend to be reactive, are not data efficient, and are not suitable for long-distance navigation tasks.

Our method builds a graph that represents an environment where each node corresponds to a particular image of a location. Figure 1.1 illustrates how our agent uses the graph for planning a navigation task in the image space. A crucial component of the system is a trained model for estimating connectivity and traversability between nodes. Our work specifically addresses erroneous labelling when building the graph, which can hamper navigation performance. False positives from this model will cause the robot to try to execute potentially infeasible plans, while false negative predictions may result in failure to find a feasible plan when one actually exists. Compared with other learning-based topological navigation methods [**92, 74, 29, 96**], we share a common algorithmic structure, as shown in Figure 1.2, yet differ in choices for the learned model, data collection procedure, graph building approach, what graph edges encode, controller used, and, in particular, graph update strategy.

To help with graph building, localization, and control, we train our single neural model to take two images and jointly predict if one is reachable from the other, and, if so, their relative transformation in the *Special Euclidean* group in two dimensions (i.e., $SE(2)$). While we initially train this model using simulated environments, we can later finetune it in the target (e.g., real-world) environment. To finetune the model, we collect a small added set of trajectory images with the help of relative geometric information from odometry on the real robot, and generate a finetuning dataset from these images.

We take inspiration from sampling-based motion planners such as Probabilistic Roadmaps (PRM) [**54**] and Rapidly-exploring Random Tree (RRT) [**61**] to build a graph, by sampling nodes from a pool of collected images and using our model to determine their connectivity.

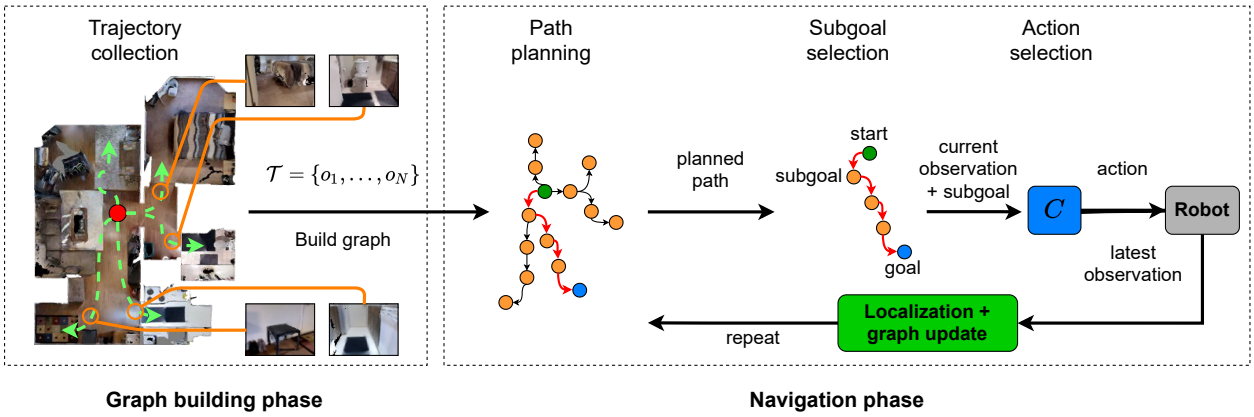


Figure 1.2. Our navigation framework consists of two phases: graph building and navigation. During the graph building phase, the robot collects sequences of observations from the environment and builds a graph using a previously learned model. In the navigation phase, the agent is given a query (i.e., a goal observation), localizes itself on the graph, and plans a path to reach the goal. The agent picks a subgoal from the planned path, predicts what action to take to reach the subgoal, and executes the action with its controller. The agent then relocalizes itself and updates the graph using its latest observations.

During navigation, our agent continuously refines the graph based on its experience, to improve lifelong navigation performance.

The main contributions of our method are as follows:

(1) We develop a sampling-based graph building process that, compared to several existing methods, produces sparser graphs and improves navigation performance,

(2) Our model can be finetuned in a target domain using only a small amount of data while yielding large performance gains,

(3) We propose a graph update procedure that enables the graph to be continually refined during operation, which further improves lifelong navigation performance as the agent executes more navigation queries.

Our evaluations show that our method outperforms the baseline methods when evaluated in various simulated environments. We also show how our graph update method is able to remove spurious edges, which leads to significant improvement in navigation performance over time. We further show real-world navigation instances, where our system can navigate successfully across cluttered environments.

Chapter 2

Background

In this section, we discuss the necessary background to understand the problem of visual navigation that we tackle. First, we discuss various visual navigation tasks variations in Chapter 2.1.1. We then continue on with discussions of the vital components of this work: motion planning (Chapter 2.1.2), control (Chapter 2.1.3), belief update (Chapter 2.1.4), map-based navigation (Chapter 2.1.5), and deep learning for robotics (Chapter 2.1.6). We present related works in Chapter 2.2, which comprises classical visual navigation (Chapter 2.2.1), learning-based visual navigation (Chapter 2.2.2), and learning-based topological navigation (Chapter 2.2.3). Finally, we discuss the baseline methods that we use as comparisons in our experiments in Chapter 2.3.

2.1. Preliminaries

2.1.1. Visual Navigation Tasks

Visual navigation is a task where a robot is given a query to navigate within an environment using only its visual sensors (i.e., cameras). For example, a query can be defined by asking the robot to go to a particular room inside a house. A room, however, can be specified in many different ways, be it as a point coordinate, a description of how the room looks, or an image of the room. Regardless of how the goal is specified, the robot needs to know how to navigate safely and efficiently. Table 2.1 describes different types of visual navigation tasks based on how a query is specified. While there are many variants of tasks available in the context of visual navigation, we focus our work on image goal navigation, where the target is represented as a Red-Green-Blue-Depth (RGBD) image observation.

There are many different approaches to solve these visual navigation tasks. For example, most common visual navigation approaches are based on building a map of the environment, and using the map to guide navigation. Mapless approaches to visual navigation also exist,

Table 2.1. Different types of visual navigation tasks and their descriptions.

Visual Navigation Tasks	**Goal Specification**	**Task Description**
Point goal navigation (e.g., [**112, 5**])	Point coordinate	Navigate to an (x,y) coordinate relative to the robot.
Object goal navigation (e.g., [**114, 18**])	Object category	Navigate to find an object, usually from a predefined set of categories (e.g., "book", "television").
Place or area goal navigation (e.g., [**115**])	Place category	Navigate to a place or an area within an environment (e.g., "kitchen", "lounge").
Image goal navigation (e.g., [**123, 92, 74, 96**])	Image	Navigate to a place specified with an image of a goal location.
Vision and language navigation (e.g., [**2**])	Natural language navigation instructions	Follow a series of natural language navigation instructions (e.g., "Move forward, and then turn right at the second intersection.").
Embodied question answering (e.g., [**20, 121**])	Natural language questions	Answer a natural language question by navigating an environment (e.g., "Where is the microwave?").

and usually are based on optical flow, appearance matching, or visual recognition. We discuss the existing visual navigation techniques in Chapter 2.2.

2.1.2. Motion Planning with Graph

Sense, plan, and act is an early hierarchical robotics paradigm where a robot operates by sequentially sensing the environment, planning its action, and executing its action [**80**]. *Motion planning* is the task of finding a feasible path (i.e., a sequence of states) from a given start location to a given goal location. A *state* refers to the configuration of the robot at a specific time (e.g., spatial coordinates, images, etc.). Feasible path means that the path is free of obstacles, and can be followed by the robot without violating its kinematics or dynamics constraints. In some cases, we may also want to aim for the path to not only be

feasible, but also optimal, where the definition of optimality depends on the use case. For example, it can be finding the path with the shortest distance, the one with the lowest energy required to traverse, or the one with the highest probability of reaching its destination.

There are various techniques that we can use to solve motion planning problems. One approach is the behavior-based planner such as the *bug algorithms* [**70**]. These typically work by moving an agent toward a goal until it hits an obstacle, then traversing the boundary of the obstacle until it can continue moving toward the goal. We are interested in another class of approaches that turns planning into graph construction and search problems. In this class of approaches, once the graph is built, we can then use search algorithms, such as A* [**44**] or Dijkstra's [**24**], to plan a path. We take inspiration from sampling-based planners to develop our graph building algorithm. The idea of sampling-based planners is to iteratively sample a state from the unoccupied space in a map to enable motion planning. In particular, we take inspiration from PRM [**54**] and RRT [**61**] both during graph building and expansion.

PRM builds a graph to perform motion planning by sampling states from a given set of states and connecting them. Concretely, given a set of states S to sample from, which usually is derived from a given environment, we initialize the graph with a set of nodes V randomly sampled from S (i.e., $V \subset S$) without edges between the nodes. Then, for each state $v \in V$, we find a set of nearest nodes U around v, check if v can be reached from every $u \in U$ (and vice-versa) without collision, and if so, connect v and u. PRM is considered a *multi-query* approach, since the graph is only built once and can be reused to perform path planning for any future queries.

In contrast to PRM, RRT is a *single-query* sampling-based planner, which builds a graph for every query we receive. For a given query, RRT incrementally builds a graph until a path from the starting location to the goal is found. In each iteration, RRT samples a random node x_r from a set of nodes S, finds the nearest node x_n in the graph, and interpolates a state x_m in between x_r and x_n. If x_m can be reached from x_n, we then add x_m to the graph and connect x_m and x_n together. This iterative procedure is repeated until a path is found. While RRT does not necessarily produce an optimal path, the algorithm is guaranteed to find a path from the start to goal node, if there is any, as the number of samples goes to ∞.

Note that both PRM and RRT require a perfect knowledge of the map to determine distance between states and to check for collision. In this work, we train a single neural model that can perform both of these functionalities. Similar to PRM, our graph building method samples nodes from the collected trajectory data and produces a graph that can be reused for path planning. Additionally, when a path cannot be found for a given query during test time, we expand our graph by iteratively sampling the remaining nodes from the trajectory data until a path is found, which is similar to RRT. We discuss our graph building and update approaches in more detail in Chapter 3.2.2 and 3.2.3.

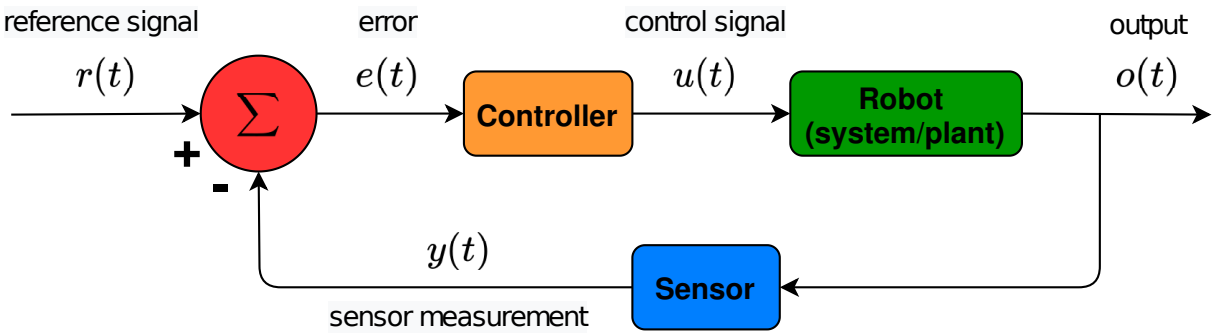


Figure 2.1. Illustration of a feedback control loop. The controller takes the error signal $e(t)$ as an input to compute the new control signal $u(t)$ at each time step t.

2.1.3. Robot Control

Control refers to the problem of determining what command to issue to our robot. A controller therefore outputs desired actions, motions, or control signals that can be executed by our robot. For example, in the case of a ground or planar-motion robot, the control signals can be the linear velocity and angular velocity (i.e., turning rate) of the robot.

A feedback controller is a type of controller that adjusts the control signal $u(t)$ based on error signals $e(t)$ from some reference point $r(t)$, so that the error decreases as the time t increases. Figure 2.1 illustrates how a feedback controller operates. Perhaps one of the most commonly used feedback controllers is the Proportional-Integral-Derivative (PID) controller [**3**]. In PID control, the control law is defined by three different terms: proportional, integral, and derivative terms. The proportional term is responsible to adjust the control signals proportionally to the error signal, and can be used to adjust the time needed to reach the reference point. The integral term adjusts the controller output based on the integral of the error over time, and can be used to attenuate steady-state bias. The derivative term adjusts the output based on how fast the error changes over time, and can be used to adjust the amount of overshoot. Concretely, the control signal $u(t)$ is computed as

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt}, \tag{2.1.1}$$

where K_p, K_i, and K_d denote the adjustable parameters for each of the terms.

Feedback control can also be formulated as an optimization problem. That is, we aim to find the control function $u(t)$ that causes the robot system to approach a target by minimizing some cost function, while also respecting the constraints of the system. One valid method to solve this is by using dynamic programming [**9**]. For example, a method called Linear Quadratic Regulator (LQR) assumes a linear system, a linear controller, and a quadratic cost function [**53**]. Another well-known optimization-based control is the Model Predictive

Control (MPC) [**35**], which solves the optimization problem at every single time step, and only applies the computed control signal for one time step.

Whereas conventionally, control methods are carried out in the robot's positional space, feedback control methods can also be applied to the image domain. This class of methods where a controller is given a starting image and a goal image as inputs is called *visual servoing* [**49**]. Visual servoing typically computes the error signal by comparing the image features from the two images. For example, given the camera parameters and features from an image pair, we can geometrically compute the pose of where the goal image is taken relative to the starting image. We can then compute the required control signals that will bring us closer to the goal location. In this proposed topological visual navigation work, our controller can be thought of as a learning-based visual servo controller which combines a pose estimation model and a feedback controller. While the controller is only able to reach a nearby goal, its combination with graph planning enables the system to perform long-distance navigation tasks. We discuss our controller in Chapter 3.2.1.

2.1.4. Belief Update

Belief update is the task to updating our belief about the world after receiving new information. In the context of robotics, this means to update the belief state by incorporating noisy sensor measurements $z_{0:t}$:

$$bel(x_t) \triangleq p(x_t|z_{0:t}), \tag{2.1.2}$$

where x_t is the state random variable at time t and $z_{0:t}$ are the state measurements seen so far up to time t, which we may obtain from a sensor.

One general framework to do this is via Bayesian inference. To simplify the problem, we often assume that given the current state x_t, the current measurement z_t is conditionally independent of past measurements (i.e., $p(z_t|x_t, z_{0:t-1}|) = p(z_t|x_t)$). Thus, we can directly apply the Bayes rule to compute the posterior $p(x_t|z_t)$:

$$p(x_t|z_t) = \frac{p(z_t|x_t)p(x_t)}{p(z_t)}, \tag{2.1.3}$$

where $p(z_t|x_t)$ is the likelihood or measurement model.

In practice, Bayes update is not tractable for continuous random variables, due to the marginal $p(z_t)$. Thus, an additional assumption is usually proposed to make this practical. Most commonly, if we assume that the measurement model and the prior are Gaussian, the posterior is also Gaussian. This family of approaches is also often referred as the *Gaussian filters* [**105**]. For a single measurement z_t, the closed-form solution for the posterior is:

$$\mu_n = \frac{\sigma^2}{\sigma_0^2 + \sigma^2}\mu_0 + \frac{\sigma_0^2}{\sigma_0^2 + \sigma^2} z_t,$$
$$\sigma_n = \left(\frac{1}{\sigma_0^2} + \frac{1}{\sigma^2}\right)^{-1}, \tag{2.1.4}$$

where (μ_0, σ_0^2) are the parameters of the prior, σ^2 is the variance of the measurement model, and (μ_n, σ_n^2) are the parameters of the posterior. In fact, one particular implementation of Gaussian filters known as the *Kalman filter* [**52**] has been widely used in many robotics applications, including in the Project Apollo [**41**].

In this work, we apply the discrete Bayes update to update the connectivity of the graph that we build for navigation purposes. In addition, since the edge weights of the graph are continuous variables, we also use the Gaussian filter to update them. For further in-depth discussion about the topic of state estimation and belief update in robotics, we refer readers to Thrun *et al.* [**105**] and Barfoot [**6**].

