Building Autonomous Agents with Hybrid Navigation Policies

Assem Sadek

N°d'ordre NNT : 2024ISAL0013

Thèse de Doctorat de l'INSA LYON,

membre de l'Université de Lyon

Ecole Doctorale N° 512
Informatique et Mathématique de Lyon
(INFOMATHS)

Spécialité de doctorat :
Informatique

Soutenue publiquement le 31 Janvier 2024, par :
Assem Sadek

Building Autonomous Agents with Hybrid Navigation Policies

Devant le jury composé de :

BABEL, Marie	Professeur des Universités	INSA RENNES	Présidente
MOUTARDE, Fabien	Professeur des Universités	MINES PARIS	Rapporteur
FILLIAT, David	Professeur des Universités	ENSTA PARIS	Rapporteur
WOLF, Christian	Principal Scientist - HDR	Naver Labs Europe	Examinateur
BASKURT, Atilla	Professeur des Universités	INSA LYON	Directeur de thèse
CHIDLOVSKII, Boris	Principal Scientist	NAVER Labs Europe	Invité

Département FEDORA – INSA Lyon - Ecoles Doctorales

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
CHIMIE	**CHIMIE DE LYON** **https://www.edchimie-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage secretariat@edchimie-lyon.fr	**M. Stéphane DANIELE** C2P2-CPE LYON-UMR 5265 Bâtiment F308, BP 2077 43 Boulevard du 11 novembre 1918 69616 Villeurbanne **directeur@edchimie-lyon.fr**
E.E.A.	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** **https://edeea.universite-lyon.fr** Sec. : Stéphanie CAUVIN Bâtiment Direction INSA Lyon Tél : 04.72.43.71.70 secretariat.edeea@insa-lyon.fr	**M. Philippe DELACHARTRE** INSA LYON Laboratoire CREATIS Bâtiment Blaise Pascal, 7 avenue Jean Capelle 69621 Villeurbanne CEDEX Tél : 04.72.43.88.63 **philippe.delachartre@insa-lyon.fr**
E2M2	**ÉVOLUTION, ÉCOSYSTÈME, MICROBIOLOGIE, MODÉLISATION** **http://e2m2.universite-lyon.fr** Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.e2m2@univ-lyon1.fr	**Mme Sandrine CHARLES** Université Claude Bernard Lyon 1 UFR Biosciences Bâtiment Mendel 43, boulevard du 11 Novembre 1918 69622 Villeurbanne CEDEX **sandrine.charles@univ-lyon1.fr**
EDISS	**INTERDISCIPLINAIRE SCIENCES-SANTÉ** **http://ediss.universite-lyon.fr** Sec. : Bénédicte LANZA Bât. Atrium, UCB Lyon 1 Tél : 04.72.44.83.62 secretariat.ediss@univ-lyon1.fr	**Mme Sylvie RICARD-BLUM** Institut de Chimie et Biochimie Moléculaires et Supramoléculaires (ICBMS) - UMR 5246 CNRS - Université Lyon 1 Bâtiment Raulin - 2ème étage Nord 43 Boulevard du 11 novembre 1918 69622 Villeurbanne Cedex Tél : +33(0)4 72 44 82 32 **sylvie.ricard-blum@univ-lyon1.fr**
INFOMATHS	**INFORMATIQUE ET MATHÉMATIQUES** **http://edinfomaths.universite-lyon.fr** Sec. : Renée EL MELHEM Bât. Blaise PASCAL, 3e étage Tél : 04.72.43.80.46 infomaths@univ-lyon1.fr	**M. Hamamache KHEDDOUCI** Université Claude Bernard Lyon 1 Bât. Nautibus 43, Boulevard du 11 novembre 1918 69 622 Villeurbanne Cedex France Tél : 04.72.44.83.69 **hamamache.kheddouci@univ-lyon1.fr**
Matériaux	**MATÉRIAUX DE LYON** **http://ed34.universite-lyon.fr** Sec. : Yann DE ORDENANA Tél : 04.72.18.62.44 yann.de-ordenana@ec-lyon.fr	**M. Stéphane BENAYOUN** Ecole Centrale de Lyon Laboratoire LTDS 36 avenue Guy de Collongue 69134 Ecully CEDEX Tél : 04.72.18.64.37 **stephane.benayoun@ec-lyon.fr**
MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** **http://edmega.universite-lyon.fr** Sec. : Stéphanie CAUVIN Tél : 04.72.43.71.70 Bâtiment Direction INSA Lyon mega@insa-lyon.fr	**M. Jocelyn BONJOUR** INSA Lyon Laboratoire CETHIL Bâtiment Sadi-Carnot 9, rue de la Physique 69621 Villeurbanne CEDEX **jocelyn.bonjour@insa-lyon.fr**
ScSo	**ScSo*** **https://edsciencessociales.universite-lyon.fr** Sec. : Mélina FAVETON INSA : J.Y. TOUSSAINT Tél : 04.78.69.77.79 melina.faveton@univ-lyon2.fr	**M. Bruno MILLY** Université Lumière Lyon 2 86 Rue Pasteur 69365 Lyon CEDEX 07 **bruno.milly@univ-lyon2.fr**

*ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

ABSTRACT

Recent advancements in AI, and specifically Machine Learning, are enabling robots to more seamlessly integrate into our everyday routines. The objective of this thesis is to take a further step towards the development of intelligent autonomous agents that can be embedded in our daily environment, such as houses, hospitals, shopping malls, and so forth. These agents ought to possess the capability to effectively navigate their surroundings to achieve a certain target, such as reaching a certain place in the environment or finding a certain object. Therefore, we examine a wide range of existing techniques for building an embodied navigation agent. These techniques can be fully learned by neural networks (learned-based techniques) or they can be based on geometry techniques that rely on explicit modeling of the agent and its environment. In this thesis, we build hybrid approaches that use both techniques in such a way that they can work not only in a simulation but also in a real physical environment. This is a common goal for all the contributions to this thesis.