2.1.5. Map-based Navigation

Classically, visual navigation can be categorized into map-based and mapless navigation [**23, 13**]. In map-based approaches, we explore the environment, integrate the observed data, and build a map that can be used for navigation. Once a map is built, we can then localize the robot and plan a feasible path to reach the goal using classical search algorithms such as A* [**44**] and Dijkstra's [**24**]. Thus, navigation tasks are decomposed into perception, mapping, localization, planning, and control problems. The types of map can further be categorized into metric, topological, and topological-metric [**13**]. In contrast, mapless approaches directly infer a control command from an observation.

A metric map is built using a predefined coordinate system and stores metric information. For example, one may represent a map using an occupancy grid [**28**] where each grid cell is associated with a point coordinate and an indicator of the free space. To build a metric map, one typically resorts to SLAM or CML techniques [**63, 100**]. Building a metric map is challenging because whenever we take a sensor measurement, we need to know its location, and to combine them with techniques such as the Iterative Closest Point (ICP) [**11**] algorithm. It is even more challenging in practice since real world sensors are noisy, requiring us to use filtering techniques such as the Kalman filter [**52, 50**]. When we constrain ourselves to only using image measurements, we also need to extract features from the collected images that can be aggregated into a map. In the past, methods such as Scale-Invariant Image Features (SIFT) [**69, 68**], Features from Accelerated Segment Test (FAST) [**88**], Speeded Up Robust Features (SURF) [**7**], Binary Robust Independent Elementary Features (BRIEF) [**17**], and Oriented FAST and Rotated BRIEF (ORB) [**90**] were often used to extract image features

for map building [**95, 30, 47, 78**]. Nevertheless, building a metric map is not within the scope of this work. For recent reviews of SLAM and CML techniques we refer readers to Cadena *et al.* [**16**], Taketomi *et al.* [**102**], or Rosen *et al.* [**87**].

Another type of map – which is the focus of this work – is the topological map representation. A topological map is a graph where the nodes represent some distinctive feature of a location within an environment (e.g., poses, objects, places [**19, 122, 8**]), and the edges represent the traversability between the nodes. The edge can encode different things depending on what the node represents. For example, if a state corresponds to a coordinate in two-dimensional space, then a simple Euclidean distance between two points may be used. Unlike a metric map representation, a topological map does not usually contain information such as free space. Thus, the navigation system needs to consider other strategies to avoid obstacles. Compared to metric representations, topological maps tend to be more compact, thus require less storage and allow the robot to perform faster computation during localization and planning. However, the sequence of states that the robot observes during test time will be different than the nodes on the graph. Therefore, the system needs to be able to associate new observations with the nodes on the graph robustly.

A topological-metric map is a hybrid between metric and topological maps, where the construction of both maps are consistent with each other. One approach to do this is to partition the grid map into different regions within a Voronoi graph [**104, 8, 33**]. We can then use the graph for planning, and enjoy the benefits of using a purely topological map for navigation. While intriguing, this class of approaches still requires one to build a metric map, which as we discussed previously, is challenging and computationally expensive.

In this work, we focus on topological visual navigation where the graph is built in the image space. We show that we can build a graph using a learned reachability and pose estimator model, and use the built map to perform image goal visual navigation tasks both in the simulation and the real world. While the edges in our graph encode metric information that are inferred with the learned model, the graph is not derived from a metric map. Furthermore, we show how we can update our graph based on what our agent observes during navigation to improve lifelong navigation performance.

2.1.6. Deep Learning for Robotics

Deep learning [**62, 38**] is a family of machine learning techniques that is based on neural networks. Neural networks are parametric models that can be trained to approximate a function. To train neural networks, one needs to first specify a loss function that measures the performance of the model. For example, in supervised learning, the loss function measures how far the model predictions are from the ground truth labels. Typically, for a differentiable loss function, one can then compute the gradient of the loss function with respect to

the parameters of the neural network using the backpropagation algorithm [**91**]. Once the gradient is known, we iteratively update the neural network parameters using optimization algorithms such as stochastic gradient methods [**15**].

When dealing with image inputs, we usually resort to a class of neural networks known as Convolutional Neural Network (CNN). CNNs employ convolution operations and the parameters of the neural network are made of convolutional filters. Ever since the remarkable achievements of CNNs in solving image classification tasks [**57**], deep learning has also been applied to solve various computer vision tasks such as object detection [**37, 36, 86, 85**], visual object tracking [**46, 10, 108**], image segmentation [**97, 45**], and visual odometry [**110, 64**]. Since many robotics tasks are heavily related to vision tasks, we can directly apply these techniques to solve various robotics tasks. In fact, deep learning has also been used to develop mapless end-to-end visual navigation techniques that directly predict a control signal from image inputs [**12, 123**].

Deep learning has also been heavily used together with Reinforcement Learning (RL) to solve various robotics tasks. In RL, an agent learns how to perform a task via learning a value function or a policy by acting in the world and maximizing its reward. The value function is a measure of how good a state is for the agent to be in to maximize its future reward, while a policy defines how the agent should behave in order to maximize its future reward. With deep RL, we parametrize the value function or the policy with deep neural networks. RL, however, is not within the scope of this work. Thus, we refer readers to Sutton and Barto [**101**] for a comprehensive background of RL.

We will see in Chapter 2.2 that many recent visual navigation techniques rely on deep learning at their core. In this work, we apply deep learning to determine whether two images are taken from locations that are nearby, and to predict the relative pose difference between the two frames. We then use this model to build a graph and to provide our controller with an intermediate waypoint during navigation. We show that we can solve complex visual navigation tasks with this hybrid approach.

2.2. Related Work

2.2.1. Classical Visual Navigation

As discussed in Chapter 2.1.5, classical visual navigation approaches can be categorized into map-based and mapless methods. One class of map-based approaches is to build a detailed geometric description of an environment known as a metric map, and use it to guide the robot during navigation. Various metric-map navigation systems that rely on range sensors have been developed [**26, 27, 14, 25**]. However, they have difficulty distinguishing between hallways and different doors along a hallway [**56**]. While this problem may be

alleviated using image sensors, image signals are ambiguous and hard to interpret, making localization and data association more difficult to do in map building.

The Stanford cart early visual navigation stack builds a metric map by tracking and correlating image features acquired from its stereo camera [**76**]. A robotic museum guide system called MINERVA [**103, 22**] combines cameras and laser sensors to build occupancy and ceiling maps of the museum. Curious George [**73, 32**] is another indoor navigation system that builds an occupancy grid map with cameras and laser sensors, that is also equipped with an object recognition model to perform visual search tasks.

An alternative to the metric map is the topological map representation. Kortenkamp and Weymouth [**56**] build a topological map of gateways' images, where the features of the images can be extracted for localization and navigation purposes. Another valid class of approaches is to pair visual servoing with image graphs where the nodes correspond to image features [**71, 72, 113**]. Kuipers and Beeson [**59**] cluster the collected trajectory images based on their visual features, and build a graph where the nodes correspond to each cluster, and the edges are determined based on their temporal occurrence during trajectory collection. Vasudevan *et al.* [**109**] build a probabilistic topological map of detected objects and doors. During navigation, localization is then done based on the detected objects.

Visual navigation approaches that rely on hybrid topological and metric maps also abound. The Tour model [**58**] builds an image-based graph with high level actions that connect the nodes. In addition, the graph is accompanied with a metric information such as the location of the nodes and the magnitude of the actions. Tomatis *et al.* [**107**] couple a metric map with a landmark-based graph where landmarks are defined as room corners and openings. Other techniques build a dense grid map which is then partitioned into different regions using Voronoi diagram [**4**] to build the topological map [**104, 8, 33**]. Visual Teach and Repeat (VT&R) [**34**] collects sequence of stereo images and clusters them into a set of overlapping submaps. Each submap is made of feature keypoints such that they are sufficient to compute the camera pose and position when the robot observes a new image during the repeat pass to enable localization and navigation.

As previously discussed, building a detailed metric map usually requires heavy engineering and costly equipment. Classical navigation methods that rely only on vision typically are also constrained by the limitation of "hand-crafted" visual features when building the maps and during navigation. Furthermore, existing topological maps are typically built without considering the capability of the controller. In contrast, our map building method is controller-dependent, thus reducing the amount of spurious edges that are not actually traversable by the controller. In this work, we combine classical navigation with learning-based approaches to enable lifelong topological visual navigation.

2.2.2. Learning-based Visual Navigation

As machine learning tools are becoming ever more powerful, we are seeing the emergence of new techniques that learn navigation behaviors directly from sensor inputs without explicitly building the map of the environment. Today, this is typically done via deep and/or reinforcement learning. Nevertheless, this idea is not new and has been explored long before the era of deep learning. Autonomous Land Vehicle In a Neural Network (ALVINN) [**84**] is one of the early applications of artificial neural networks in autonomous driving. Concretely, ALVINN trains a shallow neural network to learn a road following behavior directly from image and laser range finder inputs.

Deep learning has demonstrated even more promising results in solving visual navigation tasks. For example, CNNs have been successfully trained end-to-end to directly map camera frames into steering commands to perform autonomous driving [**12**]. Another approach leverages learning-based object detection and segmentation models to produce object bounding boxes and segmentations maps, which are then used as the inputs to train a policy network via imitation learning [**77**]. The cognitive mapping and planning approach jointly learns a neural mapper and a planner, where the mapper integrates observations to build a map, while the planner uses the latest map to plan a path to the goal and take actions to follow the path [**43**]. PoliNet [**48**] performs visual path following task with learning-based MPC. Learning-based waypoint navigation [**5**] is another hybrid approach that combines an LQR feedback controller with learning-based waypoint prediction.

Deep RL techniques were also proposed as promising approaches for developing visual navigation policies. For example, actor-critic style models have been used to train an RL policy for solving goal-conditioned navigation tasks [**75, 123, 94**]. An RL policy for object goal navigation tasks can also be trained with the help of a pretrained object recognition model as the reward provider [**120, 119, 117**]. SplitNet [**39**] further shows that an RL-based navigation agent can be improved by separating visual representation and policy learning. Self-adaptive visual navigation [**114**] proposes a method to adjust an RL navigation policy during test time via self-supervised learning and meta-learning. A deep RL policy can also be augmented with priors of how objects in the world are related to each other to improve navigation performance [**118**]. Furthermore, various RL policies have also been developed for language-based navigation tasks [**2, 20, 40, 111**].

Although learning-based methods have shown promising results, they still struggle to solve long-distance navigation tasks. Furthermore, training RL policies requires significant computation and time, and thus is impractical to do in real-world environments. While many simulators have been created to support the development of embodied agents [**55, 116, 93, 98, 21**], models trained with simulated data usually do not transfer well to the

real-world [**106**]. In contrast, our method combines a feedback position-based controller with a learning-based pose estimation model that can be finetuned in a target domain.

2.2.3. Learning-based Topological Navigation

As discussed in Chapter 2.2.1, the idea of representing robot motion as a sequence of images with the help of topological map is not novel. However, classical topological navigation approaches struggle especially when restricted to only use camera data. Recently, learning-based approaches have further pushed the boundary of visual topological navigation due to their strengths in understanding images.

The most comparable approaches to our method are Semi-Parametric Topological Memory (SPTM) [**92**] and Visual Navigation with Goals (ViNG) [**96**]. SPTM trains a binary classifier to predict if two frames are close as measured by temporal distance during trajectory collection, and uses it to build a graph. Similar to SPTM, ViNG determines the connectivity between two frames based on their temporal difference. However, instead of a binary classifier, ViNG regresses the number of steps required to move from one image to the other, and uses this to weigh the graph edges. We compare our method with both SPTM and ViNG, and we discuss them in more details in Chapter 2.3.1 and 2.3.2. As a common concern with SPTM and ViNG, they build a navigation graph using *all* images within the collected trajectories, which poses scalability issues. Also, these methods build the graph without considering the capability of their controller, which may result in edges that are not actually traversable. Moreover, by solely relying on temporal distance within collected trajectories, these methods are blind to image pairs that are spatially close, yet temporally far within the explored trajectories circumstantially.

Bayesian Relational Memory (BRM) [**115**] builds a fully-connected graph where nodes and edges correspond to room types and the probability of room connectivity. BRM trains a classifier that predicts the probability of an image belonging to different room types. As the agent navigates, BRM uses its learned classifier to infer the type of rooms that it has seen so far. BRM assumes that room types that are seen within small temporal distance are connected, and then refines the edge weights of the graph using Bayesian updates. Our graph update strategy also uses Bayesian updates at its core, but we determine the connectivity measurement based on an actual attempt in traversing an edge. In addition, we also introduce new nodes to the graph to improve lifelong navigation performance.

Hallucinative Topological Memory (HTM) [**66**] trains a generative model to generate the possible states of an environment given information such as the floor plan or image of the environment. This model is used to replace the trajectory collection phase previously depicted in Figure 1.2. The generated states are then used to build a graph that represents

the environment, where Contrastive Predictive Coding (CPC) [**82**] is used to differentiate reachable image pairs from non-reachable ones, as well as to weigh the edges.

Meng *et al.* [**74**] proposed a controller-dependent graph building method. At its core, a classifier is trained based on the controller rollout outcome in simulation to predict if an image pair is reachable. To build the graph, this classifier model is used to first sparsify highly reachable redundant nodes in the collected trajectories. Then, remaining nodes are connected with edges weighted by predicted reachability scores. As a drawback, it is impractical to finetune this reachability model in the real world, as it would require empirically unsafe controller rollouts between location pairs.

There are also methods that rely on an actor-critic RL policy such as Latent Embeddings for Abstracted Planning (LEAP) [**81**], Search on Replay Buffer (SoRB) [**31**], and Sparse Graphical Memory (SGM) [**29**], which evaluate node connectivity using the value function predicted by the critic. In addition, SGM proposes to sparsify the graph by only adding perceptually distinct nodes, merging nodes with shared connections, limiting the number of edges per node, and removing edges predicted as not traversable during test time. However, these sparsification strategies may lead to excessive false negative edges. Additionally, such simulation-trained RL policies may not transfer well to real-world environments.

While we use the same framework for topological navigation shown in Figure 1.2, we highlight several design decisions that improve the overall performance of the system. These include: the learned model, data collection procedure, graph construction, what the edges of the graph encode, the controller, and graph update strategy. Table 2.2 summarizes the differences between various learning-based topological navigation techniques, and we detail our approach in Chapter 3.

Table 2.2. Comparison of various learning-based topological navigation methods for image-goal navigation tasks.

	SPTM [92]	**HTM** [66]	**Meng *et al.*** [74]	**LEAP** [81]	**SoRB** [31]	**SGM** [29]	**ViNG** [96]	**Ours**
Query type	Multi-query	Multi-query	Multi-query	Single-query	Multi-query	Multi-query	Multi-query	Multi-query
Controller	Inverse dynamics	Inverse dynamics	Potential-based	RL	RL	RL	Position-based	Position-based
Trajectory type	Sequential	Model-generated	Sequential	Model-generated	Any	Any	Any	Any
Node selection	All nodes	All nodes	Incrementally selected	Optimization-based	All nodes	Incrementally selected	All nodes	Sampling-based
Edge weight	Unweighted, temporal-based	Contrastive loss	Reachability score	Value function	Value function	Value function	Number of steps	Geodesic distance
Path planner	Graph search	Graph search	Graph search	Optimization-based	Graph search	Graph search	Graph search	Graph search
Graph update	None	None	None	None	None	Edge pruning	None	Edge update, node addition
Model finetuning	Self-supervised	None	None	None	None	None	Self-supervised	Self-supervised

2.3. Baseline Methods

In this section, we discuss two of the learning-based topological navigation techniques that we compare against in our experiments.

2.3.1. Semi-parametric Topological Memory (SPTM)

SPTM [**92**] applies deep learning to visual topological navigation by learning a reachability estimator between two images as well as a controller. The learning components consist of retrieval and locomotion models that are parametrized by neural networks. To train these models, SPTM first rolls out a random policy to explore the training environments and collects a sequence of states and actions dataset $D = \{(o_1, a_1), ..., (o_N, a_N)\}$. The retrieval model $R(o_i, o_j)$ is a binary classifier that takes two observations (o_i, o_j) as inputs, and is trained to predict the probability of whether these two frames are taken at most l time steps away from each other (i.e., $|i - j| <= l$). To reduce the amount of noisy data, SPTM only considers the observation pairs as negative samples if they are at least $l \times m$ steps away. The locomotion model $L(o_i, o_j)$ also takes two observations as inputs, and is trained to predict what action to take (from a given set of discrete actions) such that the agent goes from one observation to the other (i.e., an inverse dynamics model). To train the locomotion model, SPTM samples two observations from D that are at most l steps apart and uses the action corresponding to the first observation o_i as the label.