In the first part of this manuscript, we study the generalization capacities of different variants of an agent trained in simulation and deployed directly in physical environments. Our results demonstrate that, for a point-to-point navigation task, an agent trained on a wide range of scenes and fine-tuned on a simulation twin of the targeted scene can reach a high performance and reduce the simulation-to-real gap. Furthermore, we scrutinize the reasoning and perception abilities of the agent by analyzing its use of sensors to execute a specific behavior. We demonstrate that the agent pays close attention to visual cues when necessary to steer clear of the surrounding obstacles.

In the second, we go a step further by tackling the more advanced task of finding a set of objects sequentially, which requires more reasoning and memorization capacities. We introduce a novel hybrid modular agent through the combination of the SLAM technique and learned components. We show that the performance of our agent has the best results in real environments compared to state-of-the-art agents on the same task. The results show the robustness of using hybrid agents.

Finally, we focus on enhancing the decision-making capacity of navigation agents by introducing another hybrid agent with a hierarchical approach. We propose a high-level meta-planner that dynamically switches between both decision-making approaches: the classical symbolic planner and the neural-based planner. We demonstrate that the hybrid solution can perform efficiently in real environments by combining both approaches.

RÉSUMÉ

Les progrès récents de l'IA, et plus particulièrement de l'apprentissage automatique, permettent aux robots de s'intégrer de manière plus transparente dans nos habitudes quotidiennes. L'objectif de cette thèse est de faire un pas de plus vers le développement d'agents autonomes intelligents qui peuvent être intégrés dans notre environnement quotidien, comme les maisons, les hôpitaux, les centres commerciaux, etc. Ces agents devraient posséder la capacité de naviguer efficacement dans leur environnement pour atteindre un certain objectif, comme atteindre une certaine zone de l'environnement ou trouver un certain objet. C'est pourquoi nous examinons le large éventail de techniques existantes pour la construction d'un agent de navigation incarné. Ces techniques peuvent entièrement être apprises par des réseaux neuronaux (techniques basées sur l'apprentissage) ou elles peuvent être des techniques fondées sur la géométrie qui reposent sur une modélisation explicite de l'agent et de son environnement. Dans cette thèse, nous construisons des approches hybrides qui utilisent les deux techniques afin de pouvoir fonctionner, non seulement dans une simulation, mais également dans un environnement physique réel. Il s'agit d'un objectif commun dans toutes les contributions de cette thèse.

Dans la première partie de ce manuscrit, nous étudions les potentialités de généralisation de différentes variantes d'agents entraînés en simulation et déployés directement dans des environnements physiques pour une tâche de navigation point à point. Nos résultats démontrent qu'un agent entraîné sur une large gamme de scènes et ajusté dans la simulation par la scène ciblée peut atteindre une performance élevée et réduire l'écart entre la simulation et la réalité. En outre, nous examinons les capacités de raisonnement et de perception de l'agent en analysant son utilisation des capteurs pour exécuter un comportement spécifique. Nous démontrons que l'agent prête effectivement une attention particulière aux indices visuels lorsque cela est nécessaire pour éviter les obstacles environnants.

Dans la seconde partie, nous allons plus loin en se concentrant sur la tâche plus avancée qui consiste à trouver un ensemble d'objets de manière séquentielle, ce qui nécessite davantage de capacités de raisonnement et de mémorisation. Nous introduisons un nouvel agent modulaire hybride en combinant la technique SLAM et des composants appris, tels qu'une nouvelle politique d'exploration entraînée par l'apprentissage par renforcement. Nous comparons les performances de notre agent à celles d'agents de pointe. En outre, nous montrons que notre agent obtient les meilleurs résultats dans des environnements réels, ce qui démontre sa

robustesse.

Enfin, nous nous concentrons sur l'amélioration de la capacité de prise de décision des agents de navigation en introduisant un autre agent hybride avec une approche hiérarchique. Alors, nous proposons un méta-planificateur de haut niveau qui passe dynamiquement d'une approche décisionnelle à l'autre : le planificateur symbolique classique et le planificateur à base de neurones. Nous démontrons que la solution hybride peut efficacement fonctionner dans des environnements réels en combinant les deux approches.

ACKNOWLEDGMENTS

The journey comes to an end! I'm pleased and honored to have had the opportunity to deliver my PhD thesis and to have access to a higher degree as a PhD from one of the notable institutions. In addition, I had the opportunity to work and collaborate with outstanding researchers and engineers between INSA Lyon and Naver Labs, where we can find robots navigating in a classic Alpine castle (where it has a view of the Belledonne... specifically, a view of the Chêne de Venon, interesting! isn't it?). Therefore, I have to thank everyone who contributed to making this opportunity the best.

First, I would love to thank the one who opened the door for this opportunity, Boris. Thank you for being my supervisor for this long journey since I stepped into the AI and Robotics world: starting with the first day of my Naver internship on 3D Vision, ending with the PhD defense. Your continuous feedback and support were crucial to the progress of this work. Similarly, I was lucky to be supervised by one of the most passionate scientists in the world, Christian. Thank you for always taking care of every single detail in this three-year work. Your optimism and enthusiasm have always played a big role in the advancement of our research. Also, I would like to thank Atilla for "taking the torch" to ensure that the thesis proceeds well and ends successfully.

As I said, I had the opportunity to work with great people. In particular, let me express my gratitude to Guillaume for all the collaboration we had during the thesis. I learned a lot from you in both aspects: research and engineering. Also, I have to thank Hervé for his patience with my questions and requests to conduct our robot experiments. To all Naver colleagues, and in particular the Spatial AI Team and the former 3D Vision Team, I'm grateful to be part of your team during this journey. Special thanks to my collaborators: Sombit, Gianluca, and Leonid for our research collaborations. Moreover, I was lucky and grateful to be surrounded by amazing PhD students from INSA Lyon: Steeven, Quentin, Theo, and in particular, thanks to Pierre and Edward, with whom I was pleased to collaborate. Last but not least, thank you, LoCoBot, for being a courageous robot and having all the power and eagerness to conduct +100 experiments.