During test time, the agent is given time to explore the environment and collect sequences of images, which will be used to build the graph of the environment where the nodes correspond to images. To determine the connectivity between nodes, SPTM first connects all the nodes that are at most one step apart. In addition, SPTM also creates shortcut connections between nodes whenever there is a node pair o_i and o_j where $R(o_i, o_j) > s_{shortcut}$ and $|i - j| \geq \Delta T_\ell$. To increase robustness, rather than predicting only the pair of interest at a time, SPTM takes ΔT_w nearest neighbors of o_i and o_j and form a sequence of observation pairs. To get the final score, SPTM takes the median of the predictions from the sequence. Concretely, instead of checking if $R(o_i, o_j) > s_{shortcut}$, SPTM checks if $\text{median}\{R(o_{i-\Delta T_w}, o_{j-\Delta T_w}), ..., R(o_i, o_j), ..., R(o_{i+\Delta T_w}, o_{j+\Delta T_w})\} > s_{shortcut}$.

Once the graph is built, given a query, the SPTM agent localizes itself and the goal on the graph, and plans a path using Dijkstra's algorithm [**24**]. To localize an observation on the graph, SPTM uses the retrieval model to find a node that is closest to the observation based on its output. To save computational cost, SPTM first only performs a local localization by searching only within k-nearest neighbors from the last known position of the agent. When performing local localization, SPTM also checks if the score predicted by the retrieval model is higher than s_{local}. SPTM then chooses the furthest node in the path that is within H_{min}

to H_{max} steps, and has a retrieval score of at least s_{reach} to be the intermediate subgoal for the locomotion model to reach.

2.3.2. Visual Navigation With Goals (ViNG)

Like SPTM, ViNG also trains a neural network to learn a distance function between images based on their temporal difference during trajectory collection. ViNG formulates this as a multinomial classification problem by discretizing the number of steps into d_{max} bins. As the controller, ViNG trains a network that maps two observations (o_i, o_j) to the relative pose of o_j from where o_i is taken, and uses a simple proportional-derivative controller to reach the estimated pose during navigation.

To train the network, ViNG collects a dataset of a robot moving following a time-correlated random walk in various environments from Kahn *et al.* [**51**]. The dataset consists of multiple trajectories, where each trajectory is made of a sequence of observations. An image pair (o_i, o_j) is considered as a positive sample if they come from the same trajectory and if $0 \leq j - i < d_{max}$. To generate negative samples, ViNG takes any other pairs that do not meet these criteria. Both the positive and negative samples dataset are then used to train the distance estimator to minimize the cross entropy loss, where all the negative samples are labeled as class d_{max}. The pose estimator is trained exclusively using the positive samples dataset to minimize the L_2 regression loss.

To build the graph, ViNG considers all observations in the trajectory and connects them using the predicted distance as the edge weight. To reduce the number of edges in the graph, ViNG removes edges where the distance is less than a constant $\delta_{sparsify}$. During test time, the ViNG agent takes its current observation at each time step, adds it to the graph according to the distance estimator, plans the path to reach the goal using Dijkstra's algorithm [**24**], and chooses the first node within the path to be the intermediate subgoal.

Chapter 3

Lifelong Topological Visual Navigation

We discuss our methodology in this section. We start by formalizing the navigation problem and setting in Chapter 3.1. We then discuss each of the components of our navigation system in Chapter 3.2, as well as main differences of our approach compared to other existing learning-based topological navigation methods. These include the controller, the reachability and waypoint prediction model (Chapter 3.2.1), graph building algorithm (Chapter 3.2.2), graph update procedure (Chapter 3.2.3), and real world finetuning (Chapter 3.2.4).

3.1. Problem Formulation

We focus our work on visual navigation tasks with image goals. When we deploy our agent in a target environment, we first execute a trajectory collection phase to obtain a sequence of RGBD images $\mathcal{T} = \{o_1, ..., o_N\}$. We then use $\mathcal{T}$ to build a graph $G = (V,E)$, where vertices V are a subset of the collected images, and directed edges E are weighted by some distance function $d(o_i, o_j)$. The distance function can be determined based on what we value most during navigation such as safety or navigation efficiency.

During navigation, we present the agent with a query represented as an RGBD goal image o_g. The agent first localizes itself on the graph based on its current observation o_c, and plans a path to reach o_g. From the planned path, the agent picks a subgoal observation o_{sg} and moves towards it using its controller. After executing the control command, the agent relocalizes itself on the graph using its latest observation, and updates the graph. These processes are repeated until the robot reaches o_g.

3.2. Components of the Proposed Method

3.2.1. Controller, Reachability, and Waypoint Prediction Model

We train a CNN $f : \mathcal{I} \times \mathcal{I} \mapsto (0,1) \times SE(2)$ that takes two images and jointly predicts the probability of reachability $\hat{r}$ from one image to the other, and the relative transformation in

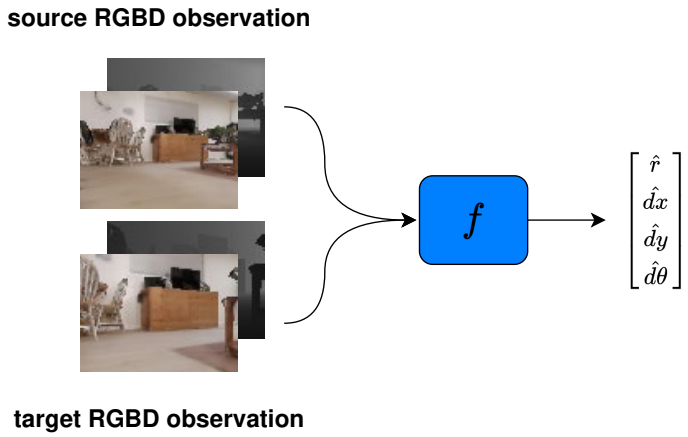


Figure 3.1. Illustration of our model. Our model takes two RGBD images and predicts a reachability score $\hat{r}$ and a waypoint $\hat{w} = [\hat{dx}, \hat{dy}, \hat{d\theta}]$.

$SE(2)$ that relates them (i.e., a waypoint $\hat{w} = [\hat{dx}, \hat{dy}, \hat{d\theta}]$) as illustrated in Figure 3.1. For practicality, we only consider $\hat{w}$ to be valid for reachable image pairs. We train our model by minimizing the binary cross-entropy loss for reachability, L_2 loss for the relative position, and L_1 loss for the relative rotation. Concretely, the loss functions are as follow [1]:

$$
\begin{aligned}
L_r(r, \hat{r}) &= -(r \log(\hat{r}) + (1 - r) \log(1 - \hat{r})), \\
L_p(dx, dy, \hat{dx}, \hat{dy}) &= ||[dx, dy] - [\hat{dx}, \hat{dy}]||_2, \\
L_\theta(d\theta, \hat{d\theta}) &= |\sin(d\theta) - \sin(\hat{d\theta})| + |\cos(d\theta) - \cos(\hat{d\theta})|, \\
L_{total}(r, dx, dy, d\theta, \hat{r}, \hat{dx}, \hat{dy}, \hat{d\theta}) &= L_r(r, \hat{r}) + \alpha L_p(dx, dy, \hat{dx}, \hat{dy}) + \beta L_\theta(d\theta, \hat{d\theta}), \qquad (3.2.1)
\end{aligned}
$$

where $L_r(r, \hat{r})$ is the reachability loss, $L_p(dx, dy, \hat{dx}, \hat{dy})$ is the position loss, $L_\theta(d\theta, \hat{d\theta})$ is the rotation loss. Here, the variables r, dx, dy, d_θ are the ground truth labels for the reachability and the relative waypoint predictions, whereas α and β are hyperparameters. After training, we also calibrate the reachability estimator using Platt scaling [**42**] on a validation dataset.

In this work, we use a position-based feedback controller to execute predicted waypoints. Similar to Meng *et al.* [**74**], we define node-to-node reachability to be controller-dependent. In our case, we assume a control strategy based on motion primitives, namely straight-line motion and in-place rotation, which imposes simple geometric constraints. Specifically, two nodes should be connected when:

(1) The visual overlap ratio between the two images, p, is larger than P_{min}, computed based on depth data;

1. Note that $L_r(r, \hat{r})$ may not be numerically stable. Fortunately, standard machine learning libraries such as PyTorch [**83**] normally have the numerically stable implementation of the loss function.

(2) The ratio of the shortest feasible path length over Euclidean distance between the poses, r_d, is larger than R_{min}, to filter out obstacle-laden paths;

(3) The target pose is visible in the initial image, so that our model can visually determine reachability;

(4) The Euclidean distance to the target is less than E_{max}, and the relative yaw is less than Θ_{max}.

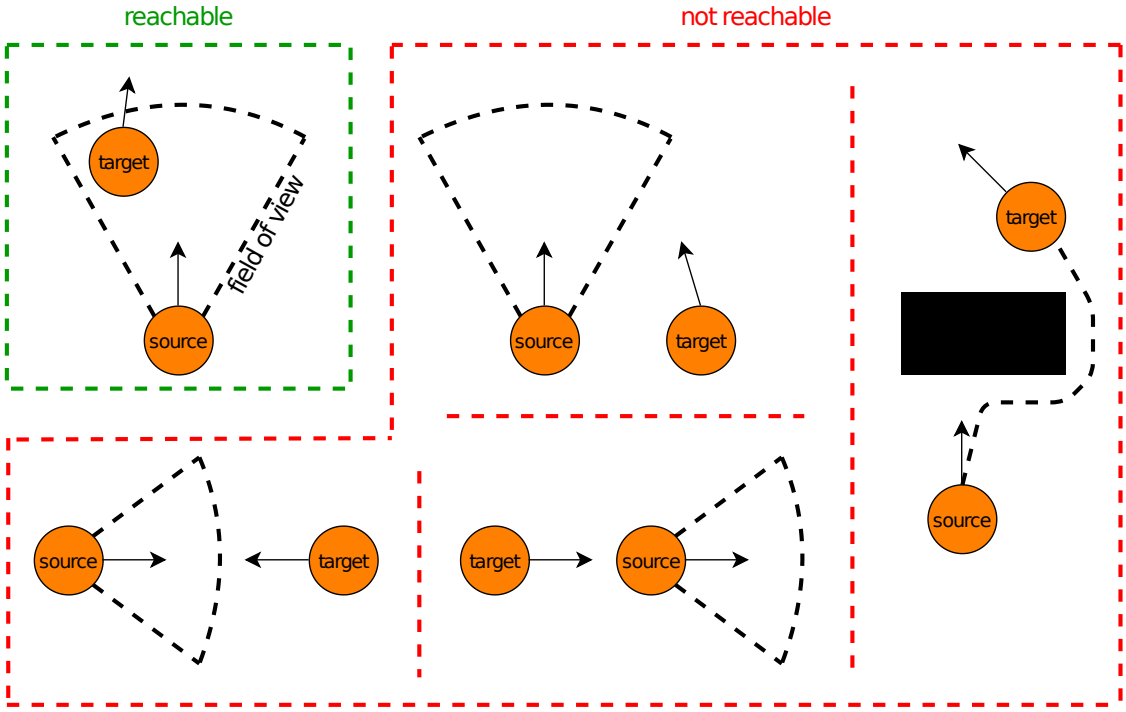


Figure 3.2. Illustration of reachable and non-reachable situations.

During training, we define o_j to be reachable only if it is in front of o_i. Yet, when navigating, the robot can move from o_j to o_i by following the predicted waypoint w in reverse. Figure 3.2 illustrates various reachable and non-reachable situations during data collection.

3.2.2. Graph Building, Planning, and Localization

Instead of using all images in $\mathcal{T}$, we build the graph G incrementally, as seen in Algorithm 1. We initialize G as a randomly drawn node $o \in \mathcal{T}$. Then, in each iteration, we sample a node $o_{rand} \in \mathcal{T}$, and use our model to check if it can be merged with or connected to existing graph vertices, and if so, remove it from $\mathcal{T}$. Concretely, let the geodesic distance of a predicted waypoint be $d(\hat{w}) = ||\log T(\hat{w})||_F$ [**6**], where $T(\cdot)$ converts a waypoint into its homogeneous transformation matrix representation in $SE(2)$ [2], and $||\cdot||_F$ computes the

2. A matrix in the form of $\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}_{1\times 2} & 1 \end{bmatrix}$, where $\mathbf{R} \in \mathbb{R}^{2\times 2}$ denotes the rotation matrix, and $\mathbf{t} \in \mathbb{R}^2$ denotes the translation vector.

Algorithm 1 Graph Building

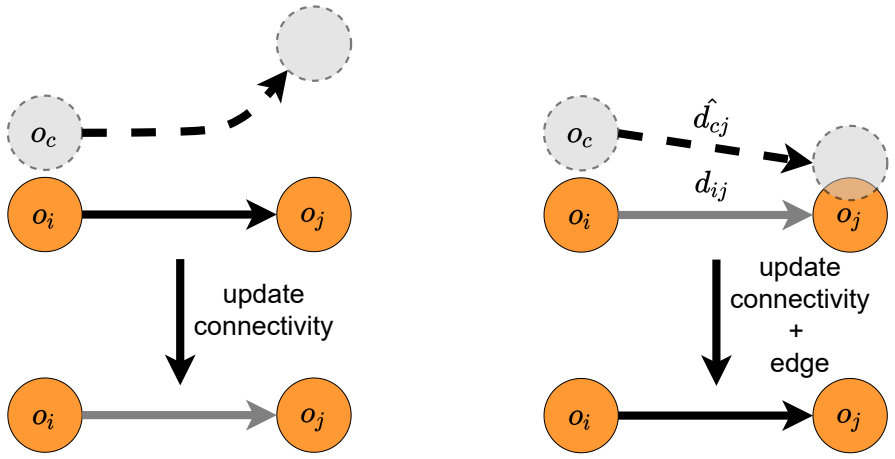
Input: trajectory data $\mathcal{T}$, reachability score threshold ϵ, Euclidean distance threshold δ_e, yaw threshold δ_θ, and merging threshold D_{merge}
Output: graph $G = (V,E)$
Initialize: $V = \{o \in \mathcal{T}\}$, $E = \varnothing$
while there are still connectable nodes in $\mathcal{T}$ **do**
  Sample random node $o_{rand} \in \mathcal{T}$
  for all $o_j \in V$ **do**
    $(\hat{r_f}, \hat{w_f}) \leftarrow f(o_{rand}, o_j)$
    $(\hat{r_b}, \hat{w_b}) \leftarrow f(o_j, o_{rand})$
    if $\hat{r_f} > \hat{r_b}$ **and** $(\hat{r_f}, \hat{w_f})$ satisfy connectivity test **then**
      Connect o_{rand} to o_j with $d(\hat{w_f})$ as the weight
    end if
    if $\hat{r_b} > \hat{r_f}$ **and** $(\hat{r_b}, \hat{w_b})$ satisfy connectivity test **then**
      Connect o_j to o_{rand} with $d(\hat{w_b})$ as the weight
    end if
  end for
  if o_{rand} is connectable or mergeable **then**
    remove o_{rand} from $\mathcal{T}$
  end if
end while
Return: G

Frobenius norm. The node o_{rand} is mergeable to an existing graph vertex if $d(\hat{w}) < D_{merge}$ in either direction. To determine node-to-node connectivity, we check if reachability $\hat{r} > \epsilon$, the Euclidean distance $e = ||[\hat{dx}, \hat{dy}]||_2 < \delta_e$, and the relative yaw angle $\hat{d\theta} < \delta_\theta$. If o_{rand} is deemed connectable to graph nodes, each new edge is assigned $d(\hat{w})$ as weight.

Once the graph is built, we use Dijkstra's algorithm [**24**] to find a path to a given goal vertex, and select the first node in the path as subgoal o_{sg}. To localize the agent's current observation o_c on the graph, we use our model to compare pairwise geodesic distances between o_c and all nodes in the graph, and identify the closest vertex. We also require the reachability score between o_c and the closest node to be larger than ϵ_ℓ, and the geodesic distance between them to be smaller than δ_ℓ. To save computational cost, we first attempt to localize locally by considering only directly adjacent vertices from nodes within the last planned path, and then reverting to global localization among all graph nodes if it fails.

3.2.3. Edge Traversal and Graph Updates

To traverse an edge, once we identify o_{sg} from planning, we predict its relative waypoint from o_c, and use the controller to move towards it. Then, we update o_c to be the latest observation, and check if edge traversal was successful: either if o_c localizes to o_{sg}, or, if our model's predicted waypoint distance to o_{sg} is below D_{min}.



(a) Connectivity update

(b) Connectivity and edge updates

Figure 3.3. Illustration of graph update as the agent moves from o_i to o_j. If the agent diverges from the target, we reduce connectivity between o_i and o_j probabilistically, then prune the edge if the updated reachability score falls below R_p. Otherwise, we update both the connectivity and the edge weight based on the predicted geodesic distance $\hat{d_{cj}}$.

Since we built the graph using a predictive model, it is important to address the possibility of spurious edges. We thus propose two types of continuous graph refinements, towards lifelong navigation. Firstly, as seen in Figure 3.3, when traversing an edge, we can update both its connectivity and weight based on the success of traversal, z, modeled as a Bernoulli random variable. Letting the prior $p(x)$ be the predicted reachability score, and given an empirically-tuned likelihood parameter $p(z|x)$ of the edge's existence, we can compute the posterior $p(x|z)$ using a Bayesian update. When the agent fails to reach a subgoal, it decreases the edge's reachability score, and further prunes the edge if reachability falls below R_p. As for edge weights, we model each prior d_{ij} as a Gaussian around its geodesic distance $d(\hat{w})$ with a common variance derived empirically from our model's distance predictions for a validation dataset. Only upon success in traversing an edge from our current image o_c to o_j, we treat the predicted geodesic distance $\hat{d_{cj}}$ as a measurement, and compute the weight posterior akin to a Kalman filter update step [**52**].

As a second type of lifelong graph update, we add *new* nodes to the graph either when they are novel or when we cannot find a path to the goal. Concretely, an observation o_c is considered novel when we fail to localize it on the graph. When connecting a novel node to existing vertices, we loosen the graph building criteria, especially to accommodate adding locations around sharp turns. On the other hand, when we are unable to find a path during navigation, we iteratively sample from remaining trajectory nodes $\mathcal{T}$ and *tentatively* add them to the graph G, until a path is found. We then keep only new nodes in G that are

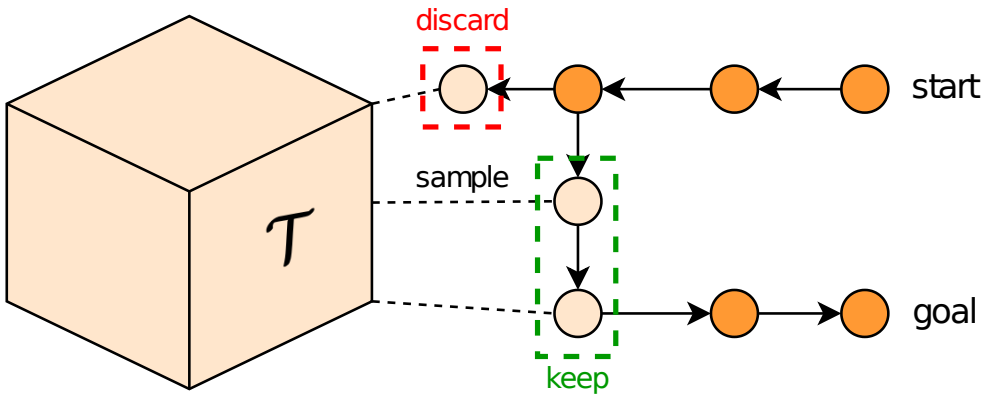


Figure 3.4. Illustration of node resampling. When we suddenly cannot find a path to reach the goal, we resample new nodes from the remaining ones in $\mathcal{T}$ and add them to the graph until we are able to find a path to reach the goal. We then keep the nodes that are within the path to reach the goal in the graph.

along the found path, while returning other tentative nodes back into $\mathcal{T}$. This process is illustrated in Figure 3.4.

3.2.4. Real World Finetuning

A key feature of our method is the ability to finetune our model in the target domain, by using the same trajectories collected to build the graph. As an added requirement for real-world finetuning, the collected trajectories must have associated pose odometry to substitute for ground-truth pose data. Thankfully, odometry is readily available from commodity robot sensors such as inertial measurement units or wheel encoders. Although other methods like ViNG can also finetune on target-domain trajectories, our model can improve performance efficiently using a small finetuning dataset, as it has already been well-trained to navigate within a diverse range of simulated environments.

For real-world trajectories, we determine node-to-node reachability using the same goal-visibility and maximum-waypoint criteria as when labeling data in simulation. Additionally, since image-to-image visual overlap and shortest feasible path length are not readily obtainable in the real-world, as a proxy criterion we take an observation pair $(o_i, o_j) \in \mathcal{T}$ where $j > i$ and check if they are separated by at most H time steps during trajectory collection. Note that constraining reachable waypoints to be temporally close is also favourable because using odometry as a supervisory signal for predicting waypoints can be noisy due to the long-term drift. Since reachable waypoints must be temporally close, the long-term pose drift should be minimal.

Chapter 4

Experiments

We now present an empirical assessment of the proposed method. First, we start by laying out the experimental setup in Chapter 4.1. We then compare navigation performance in simulation against baseline methods in Chapter 4.2. We also evaluate how the navigation performance changes as we perform graph updates over time in Chapter 4.3. Finally, we demonstrate the ability of the proposed method to navigate in the real-world setup using only a small amount of finetuning dataset in Chapter 4.4.

4.1. Setup

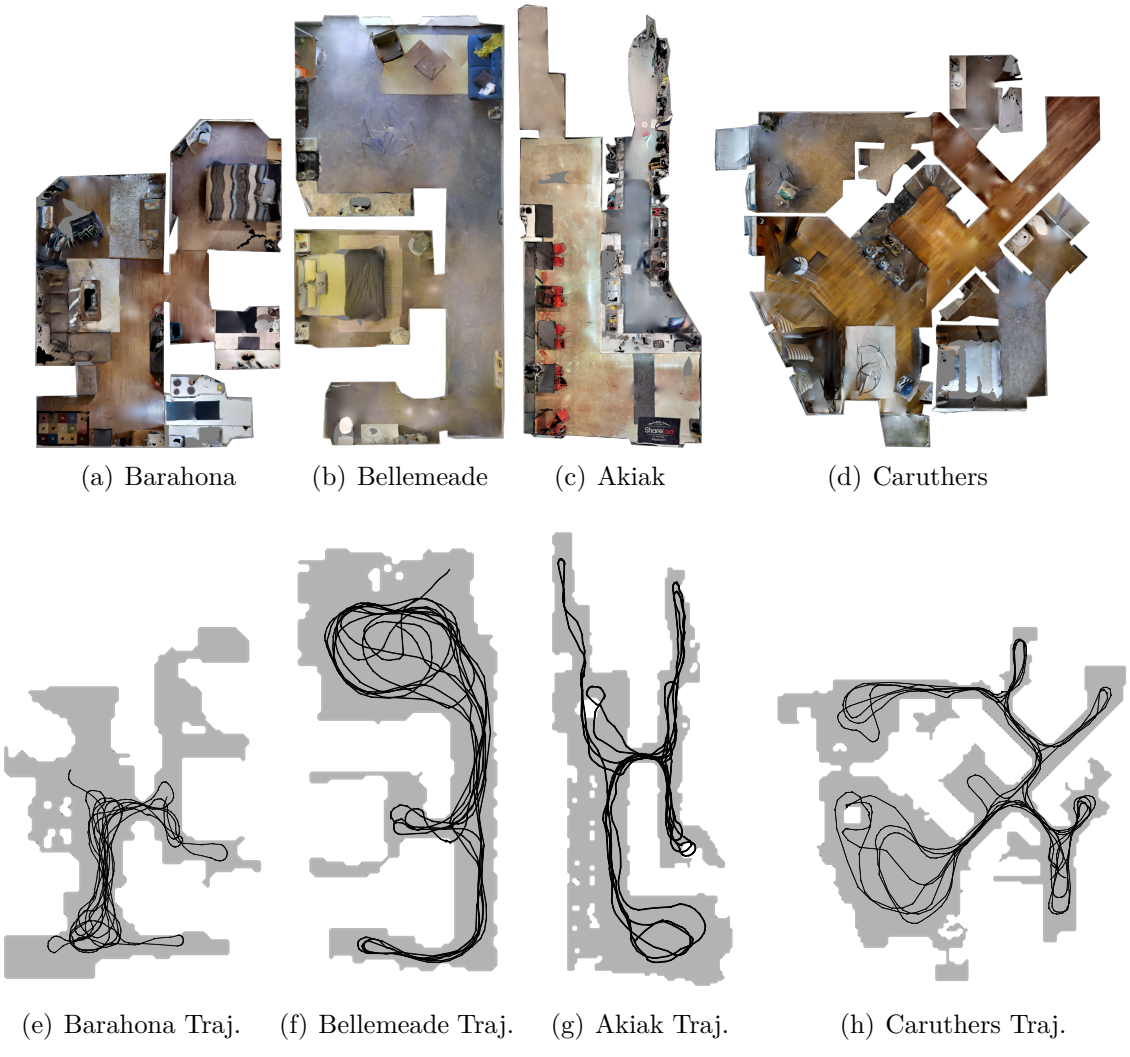
We use the Gibson [**116**] simulator to quantify navigation performance in simulation, and 10 of the interactive environments from iGibson [**98**] to generate training datasets. When generating our dataset, particularly when computing image overlap, we ignore pixels that belong to background classes (e.g., floors, walls, ceilings). To generate a dataset for our model, we set $P_{min} = 2e^{-4}$, $R_{min} = 1.1$, $E_{max} = 1.0\,\text{m}$, and $\Theta_{max} = 0.79\,\text{rad}$. For the SPTM dataset, we let the agent move around following a random policy and let $l = 15$ and $m = 5$. For the ViNG dataset, we use $d_{max} = 15$, and let the agent explore the environment by moving in forward zig-zag motion, similar to the time-correlated policy in Kahn *et al.* [**51**]. We collect 288,000 data points to train our model, and 500,000 data points for SPTM and ViNG, where the size of each RGBD observation is 96×72 pixels. All models are based on VGG16 [**99**] with different output layers (see Appendix A). We train all of the networks with AdamW optimizer [**67**]. We use the LoCoBot [**1**] in both simulated and real-world experiments, and we teleoperate it in each test map to collect trajectories for graph building.

4.1.1. Evaluation Settings

We evaluate navigation performance to reflect real-world usage: the agent should be able to navigate between any image pairs from the graph, and should not just repeat trajectories based on how the images were collected. We pick several goal images from different locations

that cover major locations in each map, and generate random test episodes. In simulation, we consider navigation as successful if the position and yaw errors from the goal pose are less than 0.72 m and 0.4 rad, respectively. We consider an episode to be a failure if the agent collides more than 20 times, and if it requires more than K simulation steps to reach the goal. For the real-world experiments, an episode is deemed successful if it finishes within 10 minutes, does not collide with the environment, and we verify that its final observation has sufficient visual overlap with the goal image.

During operations, if the agent is unable to localize itself or find a path, we rotate it in-place at 30° increments and take new observations until it recovers. To ensure fair comparison with other methods, we let SPTM to use the same controller as ours and ViNG, by coupling a position-based feedback controller with a pose estimator.



(a) Barahona (b) Bellemeade (c) Akiak (d) Caruthers

(e) Barahona Traj. (f) Bellemeade Traj. (g) Akiak Traj. (h) Caruthers Traj.

Figure 4.1. Top-down view of the test environments with the corresponding agent's trajectory during trajectory collection phase (not-to-scale).

4.2. Navigation Performance in Simulation

In this section, we compare the navigation performance of our method against SPTM and ViNG in the simulated environments. In this set of experiments, note that *we do not perform graph updates with our method*, which is evaluated separately in Chapter 4.3.

We evaluate on four test environments that are different from the training environments: Barahona (57m^2), Bellemeade (70m^2), Akiak (124m^2), and Caruthers (129m^2). For the graph building phase, we teleoperate the robot to explore roughly three to four loops around each map, resulting in 985, 1,685, 1,609, 2,243 images for Barahona, Bellemeade, Akiak, and Caruthers, respectively. Figure 4.1 visualizes the top-down view of each test environment with its corresponding trajectory. We pick 10 goal images spanning major locations in each map and generate 500 random test episodes. Given diverse map sizes, we set $K = 1{,}000$ for Barahona and Bellemeade, and $K = 2{,}000$ for Akiak and Caruthers. Since our graph building method is stochastic, we evaluate our method three times so we can report the mean and the standard deviation. We summarize the parameters for each method in Table 4.1.

As seen in Figure 4.2, our method consistently yields higher navigation success rates in all environments, especially after model finetuning. Additionally, our graphs have a significantly fewer number of nodes and edges, thus keeping planning costs practical when scaling to large environments. Therefore, compared to the baselines that use entire trajectory images to build graphs, our method produces demonstrably superior performance and efficiency.

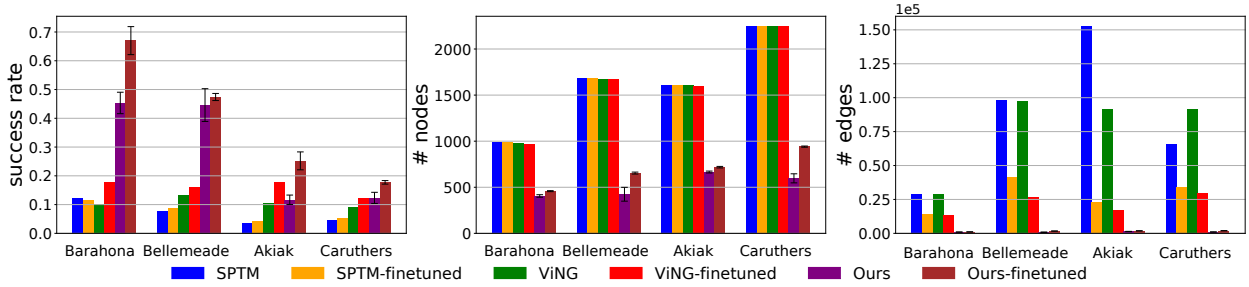


Figure 4.2. Comparison of navigation success rates and graph sizes among topological visual navigation methods in various test environments. To see visual results of our experiments, including real-world deployment videos, see our project page: https://montrealrobotics.ca/ltvn/.

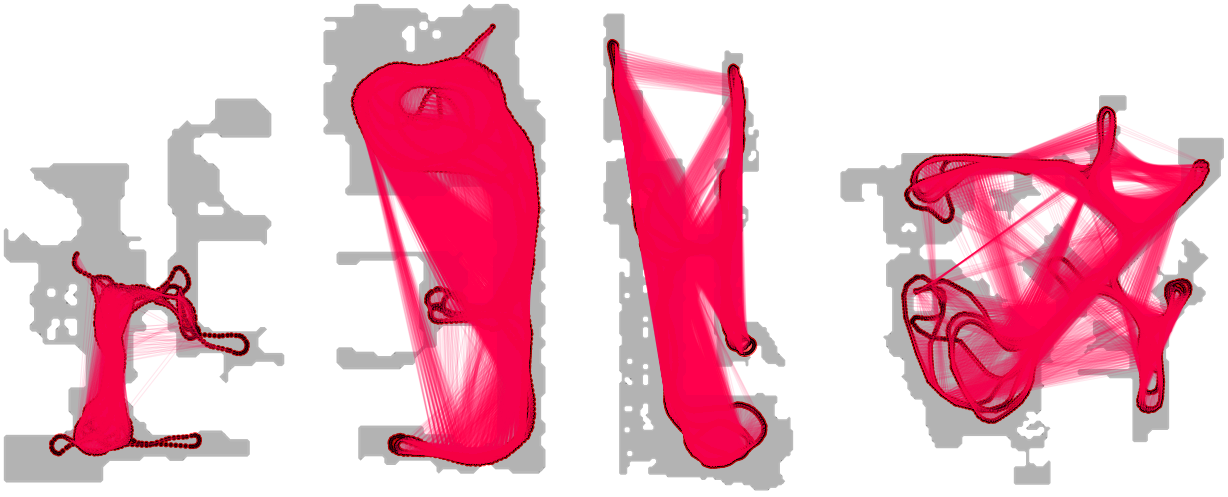
Figure 4.3 and 4.4 qualitatively compare sample graphs built using different methods. We see that our graphs have the fewest number of vertices, yet still maintain proper map coverage. Visually, our graphs also have few false positive edges through walls, and we shall later demonstrate how our graph updates can prune these in Chapter 4.3.