I would like to thank the jury members and comité de suivi for accepting to share their expertise to evaluate my work. All the sincerest gratitude to the reviewers and examiners: David Filliat, Fabien Moutarde, and Marie Babel. Thank you for your notable reviews.

To all my friends in Grenoble, and in particular, "My Grenoble Family". I'm so grateful and pleased to have your support and caring footprints on this road. To all my remote friends from France, Belgium, Luxembourg, Canada, the USA, and Egypt, thank you all (and your internet operators) for always dropping a call to motivate and support me and embed the joy and fun in the journey! Furthermore, my deepest gratitude and love go to all my family members who were always here with their limitless support.

It is worth another thesis manuscript to describe the love, the caring, and the support I had from you, Ingie. You never failed to turn any hard moment into a joyful and lovely one. You always succeed in showing me that life is wonderful and that there's beauty (just like you) in everything. Bravo! Thank you!

Finally, if one fact can represent all the luck I have in my life, it will be the fact that I'm the son of Nagwa Diab. A woman who keeps showing me every day that being a mother is a prestigious title that is worth more than all the degrees, titles, and experiences that someone can earn. You are the one who has always taught me the fundamental values of education, knowledge, faith, and love. Thank you.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

DL	Deep Learning
RL	Reinforcement Learning
IRL	Inverse Reinforcement Learning
IL	Imitation Learning
BC	Behavior Cloning
SLAM	Simultaneous Localization and Mapping
AI	Artificial Intelligence
GRU	Gated Recurrent Unit
PPO	Proximal Policy Optimization
LIDAR	LIght Detection And Ranging
EKF	Extended Kalman Filters
SSL	Self-Supervised losses
MLP	Multi-layers perceptron
MDP	Markov Decision Processes
POMDP	Partially Observable Markov Decision Processes
CAP	Credit Assignment Problem
DPL	Direct Policy Learning
DAgger	Data Aggregation
CNN	Convolutional Neural Network
LQR	Linear Feedback Controller
CV	Computer Vision
NLP	Natural Language Processing
DL	Deep Learning
ML	Machine Learning
SL	Supervised Learning
UAV	Unmanned Aerial Vehicle
VQA	Visual Question Answering
EQA	Embodied Question Answering
NeRF	Neural Radiance Fields
RNN	Recurrent Neural Networks

CHAPTER 1

INTRODUCTION

Contents

1.1 Motivation and Context

Over the last few decades, automation has become a part of many human activities. Humans began to increasingly rely on machines to perform autonomously more complex tasks that required reasoning and a deep decision-making process. With the recent breakthrough in Artificial Intelligence (AI), (more precisely Deep Learning (DL)), the opportunity for machines to reason is boosted enormously. Machines can now recognize complex patterns in collected data of independent samples of the world using non-linear function approximation models, called Deep Neural Networks (Goodfellow et al. 2016). The large-scale training of such models with available big data, opens the door for machines to achieve state-of-the-art results on many digital tasks. Such as image classification (Krizhevsky et al. 2012; He et al. 2016; Chen et al. 2023), object detection & segmentation (Girshick et al. 2014; Li et al. 2022b; Jain et al. 2023), and machine translation (Sutskever et al. 2014; Liu et al. 2020). Moreover, with the recent advances in generative AI, machines have the potential to generate new data from the patterns detected by neural networks. They can generate new texts from context (OpenAI 2023), new art from description (Ramesh et al. 2022), new captions for images (Li et al. 2022a).

Toward Embodied AI The progress of Machine Learning (ML) in such complex tasks from the digital world of Computer Vision (CV) and Natural Language Processing (NLP), has been obtained in Digital AI, i.e. non-embodied AI (Figure 1.1). Machines perform their predictions without interacting with the surrounding environment. On the other side, Embodied AI, assumes the existence of an active artificial agent (e.g., robot, AR glasses) that are embodied inside an environment. This agent is not limited to perceiving the environment passively. It should inter-

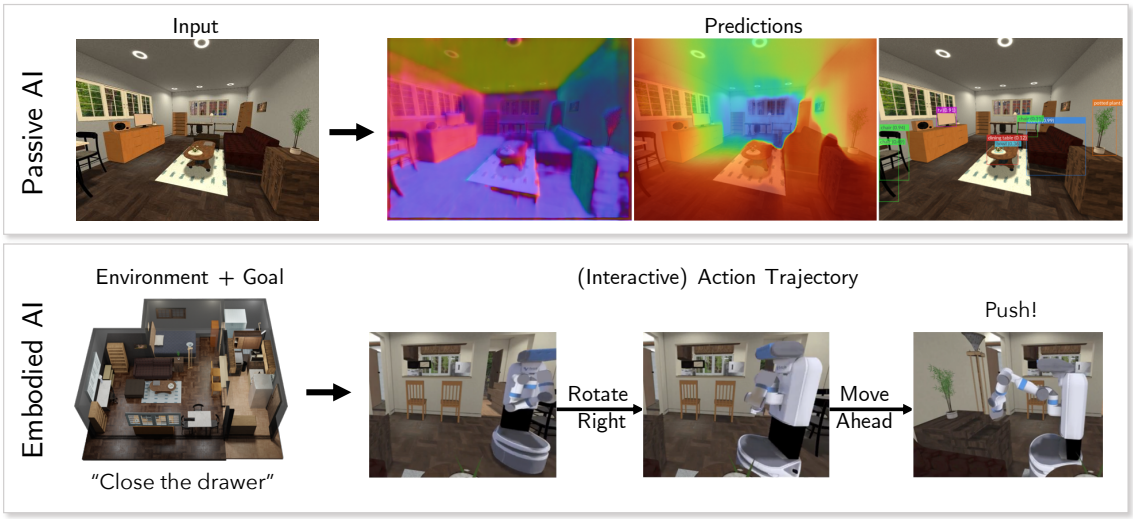


Figure 1.1 – **Passive Vs Embodied** In Passive AI tasks (Top), they doesn't interact with the environment, its predictions from the input is usually independent of its current state in the environment. Embodied AI tasks assume the existence of an artificial agent that perceives and interacts with the environment in a close-loop way to achieve a certain goal. The current state (or observations) of the agent inside is important to guide its prediction. Reproduced from (Deitke et al. 2022)

act intentionally with the environment in a close-loop way to achieve a certain goal inside the same environment for a given task. Examples of such embodied AI tasks are robot navigation, obstacle avoidance, object manipulation and arrangement. These tasks require from the agent to detect more than patterns only. The agent should also understand the geometry and dynamics of the environment to be able to interact wisely and strategically, and is required to plan.

Robot Navigation In this thesis, we focus our work on tackling the robot navigation tasks in indoor environments. Navigating inside an environment to perform a specific mission seems to be a daily, natural task that humans do. Humans navigate daily between rooms in their houses, office rooms in workplaces or stores in shopping malls. With time, they build a solid experience of the different environments they visit. Somehow, they succeed in building an "internal" map of environments. In addition, they recognize patterns and correlate between them, they exploit the layout regularities in seen environments to enable generalization to novel unseen environments. Taking a frequent example of restaurants, the task of finding your way to a specific room (or area) like the

restroom, humans solve such task efficiently in newly visited restaurants. A sense of patterns arises because of past similar experiences. We tend to favor the back of the restaurant and areas away from seating (Figure 1.2). While we start navigating, we are also open to correcting our decisions and for new exploration. We keep looking right and left, searching for signs of direction to guide and refine our choices of trajectory. From our experiences, our strategy will differ enormously in shopping centers where a restroom is probably located on any side of corridors between stores. Another common example is the arrangement of rooms in big houses. Humans capture the common layout such that on ground floor we find a kitchen, living room and dinning room. The bedrooms and bathroom are found upstairs.

Different navigation setups The complexity of reasoning required by a robot asked to navigate depends on the specific configuration and the task. A simpler version of the navigation task is waypoint navigation (e.g., Go to $5m$ north and $7m$ west). The agent can be equipped with or without means of localization and the live goal direction which impacts the difficulty of the task. Tasks can be more complex, like object searching (e.g., Go and find my pair of shoes) that require multi-reasoning such as object detection, object mapping and memorization. In some other task configurations, agent can be asked questions and the answer requires navigation and exploration inside the environment (e.g., *What is the food left on the dinning table?*). This task extends the Visual Question Answering (VQA) (Antol et al. 2015) task to the Embodied Question Answering (EQA) task (Das et al. 2018). The agent is required to process, map and correlate the input text to not a single visual observation but to the full 3D environment and by incorporating the past and future environment observations.

Depending on the task, a robot requires different capabilities, but a commonly required capability is "situation awareness": it needs to understand the spatial structure of its surroundings in order to find navigable space, avoid obstacles, identify key objects and places, and eventually anticipate the key actors (if required by the task) that contribute to achieve the main goal. Moreover, the agent needs at the end to correlate all elements of its acquired knowledge to take decisions to reach the main goal of the navigation task.

Analytical robotics as solid foundation Classically, the essential part of these capabilities have been addressed through combinations of reconstruction and planning. A large body of work is based on Simultaneous Localization and Mapping (SLAM) algorithms (Section 2.2.2), often with probabilistic models. It reconstructs the scene in the form of an actionable map (Section 2.2.1), followed by cost-based planning algorithms (Dijkstra 1959; Sethian 1996a). In another word, a classical agent solves the navigation problem by building a map and localize itself inside the map (Smith et al. 1987; Moutarlier et al. 1989). Then, it

Figure 1.2 – **Indoor Navigation:** Navigation to a certain goal (e.g. restroom) inside a restaurant, when no signs point directly to the restroom. In the left image scenario, humans tend to favor to go straight at the end of main seats. While in the right image scenario, going straight between tables is not likely to find the restroom. Humans learn some layout regularities that they can generalize in novel restaurants environments. Reproduced from (Chang et al. 2020)

performs planning (Dijkstra 1959; Ferguson et al. 2005) and controls (Chen et al. 2022b) to navigate efficiently toward the target. These methods have proved over the years their solid foundation and robustness in some real scenarios and are easy to interpret since they are based on first principles. We can trace back and know clearly why the agent took a specific action at a specific moment. However, they lack the understanding of semantics and regularities inside an environment. On the other hand, these agents can't exploit new regularities and generalize to unseen situations (e.g., getting stuck in a corridor with no new extra information to replan for an "escape" path).

End-to-end learning-based agents As an alternative, navigation can be cast as a sequential decision taking problem (Bellman 1957) and fully dealt with in a data-driven way with learning-based techniques in an end-to-end manner. By today, end-to-end neural networks are capable of delivering state-of-the-art performance with high-quality results in many CV (He et al. 2016; Chen et al. 2017) tasks. The main "motors" that contributed to these advancements are the availability of large-scale diverse datasets (Deng et al. 2009; Lin et al. 2015). Similarly, in robot navigation, neural agents can be trained in an end-to-end manner. Agents take as input the current observations (possibly, visual observations) of the environments and directly predict a motion action such as moving forward or rotating left and right (Savva et al. 2019; Chen et al. 2019). The main building blocks are Convolutional Neural Network (CNN) architectures (Lecun et al. 1998) for perception. In addition, they can rely on recurrent neural networks (Hochreiter et al. 1997; Cho et al. 2014) or transformer-based architectures (Vaswani et al. 2017; Dosovitskiy et al. 2021) to attend the history of observations. These types of

agents have been shown to generalize and exploit the semantic layout regularities in a scene (Chang et al. 2020).