In comparison, the SPTM graphs also have few false positive edges. However, because edges are unweighted in SPTM, anecdotally its agent often chose spurious edges during navigation. With ViNG, since it uses the discretized temporal distance to weigh edges, its

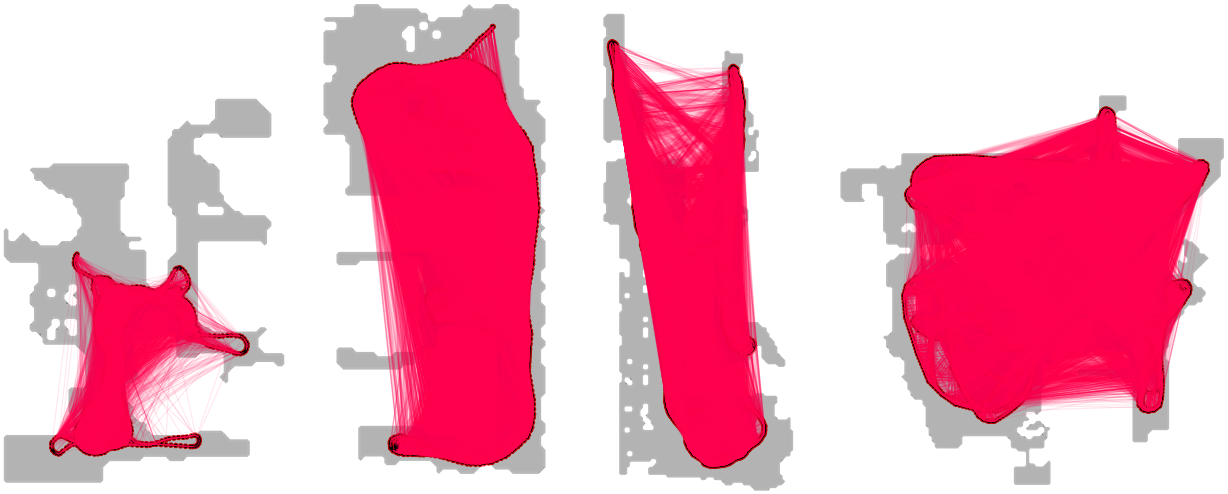
Table 4.1. Navigation parameters for each method.

Method	Parameter	Value	Description
SPTM	$s_{shortcut}$	0.9	Score threshold for determining shortcut connections.
	s_{local}	0.9	Score threshold during localization.
	s_{reach}	0.95	Score threshold when determining subgoal.
	k	5	The number of nearest neighbors to be considered during localization.
	ΔT_l	3	Minimum temporal distance between node to create shortcut connections.
	ΔT_w	6	Number of nearest neighbors to consider when making a prediction with the retrieval model.
	H_{min}	1	Minimum threshold for the subgoal search window.
	H_{max}	7	Maximum threshold for the subgoal search window.
ViNG	d_{max}	15	The maximum number of steps between two images that the model can predict.
	$\delta_{sparsify}$	3	Minimum temporal distance for graph pruning.
Ours	ϵ	0.95	Reachability score threshold for graph building.
	δ_e	0.35 m	Minimum Euclidean distance to determine connectivity during graph building.
	δ_θ	0.4 rad	Minimum relative yaw angle to determine connectivity during graph building.
	D_{merge}	0.15	Maximum geodesic distance to the existing node to be considered mergeable during graph building.
	ϵ_ℓ	0.9	Reachability score threshold for localization.
	δ_ℓ	0.4	Geodesic distance threshold for localization.

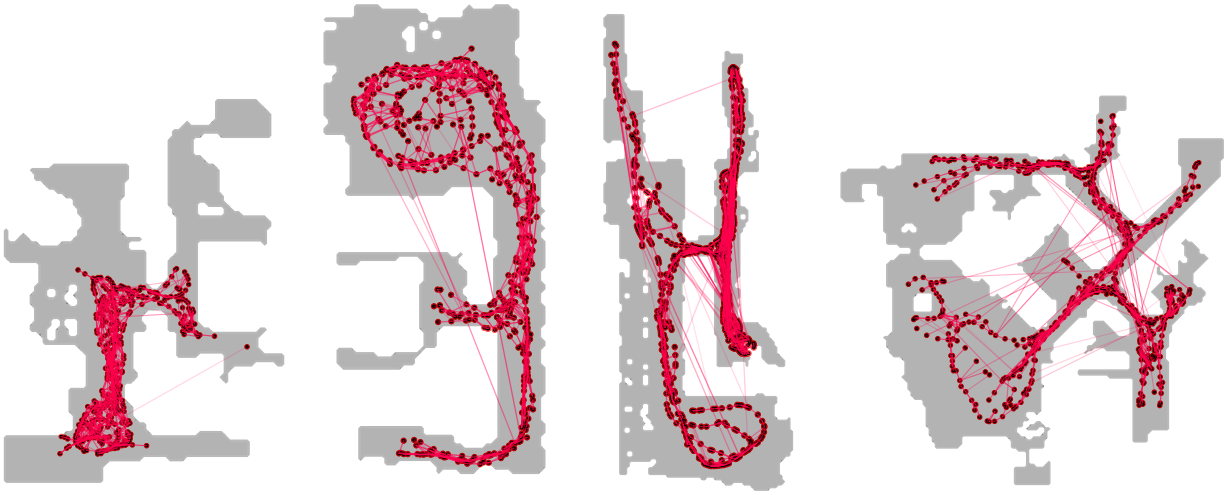
graph has many false positive edges with large weights. This becomes problematic when the goal is far away from the robot, as the planned path will likely include some spurious edges. This also leads to an unwanted behavior where the agent always outputs a path no matter how unlikely it is to reach the goal.



(a) SPTM Barahona (b) SPTM Bellemeade (c) SPTM Akiak (d) SPTM Caruthers

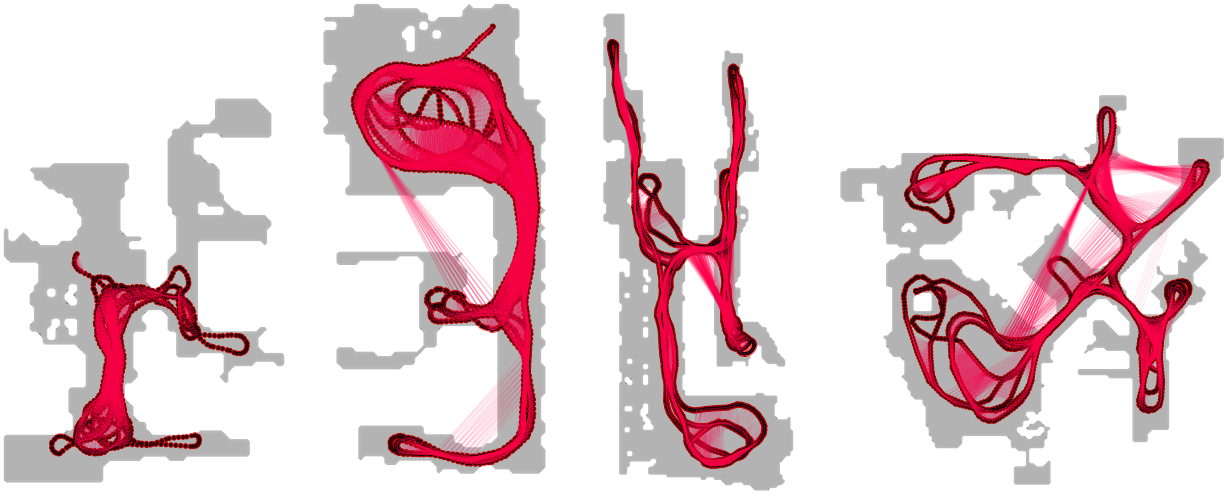


(e) ViNG Barahona (f) ViNG Bellemeade (g) ViNG Akiak (h) ViNG Caruthers

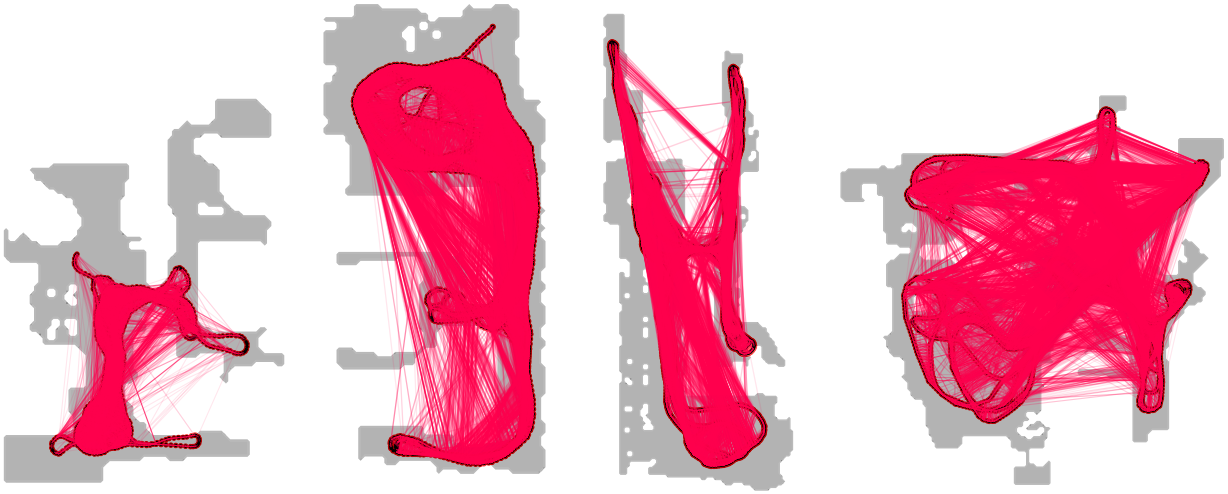


(i) Ours Barahona (j) Ours Bellemeade (k) Ours Akiak (l) Ours Caruthers

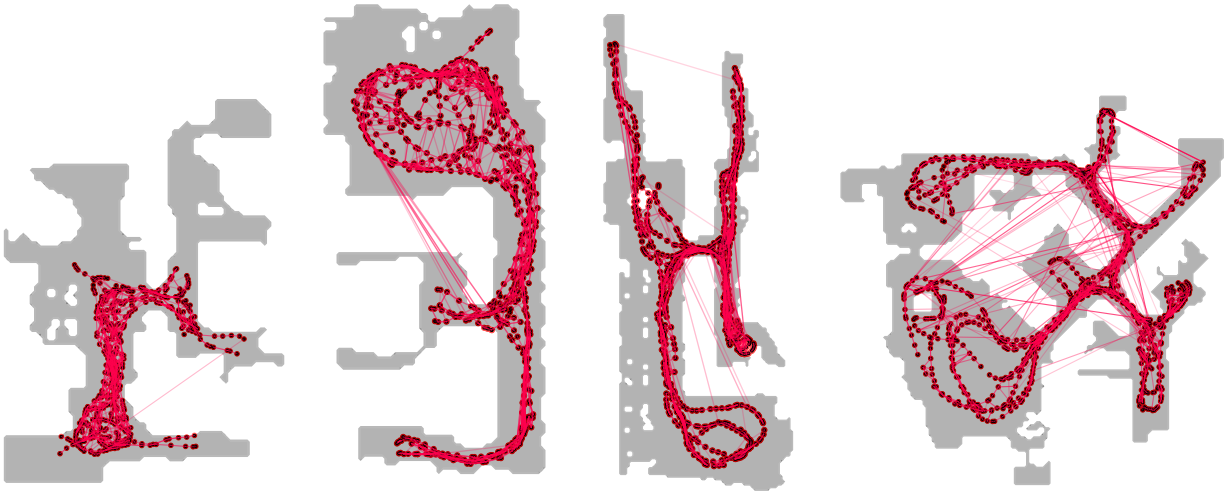
Figure 4.3. Comparison of the graphs built without any finetuning and graph updates.



(a) SPTM Barahona (b) SPTM Bellemeade (c) SPTM Akiak (d) SPTM Caruthers



(e) ViNG Barahona (f) ViNG Bellemeade (g) ViNG Akiak (h) ViNG Caruthers



(i) Ours Barahona (j) Ours Bellemeade (k) Ours Akiak (l) Ours Caruthers

Figure 4.4. Comparison of the graphs built after model finetuning.

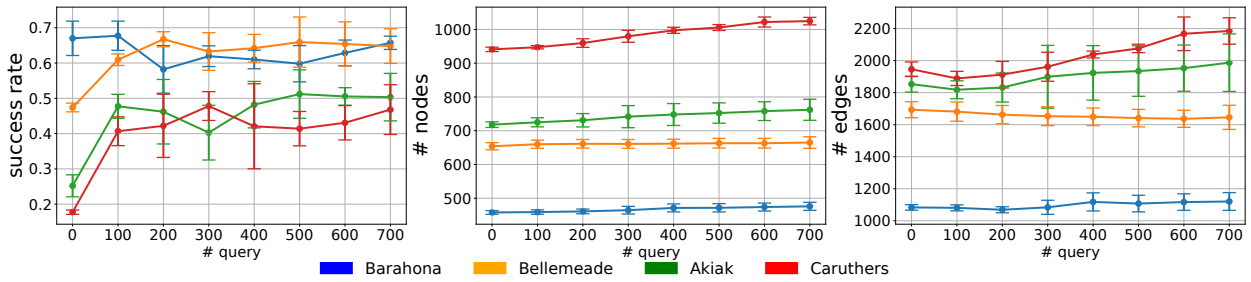


Figure 4.5. Changes in success rate, number of nodes, and number of edges as the agent performed more queries and updated its graph in each test environment.

4.3. Lifelong Improvement

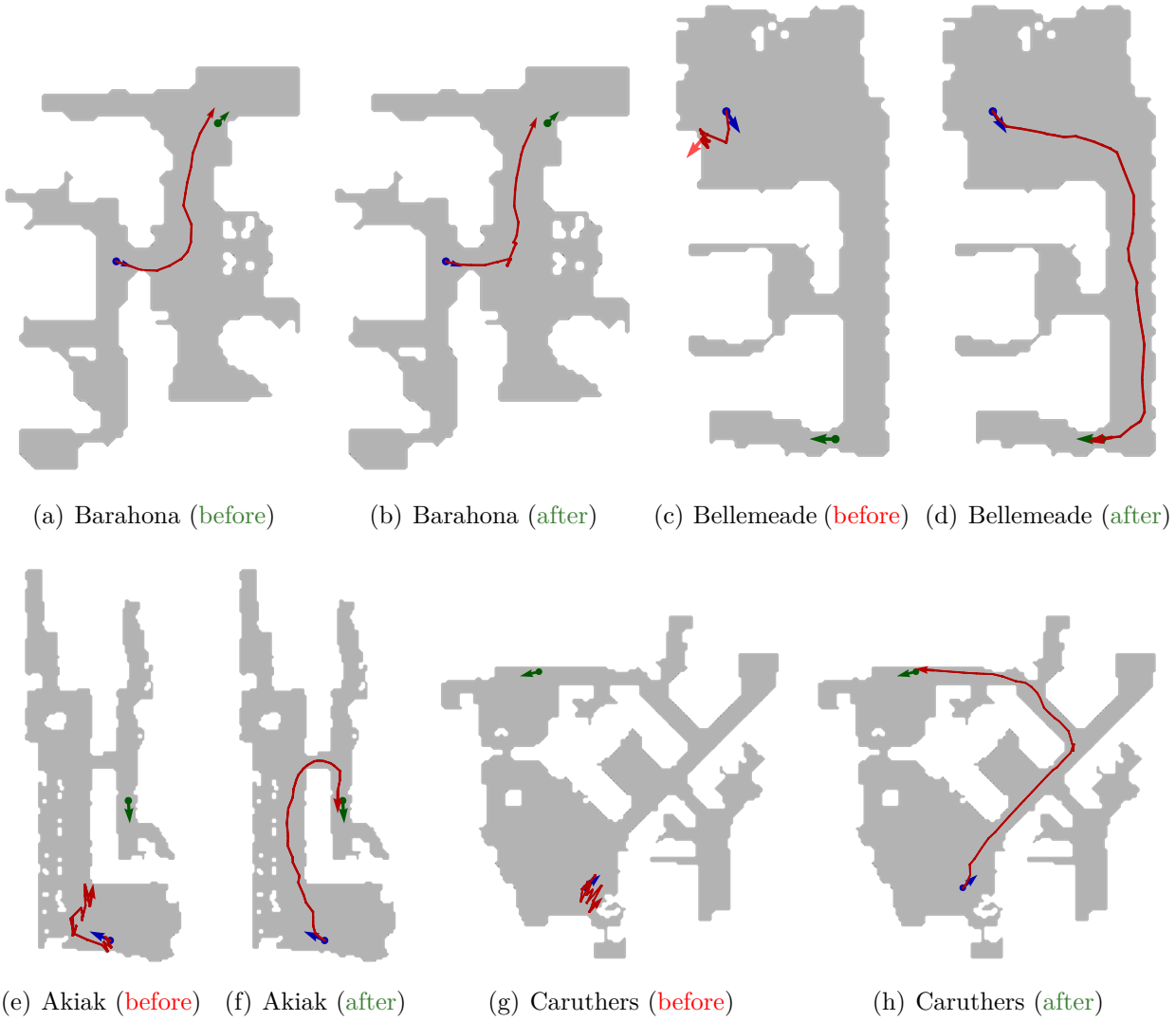
We evaluate the proposed graph update method to see how navigation performance is affected as the agent performs more queries. We start these experiments using graphs built with our finetuned models, and ask the agent to execute random queries while performing continuous graph updates. Table 4.2 summarizes the graph update parameters. After every 100 queries, we re-evaluate the navigation performance on the same static set of test episodes used in Chapter 4.2. We repeat this experiment three times so we can report the mean and the standard deviation.