With the current advancement in photorealistic simulators (Xia et al. 2018; Savva et al. 2019), supplied with physics engines (Coumans et al. 2016), these agents can be trained with Supervised Learning (SL) or Reinforcement Learning (RL) on hundreds of scenes and billions of frames. Zhu et al. (2016) shows an RL-based end-to-end learned navigation policy for image-goal navigation task. Zhao et al. (2021) uses an image-based model-free policy trained to map the input image directly to the continuous actuator control command for Unmanned Aerial Vehicle (UAV). Levine et al. (2015) used an end-to-end learning for grasping policy for a robotic arm. On the same path, Toromanoff et al. (2020) uses a fully end-to-end policy trained with RL for urban driving where the agent needs to handle scenario such as pedestrians and vehicles affordance and lane keeping.

Despite the promising results in some navigation tasks (Ramrakhya et al. 2022; Majumdar et al. 2022; Marza et al. 2022), a common challenge is the lack of interpretability of neural networks. End-to-end approaches treat neural models as a black-box. There is an active line of research that aims to understand and interpret the reasoning behavior of machine learning models (Lipton 2016) and some works already focus on the interpretability of embodied agents (Jaunet et al. 2020; Jaunet et al. 2021). We believe that end-to-end approaches are still far from being a trustful standalone system. Especially, in the case of real embodied agents, where the agent has to interact with humans and navigate the surrounding environments with safety guarantees. We need to understand how such a system works, and the reason behind its choices and steps. Failures or unsafe behavior in real environments can be damaging and costly. For these reasons, such situations and behaviors need to be understood, and we should have a clear interpretation of what leads the agent to these undesired scenarios.

Another challenge is the sample inefficiency of state-of-art training approaches such as Deep RL (Henderson et al. 2019). The agent needs billions of interactions to obtain state-of-the-art performance in the simplest navigation tasks such as *PointNav* (Section 2.1.2). Distributed RL (Espeholt et al. 2018; OpenAI 2018) can improve on wall-clock training time but with a huge cost in hardware. In navigation, Wijmans et al. (2019) propose a decentralized and distributed way to training agents on large-scale data in simulation. Although, this work shows that generalization is possible in the simulated world of high photo-realism, the transfer to the real world doesn't happen smoothly. We notice a drop in performance between simulation and real due to the gap between both setups (Sim2Real gap). Training in simulation and deploying in real is another challenge open for research (Sim2Real Transfer). Some techniques like domain adaptation (Li et al. 2020), domain randomization (Anderson et al. 2020) across different simulation domains are used to minimize the gap. Another common technique is to minimize the gap by applying domain adaptation between simulation and