Table 4.2. Graph update parameters.

Parameter	Value	Description
ϵ	0.5	Reachability score threshold for adding new nodes.
δ_e	0.35 m	Minimum Euclidean distance to determine node connectivity.
δ_θ	0.4 rad	Minimum relative yaw angle to determine node connectivity.
D_{merge}	0.0	Maximum geodesic distance to the existing node to be considered mergeable when adding new nodes.
D_{min}	0.65	Maximum geodesic distance when checking the success of edge traversal.
$p(z=1\|x=1)$	0.55	Measurement model for Bayes update.
R_p	0.5	Reachability score threshold for edge pruning.
σ	0.015	Variance of the measurement.

As seen in Figure 4.5, the success rate generally improves as we perform more queries, with notable gains initially, while the number of nodes and edges in the graphs do not substantially grow. We also see an initial *decrease* in the number of edges, suggesting that

our graph updates successfully pruned spurious edges causing initial navigation failures, then later added useful new nodes. We can see examples where the agent successfully completed initially-failed queries after performing graph updates in Figure 4.6. Qualitatively, we can see fewer spurious edges when comparing sample graphs before and after updates in Figure 4.7.

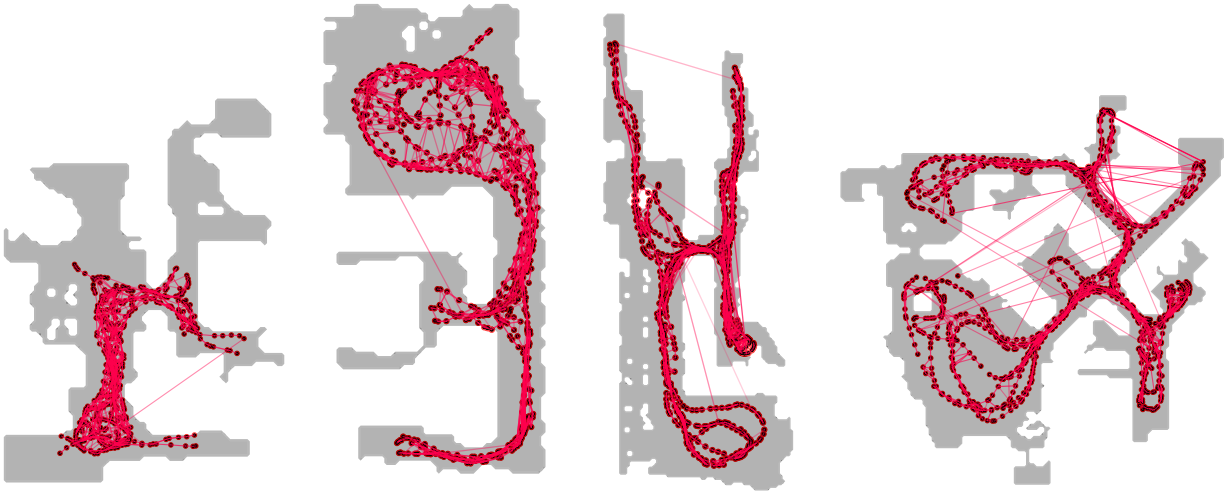


(a) Barahona (before) (b) Barahona (after) (c) Bellemeade (before) (d) Bellemeade (after)

(e) Akiak (before) (f) Akiak (after) (g) Caruthers (before) (h) Caruthers (after)

Figure 4.6. Comparison of navigation episodes in various simulated environments before and after 700 queries graph updates. Here, the blue and green arrows indicate the initial and goal pose of the agent, respectively. We can see a few instances where the agent initially failed to complete a query, and successfully completed the same query after the graph updates. To see the images seen by the agent during these episodes, see Figure A.2 in Appendix A.2.

We observe that sometimes the success rate decreased after a batch of graph updates. This is likely caused by spurious edges when adding new graph nodes near each 100th query, before we re-evaluate navigation performance. Nevertheless, such spurious edges are pruned in later updates, thus leading to increasing performance trends during lifelong navigation.



(a) Barahona-100 (b) Bellemeade-100 (c) Akiak-100 (d) Caruthers-100

(e) Barahona-400 (f) Bellemeade-400 (g) Akiak-400 (h) Caruthers-400

(i) Barahona-700 (j) Bellemeade-700 (k) Akiak-700 (l) Caruthers-700

Figure 4.7. Comparison of the updated graphs after 100, 400, and 700 queries.

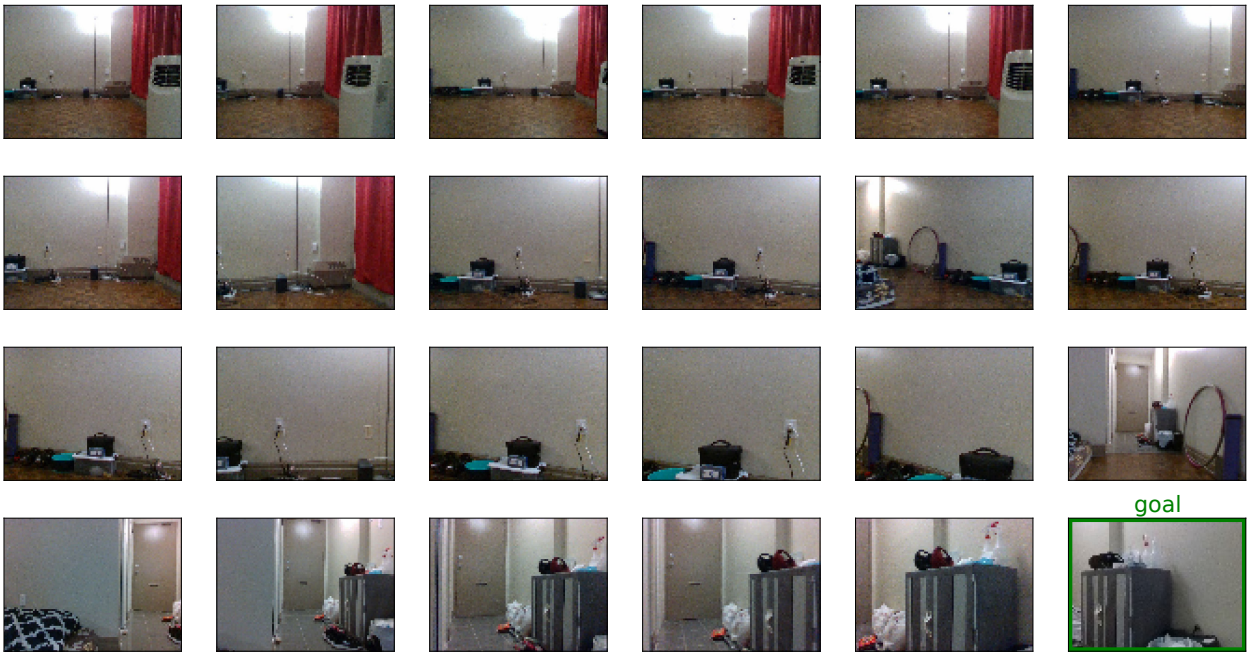
4.4. Real World Experiments

We evaluate the performance of our method in two real-world environments: a studio apartment and a medium-sized university laboratory. After teleoperating the robot in three to four loops around each space to collect trajectory data, we pick five goal images, and generate 20 test episodes. We use the iterative LQR [**65**] implementation from the PyRobot [**79**] library for our controller. In Table 4.3, we report navigation success rate before and after we perform graph updates with 30 queries.

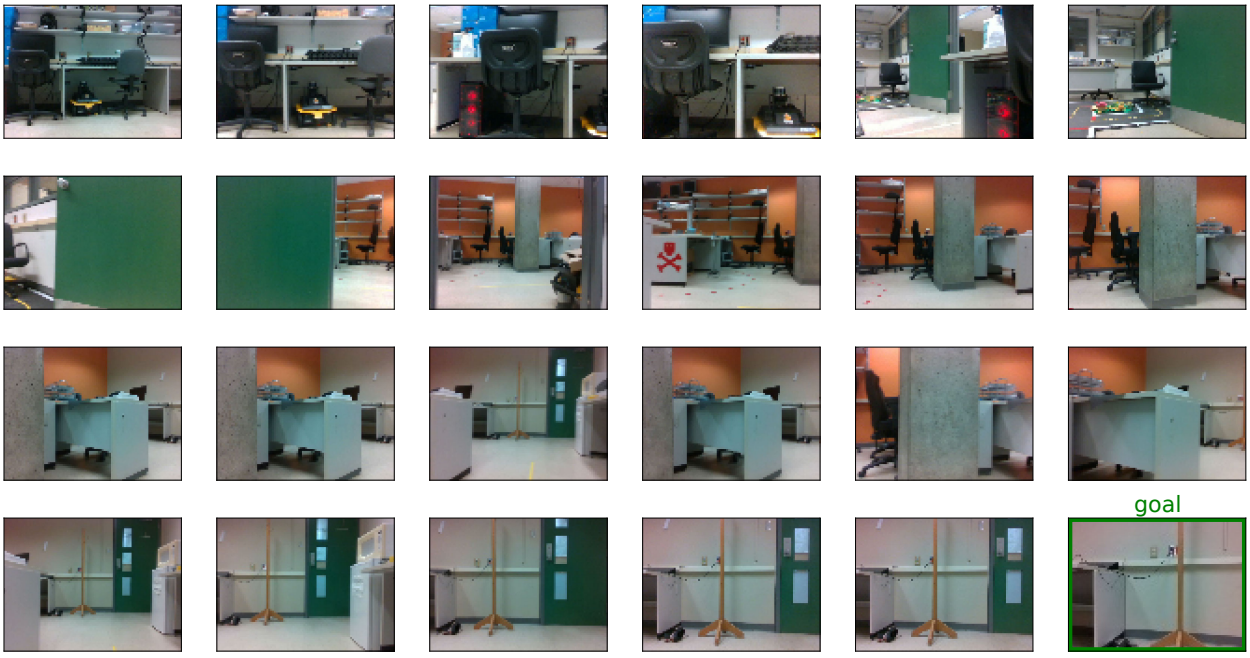
Table 4.3. Navigation success rate before and after 30 queries graph updates in real-world environments.

Environments	**Before**	**After**
Apartment	4/20	13/20
University Laboratory	4/20	14/20

These experiments confirm that our model performs well without needing large amounts of real-world data, especially when combined with our proposed graph update method. Our graph update method enhances the navigation performance with more than 3× increase in success rate in both environments. Figure 4.8 depicts successful navigation tasks across multiple twists and turns within both apartment and lab environments.



(a) Apartment



(b) University laboratory

Figure 4.8. Samples of sequence of images seen by the robot in both apartment and lab environments when navigating from top-left to the bottom-right image. To see more real-world deployment videos, see our project page: https://montrealrobotics.ca/ltvn/.

Chapter 5

Conclusion

We proposed a new image-based graph construction method that applies a sampling-based map building approach. The resulting topological visual navigation system not only produced sparser graphs compared to baseline methods, it also led to higher navigation performance. This system combines classical graph-based navigation with neural-based learning to enable the agent to update the graph continuously during navigation. Our experiments showed that these graph updates added useful new nodes and removed spurious edges, thus increasing performance throughout lifelong navigation. We also demonstrated a training regime using purely simulated data, combined with optional finetuning using small amounts of data from a target domain. Our real-world experiments showed that such deployment and finetuning resulted in significant gains in navigation performance.

While our graph update methods demonstrated promising results, its efficacy in only adding useful graph information can be further improved, especially when sampling new nodes to add when path planning fails. Furthermore, future work on learning-based topological visual navigation can also benefit from more theoretical analysis based on the capability of the learned model. In addition, our model finetuning method relies on expertly piloted trajectories with added odometry information. To improve its practicality, future work should consider methods to finetune the model using unordered set of images that can come from sources other than the robot (e.g., mobile phone cameras). Finally, in this work we consider the world to be static; extending to non-stationary environments remains a fruitful challenge. Future work should consider methods that can detect changes in the topology as the agent navigates the environment.

References

[1] LoCoBot - An open source low cost robot. http://www.locobot.org/. Accessed: 2021-09-14.

[2] Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian Reid, Stephen Gould et Anton Van Den Hengel : Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683, 2018.

[3] Kiam Heong Ang, G. Chong et Yun Li : PID control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005.

[4] Franz Aurenhammer : Voronoi diagrams — A survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, septembre 1991.

[5] Somil Bansal, Varun Tolani, Saurabh Gupta, Jitendra Malik et Claire Tomlin : Combining optimal control and learning for visual navigation in novel environments. *In* Leslie Pack Kaelbling, Danica Kragic et Komei Sugiura, éditeurs : *Proceedings of the Conference on Robot Learning*, volume 100 de *Proceedings of Machine Learning Research*, pages 420–429. PMLR, 30 Oct–01 Nov 2020.

[6] Timothy D. Barfoot : *State estimation for robotics*. Cambridge University Press, USA, 1st édition, 2017.

[7] Herbert Bay, Andreas Ess, Tinne Tuytelaars et Luc Van Gool : Speeded-up robust features (SURF). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. Similarity Matching in Computer Vision and Multimedia.

[8] P. Beeson, N.K. Jong et B. Kuipers : Towards autonomous topological place detection using the extended Voronoi graph. *In Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 4373–4379, 2005.

[9] Richard Ernest Bellman : *Dynamic programming*. Dover Publications, Inc., USA, 2003.

[10] Luca Bertinetto, Jack Valmadre, João F Henriques, Andrea Vedaldi et Philip H S Torr : Fully-convolutional siamese networks for object tracking. *In ECCV 2016 Workshops*, pages 850–865, 2016.

[11] P.J. Besl et Neil D. McKay : A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang *et al.* : End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[13] Francisco Bonin-Font, Alberto Ortiz et Gabriel Oliver : Visual navigation for mobile robots: A survey. *Journal of Intelligent and Robotic Systems*, 53(3):263, May 2008.

[14] J. Borenstein et Y. Koren : Real-time obstacle avoidance for fast mobile robots in cluttered environments. *In Proceedings., IEEE International Conference on Robotics and Automation*, pages 572–577 vol.1, 1990.