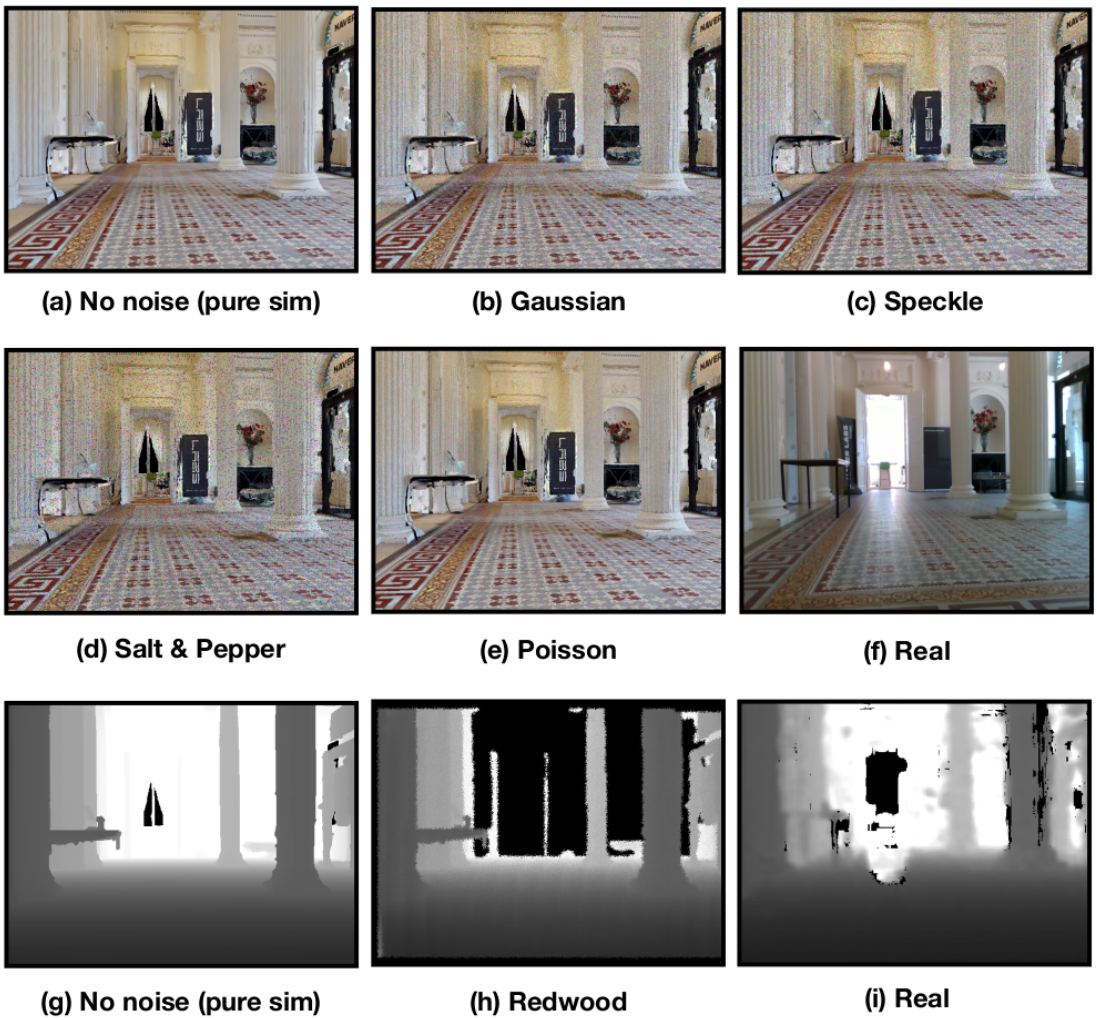


Figure 1.3 – **Statistical noises Vs Real** - Different noises applied on visual observations in simulator: RGB images (b to e) and on depth map (h). Noises are still far from the real twin in pair (f & i).

noisy simulation during training (Truong et al. 2021a). This is done by introducing some statistical noise models. As shown in figure 1.3, noises are still far from being realistic, specially in the case of a navigation agent that relies on visual input as main sensor. In the thesis, we tackled these two challenges of sample inefficiency and minimizing the gap between simulation and real to solve complex navigation tasks in real environments (Chapter 4).

Toward Hybrid Modular agents There is a of another family of embodied agents that adopt a modular approach. The idea is inspired by classical robotics on dividing the navigation task into sub-tasks such as exploration, object detection, localization, mapping, planning and control. Multiple modules are responsible for each or group of sub-tasks and during navigation, these modules communicate

with each other to achieve the main goal of navigation. Each of the learned modules can be trained separately (if trainable) (Chaplot et al. 2020a) or jointly (Chaplot et al. 2020d). Moreover, the design of such modular agents can be considered hybrid approach, where not all modules are based on end-to-end learning that maps inputs to outputs directly. Some modules can adopt a classical symbolic algorithm or partially introduce inductive biases in the design. We call such an agent, a hybrid agent.

Hybrid modular agents tend to have multiple advantages. First, the interpretability of the agent is higher due its modularity and a clear definition of the interfaces between modules. Second, the sample efficiency during training is possibly increased due to simplifying from one single module trained end-to-end to multiple submodules. Third, delegation of low-level decision-making parts such as planning and control to the existing solid well-founded symbolic algorithms in robotics, geometry and automation. Hence, acceptable performance on low-level actions could be guaranteed and leave the high-level reasoning parts to neural modules where they can understand the regularities and semantics of the environment. Therefore, many recent works focused on building hierarchical hybrid planners where a high-level planner understands the environment and proposes a way-point for low-level symbolic planners.

Objectives of the thesis The main objective of this thesis is to explore the usage of efficient hybrid navigation agents inside indoor real environments. During our study, we tried to answer the following questions:

- *Is navigating in a real environment with end-to-end neural agents possible? On which navigation tasks? Do we need any adaptation from simulated training environments?*

- *Can we interpret End-to-End agents? How do they use their sensors? Do some sensors have more important roles than others? In which situation?*

- *Can hybrid modular agents, simply designed, beat End-to-End neural agents trained with privileged information? Does the performance hold in both setups: simulated and real environments?*

- *Can we benefit simultaneously from two different planners: analytical planner and neural planner? Doesn't this put the agent in risk of a confusion that could imply longer and inefficient goal-navigation? Can a meta-planner learn an efficient dynamic switching strategy?*