[15] Léon Bottou, Frank E Curtis et Jorge Nocedal : Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[16] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid et J.J. Leonard : Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[17] Michael Calonder, Vincent Lepetit, Christoph Strecha et Pascal Fua : BRIEF: Binary robust independent elementary features. *In* Kostas Daniilidis, Petros Maragos et Nikos Paragios, éditeurs : *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[18] Devendra Singh Chaplot, Dhiraj Gandhi, Abhinav Gupta et Ruslan Salakhutdinov : Object goal navigation using goal-oriented semantic exploration. *In In Neural Information Processing Systems*, 2020.

[19] R. Chatila et J. Laumond : Position referencing and consistent world modeling for mobile robots. *In Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 138–145, 1985.

[20] Abhishek Das, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh et Dhruv Batra : Embodied question answering. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 2054–2063, 2018.

[21] Matt Deitke, Winson Han, Alvaro Herrasti, Aniruddha Kembhavi, Eric Kolve, Roozbeh Mottaghi, Jordi Salvador, Dustin Schwenk, Eli VanderBilt, Matthew Wallingford *et al.* : RoboTHOR: An open simulation-to-real embodied AI platform. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3164–3174, 2020.

[22] F. Dellaert, D. Fox, W. Burgard et S. Thrun : Monte Carlo localization for mobile robots. *In Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1322–1328 vol.2, 1999.

[23] Guilherme N. DeSouza et Avinash C. Kak : Vision for mobile robot navigation: A survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(2):237–267, février 2002.

[24] E. W. Dijkstra : A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, décembre 1959.

[25] A. Diosi, G. Taylor et L. Kleeman : Interactive SLAM using laser and advanced sonar. *In Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 1103–1108, 2005.

[26] A. Elfes : A sonar-based mapping and navigation system. *In Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1151–1156, 1986.

[27] A. Elfes : Sonar-based real-world mapping and navigation. *IEEE Journal on Robotics and Automation*, 3(3):249–265, 1987.

[28] A. Elfes : Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[29] Scott Emmons, Ajay Jain, Misha Laskin, Thanard Kurutach, Pieter Abbeel et Deepak Pathak : Sparse graphical memory for robust planning. *In* H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan et H. Lin, éditeurs : *Advances in Neural Information Processing Systems*, volume 33, pages 5251–5262. Curran Associates, Inc., 2020.

[30] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers et W. Burgard : An evaluation of the RGB-D SLAM system. *In International Conference on Robotics and Automation (ICRA)*, St. Paul, MA, USA, May 2012.

[31] Ben Eysenbach, Russ R Salakhutdinov et Sergey Levine : Search on the replay buffer: Bridging planning and reinforcement learning. *In* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[32] P. Forssen, D. Meger, K. Lai, S. Helmer, J. J. Little et D. G. Lowe : Informed visual search: Combining attention and object recognition. *In 2008 IEEE International Conference on Robotics and Automation*, pages 935–942, May 2008.

[33] Stephen Friedman, Hanna Pasula et Dieter Fox : Voronoi random fields: Extracting the topological structure of indoor environments via place labeling. *In Proceedings of the 20th International Joint Conference on Artifical Intelligence*, IJCAI'07, page 2109–2114, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[34] Paul Furgale et Timothy D. Barfoot : Visual teach and repeat for long-range rover autonomy. *J. Field Robot.*, 27(5):534–560, septembre 2010.

[35] Carlos E. García, David M. Prett et Manfred Morari : Model predictive control: Theory and practice — A survey. *Automatica*, 25(3):335–348, 1989.

[36] Ross Girshick : Fast R-CNN. *In Proceedings of the International Conference on Computer Vision (ICCV)*, 2015.

[37] Ross Girshick, Jeff Donahue, Trevor Darrell et Jitendra Malik : Rich feature hierarchies for accurate object detection and semantic segmentation. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[38] Ian Goodfellow, Yoshua Bengio et Aaron Courville : *Deep learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[39] Daniel Gordon, Abhishek Kadian, Devi Parikh, Judy Hoffman et Dhruv Batra : SplitNet: Sim2sim and task2task transfer for embodied visual navigation. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 1022–1031, 2019.

[40] Daniel Gordon, Aniruddha Kembhavi, Mohammad Rastegari, Joseph Redmon, Dieter Fox et Ali Farhadi : IQA: Visual question answering in interactive environments. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4089–4098, 2018.

[41] Mohinder S. Grewal et Angus P. Andrews : Applications of Kalman filtering in aerospace 1960 to the present [historical perspectives]. *IEEE Control Systems Magazine*, 30(3):69–78, 2010.

[42] Chuan Guo, Geoff Pleiss, Yu Sun et Kilian Q. Weinberger : On calibration of modern neural networks. *In* Doina Precup et Yee Whye Teh, éditeurs : *Proceedings of the 34th International Conference on Machine Learning*, volume 70 de *Proceedings of Machine Learning Research*, pages 1321–1330. PMLR, 06–11 Aug 2017.

[43] Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar et Jitendra Malik : Cognitive mapping and planning for visual navigation. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[44] Peter E. Hart, Nils J. Nilsson et Bertram Raphael : A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[45] Kaiming He, Georgia Gkioxari, Piotr Dollár et Ross Girshick : Mask R-CNN. *In Proceedings of the International Conference on Computer Vision (ICCV)*, 2017.

[46] David Held, Sebastian Thrun et Silvio Savarese : Learning to track at 100 FPS with deep regression networks. *In European Conference Computer Vision (ECCV)*, 2016.

[47] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren et Dieter Fox : RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[48] Noriaki Hirose, Fei Xia, Roberto Martín-Martín, Amir Sadeghian et Silvio Savarese : Deep visual MPC-policy learning for navigation. *IEEE Robotics and Automation Letters*, 4(4):3184–3191, 2019.

[49] S. Hutchinson, G.D. Hager et P.I. Corke : A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, 1996.

[50] Simon J. Julier et Jeffrey K. Uhlmann : New extension of the Kalman filter to nonlinear systems. *In* Ivan Kadar, éditeur : *Signal Processing, Sensor Fusion, and Target Recognition VI*, volume 3068, pages 182 – 193. International Society for Optics and Photonics, SPIE, 1997.

[51] Gregory Kahn, Pieter Abbeel et Sergey Levine : BADGR: An autonomous self-supervised learning-based navigation system. *IEEE Robotics and Automation Letters*, 6(2):1312–1319, 2021.

[52] R. E. Kalman : A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960.

[53] R.E. Kalman : On the general theory of control systems. *IFAC Proceedings Volumes*, 1(1):491–502, 1960. 1st International IFAC Congress on Automatic and Remote Control, Moscow, USSR, 1960.

[54] L.E. Kavraki, P. Svestka, J.-C. Latombe et M.H. Overmars : Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[55] Eric Kolve, Roozbeh Mottaghi, Winson Han, Eli VanderBilt, Luca Weihs, Alvaro Herrasti, Daniel Gordon, Yuke Zhu, Abhinav Gupta et Ali Farhadi : AI2-THOR: An interactive 3D environment for visual AI. *arXiv preprint arXiv:1712.05474*, 2017.

[56] David Kortenkamp et Terry Weymouth : Topological mapping for mobile robots using a combination of sonar and vision sensing. *In Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 2)*, AAAI'94, page 979–984, USA, 1994. American Association for Artificial Intelligence.

[57] Alex Krizhevsky, Ilya Sutskever et Geoffrey E Hinton : ImageNet classification with deep convolutional neural networks. *In* F. Pereira, C. J. C. Burges, L. Bottou et K. Q. Weinberger, éditeurs : *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.

[58] Benjamin Kuipers : Modelling spatial knowledge. *In Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 1*, IJCAI'77, page 292–298, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[59] Benjamin Kuipers et Patrick Beeson : Bootstrap learning for place recognition. *In Eighteenth National Conference on Artificial Intelligence*, page 174–180, USA, 2002. American Association for Artificial Intelligence.

[60] Benjamin J. Kuipers et Todd S. Levitt : Navigation and mapping in large scale space. *AI Magazine*, 9(2):25, Jun. 1988.

[61] Steven M LaValle *et al.* : Rapidly-exploring random trees: A new tool for path planning. 1998.

[62] Yann LeCun, Yoshua Bengio et Geoffrey Hinton : Deep learning. *Nature*, 521(7553):436–444, May 2015.

[63] J.J. Leonard et H.F. Durrant-Whyte : Simultaneous map building and localization for an autonomous mobile robot. *In Proceedings IROS '91:IEEE/RSJ International Workshop on Intelligent Robots and Systems '91*, pages 1442–1447 vol.3, 1991.

[64] Ruihao Li, Sen Wang, Zhiqiang Long et Dongbing Gu : UnDeepVO: Monocular visual odometry through unsupervised deep learning. *In 2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7286–7291, 2018.

[65] Weiwei Li et Emanuel Todorov : Iterative linear quadratic regulator design for nonlinear biological movement systems. *In* Helder Araújo, Alves Vieira, José Braz, Bruno Encarnação et Marina Carvalho, éditeurs : *ICINCO 2004, Proceedings of the First International Conference on Informatics in Control, Automation and Robotics, Setúbal, Portugal, August 25-28, 2004*, pages 222–229. INSTICC Press, 2004.

[66] Kara Liu, Thanard Kurutach, Christine Tung, Pieter Abbeel et Aviv Tamar : Hallucinative topological memory for zero-shot visual planning. *In* Hal Daumé III et Aarti Singh, éditeurs : *Proceedings of the 37th International Conference on Machine Learning*, volume 119 de *Proceedings of Machine Learning Research*, pages 6259–6270. PMLR, 13–18 Jul 2020.

[67] Ilya Loshchilov et Frank Hutter : Decoupled weight decay regularization. *In International Conference on Learning Representations*, 2019.

[68] David G. Lowe : Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov 2004.

[69] D.G. Lowe : Object recognition from local scale-invariant features. *In Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1150–1157 vol.2, 1999.

[70] Vladimir J. Lumelsky et Alexander A. Stepanov : Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2(1):403–430, Nov 1987.

[71] Y. Matsumoto, M. Inaba et H. Inoue : Visual navigation using view-sequenced route representation. *In Proceedings of IEEE International Conference on Robotics and Automation*, volume 1, pages 83–88 vol.1, 1996.

[72] Y. Matsumoto, K. Sakai, M. Inaba et H. Inoue : View-based approach to robot navigation. *In Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, volume 3, pages 1702–1708 vol.3, 2000.

[73] David Meger, Marius Muja, Scott Helmer, Ankur Gupta, Catherine Gamroth, Tomas Hoffman, Matthew Baumann, Tristram Southey, Pooyan Fazli, Walter Wohlkinger, Pooja Viswanathan, James J. Little, David G. Lowe et James Orwell : Curious George: An integrated visual search platform. *In Proceedings of the 2010 Canadian Conference on Computer and Robot Vision*, CRV '10, page 107–114, USA, 2010. IEEE Computer Society.

[74] Xiangyun Meng, Nathan Ratliff, Yu Xiang et Dieter Fox : Scaling local control to large-scale topological navigation. *In 2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 672–678. IEEE, 2020.

[75] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu *et al.* : Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016.

[76] Hans Peter Moravec : *Obstacle avoidance and navigation in the real world by a seeing robot rover*. Thèse de doctorat, Stanford, CA, USA, 1980. AAI8024717.

[77] A. Mousavian, A. Toshev, M. Fišer, J. Košecká, A. Wahid et J. Davidson : Visual representations for semantic target driven navigation. *In 2019 International Conference on Robotics and Automation (ICRA)*, pages 8846–8852, 2019.

[78] Raul Mur-Artal, Jose Maria Martinez Montiel et Juan D Tardos : ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[79] Adithyavairavan Murali, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta et Abhinav Gupta : PyRobot: An open-source robotics framework for research and benchmarking. *arXiv preprint arXiv:1906.08236*, 2019.

[80] Robin R. Murphy : *Introduction to AI robotics*. MIT Press, Cambridge, MA, USA, 1st édition, 2000.

[81] Soroush Nasiriany, Vitchyr Pong, Steven Lin et Sergey Levine : Planning with goal-conditioned policies. *In* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[82] Aaron van den Oord, Yazhe Li et Oriol Vinyals : Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[83] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai et Soumith Chintala : PyTorch: An imperative style, high-performance deep learning library. *In* H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[84] Dean A. Pomerleau : ALVINN: An autonomous land vehicle in a neural network. *In Proceedings of the 1st International Conference on Neural Information Processing Systems*, NIPS'88, page 305–313, Cambridge, MA, USA, 1988. MIT Press.

[85] Joseph Redmon, Santosh Divvala, Ross Girshick et Ali Farhadi : You only look once: Unified, real-time object detection. *In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[86] Shaoqing Ren, Kaiming He, Ross Girshick et Jian Sun : Faster R-CNN: Towards real-time object detection with region proposal networks. *In* C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015.

[87] David M Rosen, Kevin J Doherty, Antonio Terán Espinoza et John J Leonard : Advances in inference and representation for simultaneous localization and mapping. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:215–242, 2021.

[88] E. Rosten et T. Drummond : Fusing points and lines for high performance tracking. *In Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1508–1515 Vol. 2, 2005.

[89] E. Royer, J. Bom, M. Dhome, B. Thuilot, M. Lhuillier et F. Marmoiton : Outdoor autonomous navigation using monocular vision. *In 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1253–1258, 2005.

[90] Ethan Rublee, Vincent Rabaud, Kurt Konolige et Gary Bradski : ORB: An efficient alternative to SIFT or SURF. *In 2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[91] David E. Rumelhart, Geoffrey E. Hinton et Ronald J. Williams : Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, Oct 1986.

[92] Nikolay Savinov, Alexey Dosovitskiy et Vladlen Koltun : Semi-parametric topological memory for navigation. *In International Conference on Learning Representations*, 2018.

[93] Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik *et al.* : Habitat: A platform for embodied AI research. *In Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9339–9347, 2019.

[94] Alexander Sax, Bradley Emi, Amir R Zamir, Leonidas Guibas, Silvio Savarese et Jitendra Malik : Mid-level visual representations improve generalization and sample efficiency for learning active tasks. *arXiv preprint arXiv:1812.11971*, 2018.

[95] S. Se, D. Lowe et J. Little : Vision-based mobile robot localization and mapping using scale-invariant features. *In Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, volume 2, pages 2051–2058 vol.2, 2001.

[96] Dhruv Shah, Benjamin Eysenbach, Gregory Kahn, Nicholas Rhinehart et Sergey Levine : ViNG: Learning open-world navigation with visual goals. *arXiv preprint arXiv:2012.09812*, 2020.

[97] Evan Shelhamer, Jonathan Long et Trevor Darrell : Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, avril 2017.

[98] Bokui Shen, Fei Xia, Chengshu Li, Roberto Martin-Martin, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D'Arpino, Sanjana Srivastava, Lyne P Tchapmi, Kent Vainio, Li Fei-Fei et Silvio Savarese : iGibson 1.0: A simulation environment for interactive tasks in large realistic scenes. *arXiv preprint*, 2020.

[99] Karen Simonyan et Andrew Zisserman : Very deep convolutional networks for large-scale image recognition. *In International Conference on Learning Representations*, 2015.

[100] Randall Smith, Matthew Self et Peter Cheeseman : Estimating uncertain spatial relationships in robotics. *In Autonomous robot vehicles*, pages 167–193. Springer, 1990.

[101] Richard S. Sutton et Andrew G. Barto : *Reinforcement learning: An introduction*. A Bradford Book, Cambridge, MA, USA, 2018.

[102] Takafumi Taketomi, Hideaki Uchiyama et Sei Ikeda : Visual SLAM algorithms: A survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, Jun 2017.

[103] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte et D. Schulz : MINERVA: A second-generation museum tour-guide robot. *In Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 3, pages 1999–2005 vol.3, 1999.

[104] Sebastian Thrun : Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.

[105] Sebastian Thrun, Wolfram Burgard et Dieter Fox : *Probabilistic robotics (intelligent robotics and autonomous agents)*. The MIT Press, 2005.

[106] Joshua P Tobin : *Real-world robotic perception and control using synthetic data*. University of California, Berkeley, 2019.

[107] Nicola Tomatis, Illah Nourbakhsh et Roland Siegwart : Hybrid simultaneous localization and map building: A natural integration of topological and metric. *Robotics and Autonomous Systems*, 44(1):3–14, 2003. Best Papers of the Eurobot '01 Workshop.