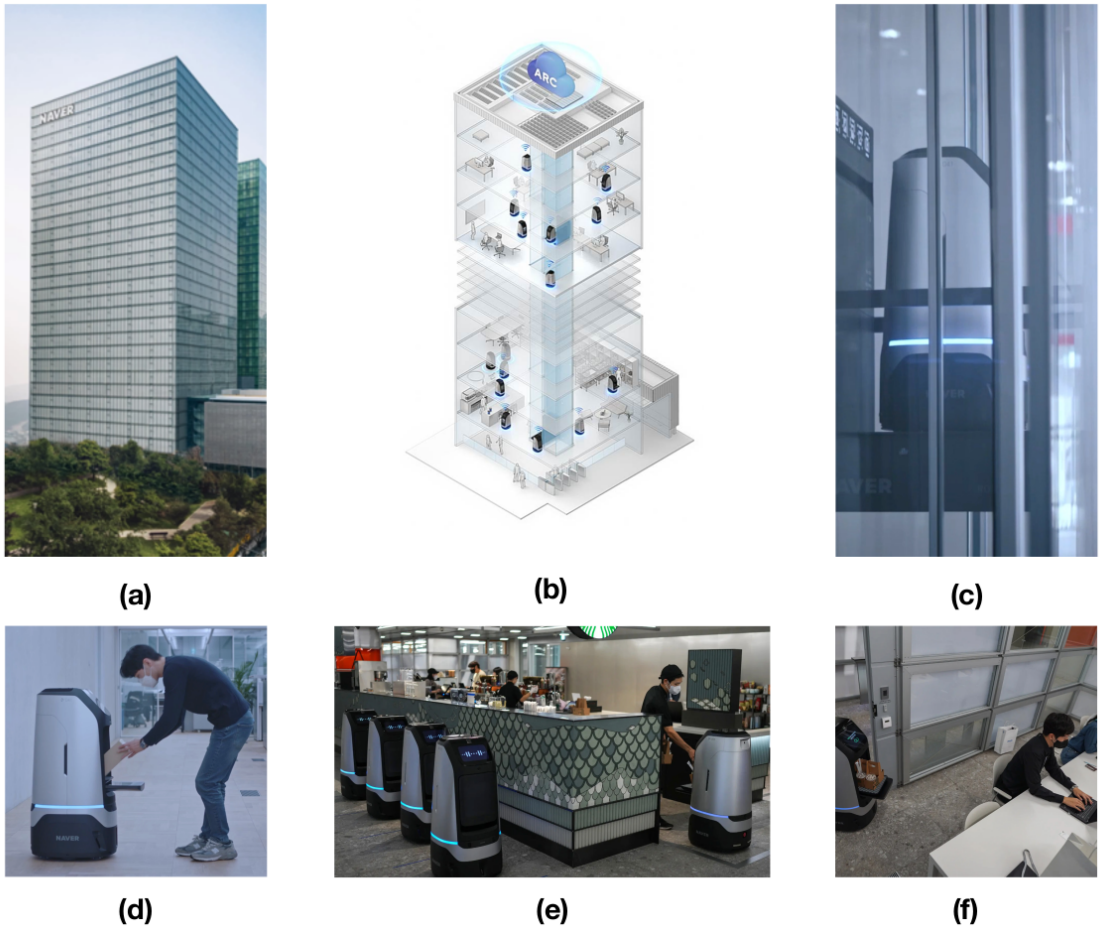


Figure 1.4 – **NAVER** 1784 - Naver new building (a and b) in Seoul, South Korea. Considered as the first fully robotic friendly workplace building (b) with robots equipped with specific lifters for robots (c). Robots navigate and interact with humans (d to f) to achieve daily tasks such as bringing coffee and parcels.

1.2 Industrial Context

This thesis is a part of a collaboration between the academic institution INSA Lyon and the industrial partner Naver Labs Europe (NLE), an R&D center of Naver Company. Naver is known to be specialized in many Digital AI products such as the Naver Search Engine and the Papago Translator. Currently, Naver focuses on contributing to the Embodied AI industry. It is highly interested in emerging robots in the daily life of humans, including different environments. In figure 1.4, we show Naver 1784 building in Seoul, the first ever robotic-friendly workplace building in the world. It's optimized for robots navigating and interacting with workers. In this context, NLE devotes its effort to conduct research toward enhancing robotic systems to perform embodied tasks such as robot navigation,

manipulation, and interaction with humans over voice. In particular, the thesis was started inside the 3*D* Vision team of NLE, working on various vision-related topics such as 3*D* scene understanding and robot localization. Then, the thesis joined the newly founded Spatial AI team, which focuses on robotics-related topics such as applying large-scale learning for robot navigation and manipulation.

1.3 Outline

The thesis is composed of six chapters including this introduction.

Literature review. In Chapter 2, we review previous work in classical robotics and DL most related to this thesis. In the first part (Section 2.1), we give an overview of the robot navigation problem and presenting different concepts and ongoing embodied AI tasks. Then in the second part (Section 2.2), we give a background of classical techniques in robotics that rely on symbolic and probabilistic approaches to solve the navigation tasks. Third, we present how learning-based techniques were used to build a neural agent that is able to reason and solve the task in an end-to-end manner (Section 2.3). Finally, we discuss how hybrid modular agents have been designed to solve navigation tasks (Section 2.4).

Experimental study on End-to-End agents in real environments. In Chapter 3, we present our first contribution, an in-depth experimental study of the performance of end-to-end neural agents. We study the generalization capacities of agents trained in simulation and deployed directly on physical environments. Alongside, we analyze the reasoning capacities of the agent for their sensor usages and the importance of different types of sensor signals.

Hybrid agent for Multi-Object Navigation. In Chapter 4, we introduce a new hybrid modular agent, through the combination of SLAM and learned components by RL and SL. A hybrid agent that leverages a novel stand-alone exploration policy. We compare the performance of our agent with the state-of-the-art agents on the task of *MultiON* (Wani et al. 2020). In addition, we show the performance of the same agents in real environments after we deployed directly on real physical robot.

Meta-agent for hybrid planning. In Chapter 5, we introduce another hybrid agent with a hierarchical approach. We propose a high level meta-planner that dynamically switches between both decision-making approaches: classical symbolic planner and end-to-end neural planner. We hypothesize that we can increase the navigation performance by leveraging both at the same time. In addition, we evaluate the agent in both simulation and real environments.

Discussion. Finally, in Chapter 6, we conclude this thesis by giving a bigger picture on the summary of the contributions and discuss the limitations of this work and what are the potential future directions for hybrid approaches.

1.4 Contributions

Publications This thesis is based on the material published in the following papers:

- Assem Sadek, Guillaume Bono, Boris Chidlovskii, and Christian Wolf (2022b). "An in-depth experimental study of sensor usage and visual reasoning of robots navigating in real environments". In: *ICRA 2022* - Chapter 3;
- Assem Sadek, Guillaume Bono, Boris Chidlovskii, Atilla Baskurt, and Christian Wolf (2022a). "Multi-Object Navigation in real environments using hybrid policies". In: *ICRA 2023* - Chapter 4;
- Sombit Dey, Assem Sadek, Gianluca Monaci, Boris Chidlovskii, and Christian Wolf (2022). "Learning whom to trust in navigation: dynamically switching between classical and neural planning". In: *IROS 2023* - Chapter 5;

CHAPTER 2

RELATED WORKS

Contents

In this thesis, we are interested in building hybrid models for visual navigation in real environments. These hybrid models will benefit from two known types of approaches in navigation: classical approaches based on geometry and optimization and learning-based approaches. Therefore, we give an overview in Section 2.1 on the navigation problem: the main concepts and the different variations of navigation tasks that currently exist and which are known in the embodied AI community. Second, we present the two aforementioned types of approaches that tackle the navigation task in Section 2.2 and Section 2.3. At the end, we describe the currently existing methods that combine the two families in Section 2.4.