[108] Jack Valmadre, Luca Bertinetto, Joao Henriques, Andrea Vedaldi et Philip H. S. Torr : End-to-end representation learning for correlation filter based tracking. *In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[109] Shrihari Vasudevan, Stefan Gächter, Viet Nguyen et Roland Siegwart : Cognitive maps for mobile robots — An object based approach. *Robotics and Autonomous Systems*, 55(5):359–371, 2007. From Sensors to Human Spatial Concepts.

[110] Sen Wang, Ronald Clark, Hongkai Wen et Niki Trigoni : DeepVO: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. *In 2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2043–2050. IEEE, 2017.

[111] Erik Wijmans, Samyak Datta, Oleksandr Maksymets, Abhishek Das, Georgia Gkioxari, Stefan Lee, Irfan Essa, Devi Parikh et Dhruv Batra : Embodied question answering in photorealistic environments with point cloud perception. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6659–6668, 2019.

[112] Erik Wijmans, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva et Dhruv Batra : DD-PPO: Learning near-perfect pointgoal navigators from 2.5 billion frames. *arXiv*, pages arXiv–1911, 2019.

[113] N. Winters, J. Gaspar, G. Lacey et J. Santos-Victor : Omni-directional vision for robot navigation. *In Proceedings IEEE Workshop on Omnidirectional Vision (Cat. No.PR00704)*, pages 21–28, 2000.

[114] Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi et Roozbeh Mottaghi : Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6750–6759, 2019.

[115] Yi Wu, Yuxin Wu, Aviv Tamar, Stuart Russell, Georgia Gkioxari et Yuandong Tian : Bayesian relational memory for semantic visual navigation. *In Proceedings of the IEEE International Conference on Computer Vision*, pages 2769–2779, 2019.

[116] Fei Xia, Amir R. Zamir, Zhi-Yang He, Alexander Sax, Jitendra Malik et Silvio Savarese : Gibson env: Real-world perception for embodied agents. *In Computer Vision and Pattern Recognition (CVPR), 2018 IEEE Conference on*. IEEE, 2018.

[117] Jianwei Yang, Zhile Ren, Mingze Xu, Xinlei Chen, David Crandall, Devi Parikh et Dhruv Batra : Embodied amodal recognition: Learning to move to perceive objects. *In 2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 2040–2050, 2019.

[118] Wei Yang, Xiaolong Wang, Ali Farhadi, Abhinav Gupta et Roozbeh Mottaghi : Visual semantic navigation using scene priors, 2019.

[119] Xin Ye, Zhe Lin, Joon-Young Lee, Jianming Zhang, Shibin Zheng et Yezhou Yang : GAPLE: Generalizable approaching policy learning for robotic object searching in indoor environment. *IEEE Robotics and Automation Letters*, 4(4):4003–4010, 2019.

[120] Xin Ye, Zhe Lin, Haoxiang Li, Shibin Zheng et Yezhou Yang : Active object perceiver: Recognition-guided policy learning for object searching on mobile robots. *In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6857–6863. IEEE, 2018.

[121] Licheng Yu, Xinlei Chen, Georgia Gkioxari, Mohit Bansal, Tamara L Berg et Dhruv Batra : Multi-target embodied question answering. *In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6309–6318, 2019.

[122] H. Zender, O. Martínez Mozos, P. Jensfelt, G.-J.M. Kruijff et W. Burgard : Conceptual spatial representations for indoor mobile robots. *Robotics and Autonomous Systems*, 56(6):493–502, 2008. From Sensors to Human Spatial Concepts.

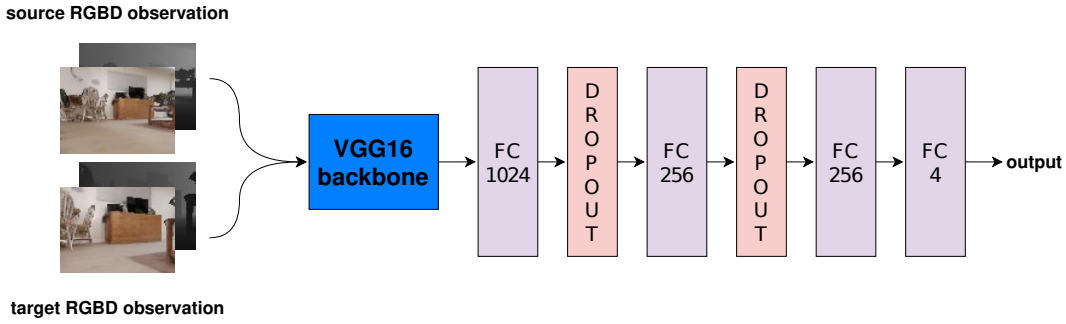
[123] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei et Ali Farhadi : Target-driven visual navigation in indoor scenes using deep reinforcement learning. *In 2017 IEEE international conference on robotics and automation (ICRA)*, pages 3357–3364. IEEE, 2017.

Appendix A

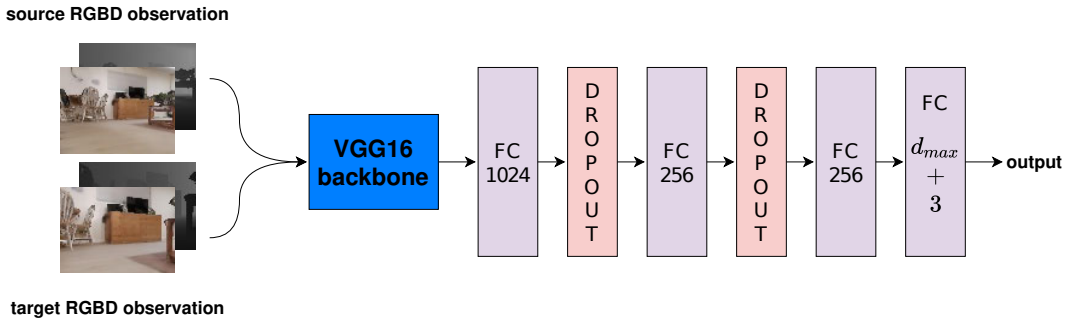
Additional Experimental Data

A.1. Network Architectures

Here, we visualize the neural networks architectures that we use to train our model and the baseline methods. All of the models used in our experiments are based on the VGG16 architecture [**99**]. Thus, the main difference in each model is the output layer. Figure A.1 illustrates the architecture for each model.



(a) Ours + SPTM

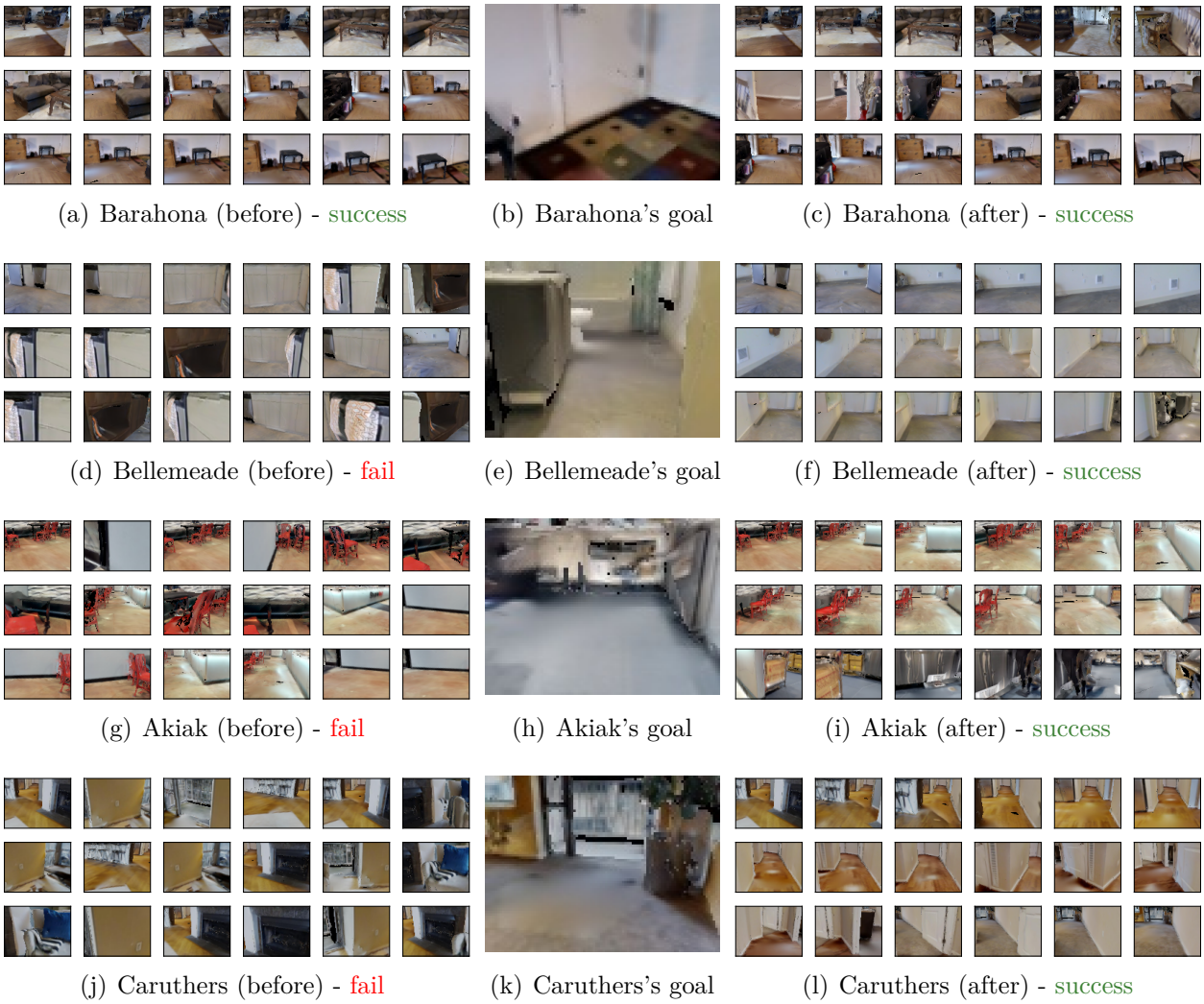


(b) ViNG

Figure A.1. Neural network architectures that we use in our experiments.

A.2. Comparison of Images Seen During Navigation Before and After Graph Updates in Simulation

Previously, we show the visualization of top-down trajectories of the agent navigating multiple episodes shown in Figure 4.6. To better understand the behavior of the agent when navigating these episodes, we also show the images as seen by the agent in Figure A.2. These are however best illustrated in the videos on our project page: https://montrealrobotics.ca/ltvn/.



(a) Barahona (before) - success (b) Barahona's goal (c) Barahona (after) - success

(d) Bellemeade (before) - fail (e) Bellemeade's goal (f) Bellemeade (after) - success

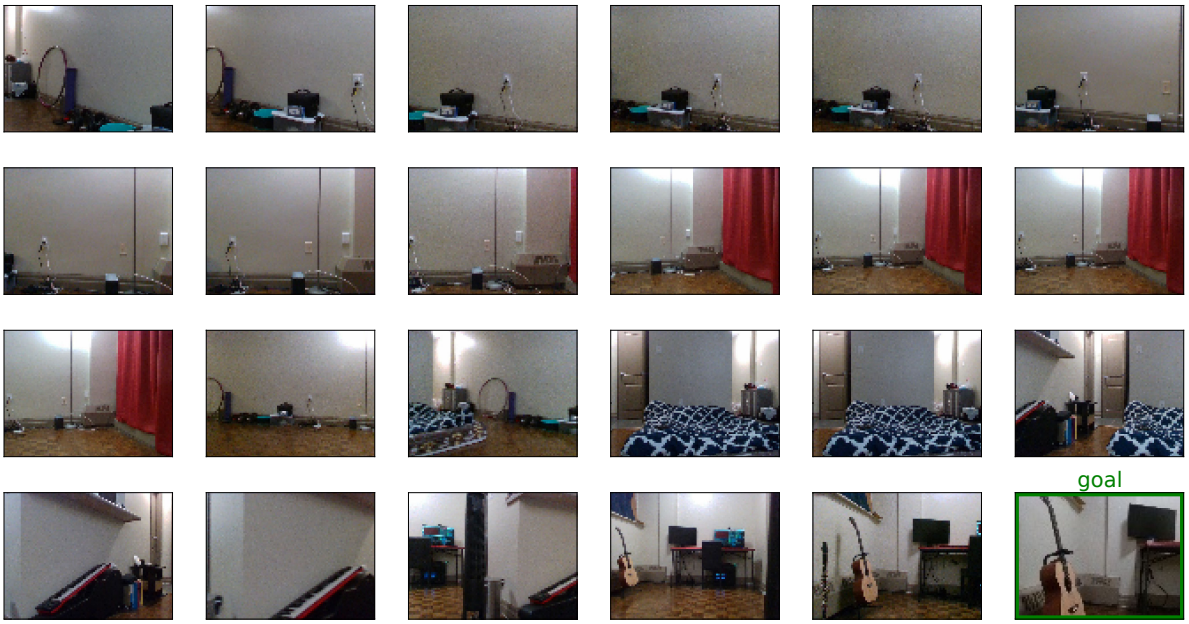
(g) Akiak (before) - fail (h) Akiak's goal (i) Akiak (after) - success

(j) Caruthers (before) - fail (k) Caruthers's goal (l) Caruthers (after) - success

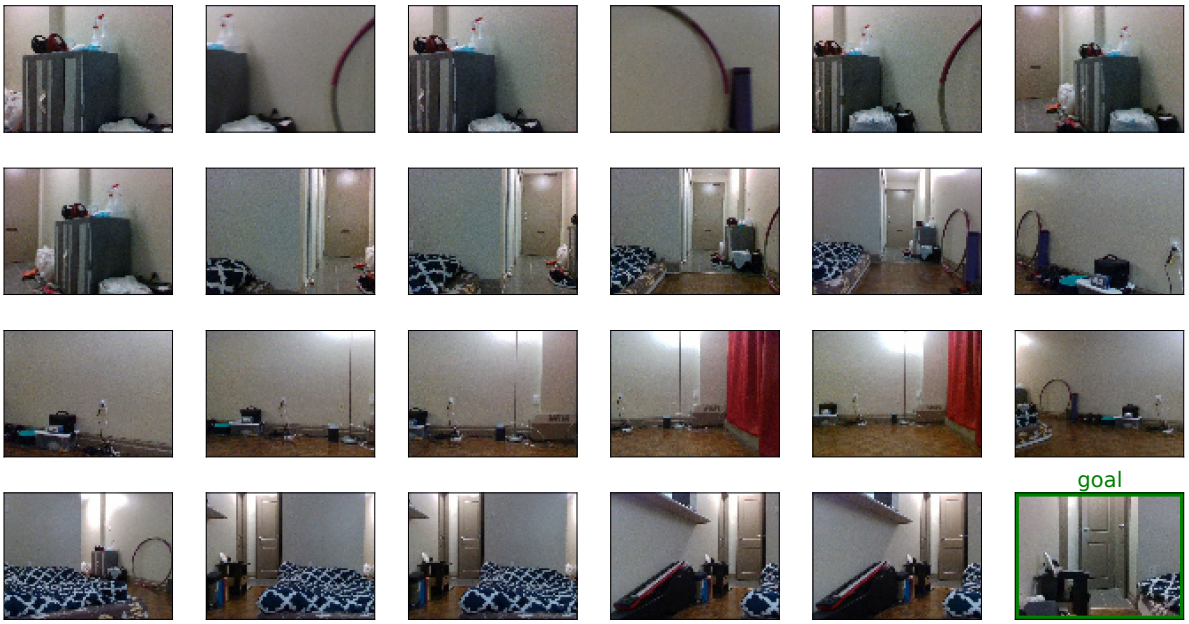
Figure A.2. Comparison of the image sequences seen during navigation before and after graph updates.

A.3. Navigation Images in Real-World Environments

We present additional real-world navigation examples after the graph updates in Figure A.3 and A.4. These are however best illustrated in the videos on our project page: https://montrealrobotics.ca/ltvn/.



(a) Episode 1



(b) Episode 2

Figure A.3. Examples of image sequences seen by the robot (from top-left to bottom-right) during successful navigation episodes in the apartment environment.

(a) Episode 1

(b) Episode 2

Figure A.4. Examples of image sequences seen by the robot (from top-left to bottom-right) during successful navigation episodes in the university laboratory environment.