2.1 Overview

The navigation task is a sequential decision-taking problem of an embodied agent in a simulated or physical environment (e.g. Video game, indoor building,

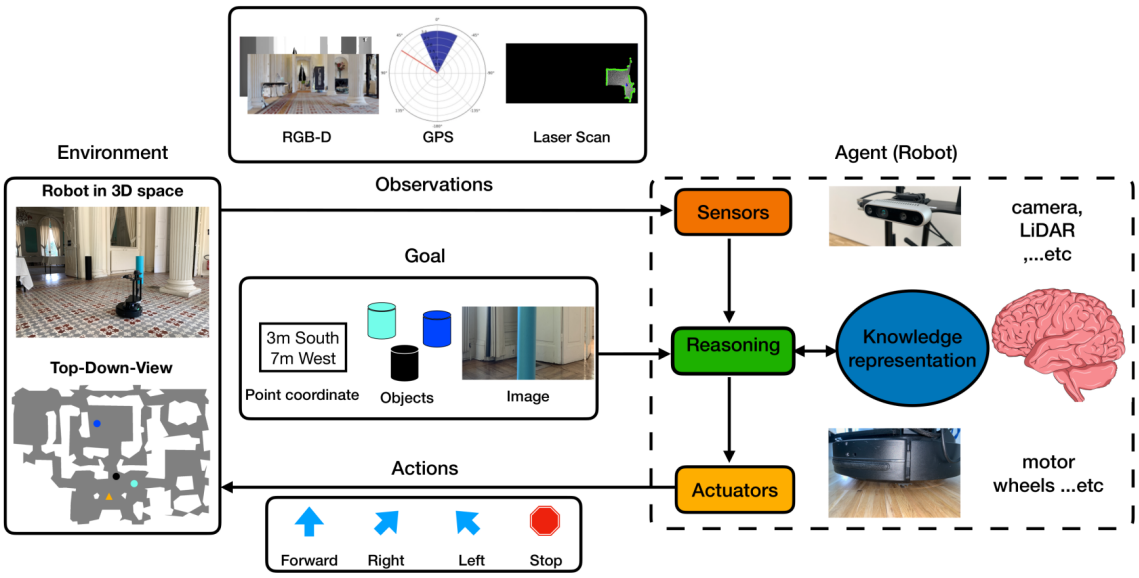
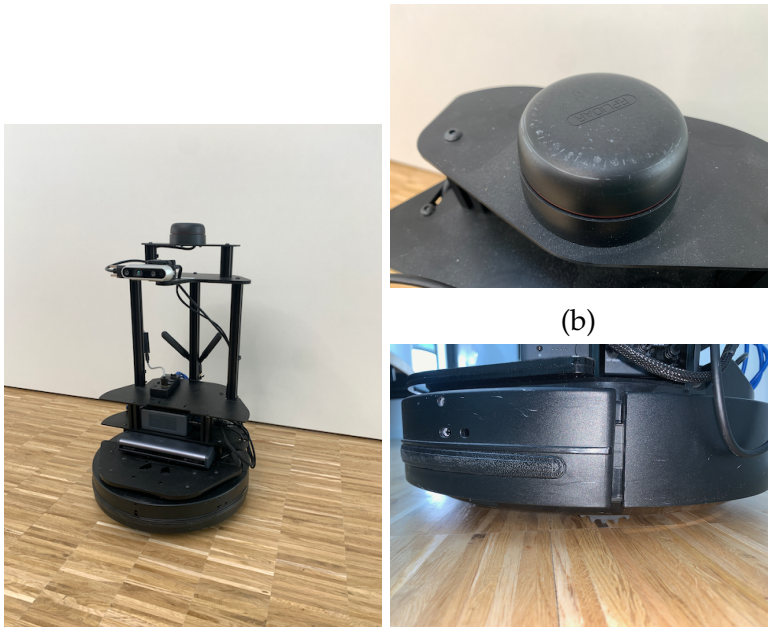


Figure 2.1 – **Agent Interaction**: In visual navigation, the agent needs to interact with the environment. The agent observes the environment using its sensors and processes the observations to turn it into knowledge as an internal representation of the environment. The agent takes a decision based on this knowledge in order to in order to achieve a certain given target (different modalities of goals are shown for illustration). Then, the agent applies these decisions toward the environment by performing specific actions using its actuators. For simplicity, we show the final actions conceptually, but in reality they are velocity commands transformed into motor velocities

outdoor area ... etc.). At each time step, an embodied agent (e.g., a robot - section 2.1.1), supplied with sensory inputs, must decide an action to take to reach a certain target (e.g. a location, an object) effectively and efficiently inside its environment (Section 2.1.3) to achieve a certain target. In this work, we focus on navigation tasks where the agent is embodied inside an indoor environment such as houses and offices. The sensory inputs, such as inputs from the camera and accelerometers, are used by the agent to decide (or predict) the optimal strategy and actions to optimize for its current target. The navigation depends on the sensor modalities and the specific type of navigation task. Therefore, in the following, we present a selected set of predefined navigation tasks and their setups: the definition of the problem, inputs and outputs of the embodied agent, the metrics used to measure the performance of a navigation agent. These navigation tasks are the current focus of the embodied AI community, and they are often presented as scientific competitions (Embodied Artificial Intelligence (AI) challenges Deitke et al. (2022)).



(a) (b) (c)

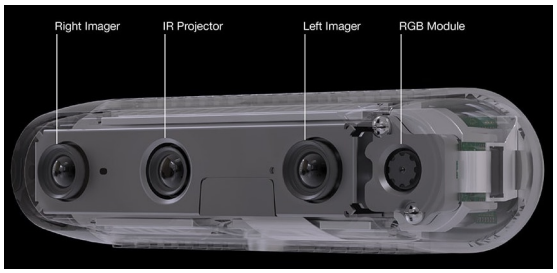
Figure 2.2 – **The LoCoBot**, a mobile robot (a) used for indoor navigation. It's equipped with LIght Detection And Ranging (LIDAR) (b), Kobuki mobile base (c) and RealSense camera for RGB and depth inputs (2.3b). The Kobuki base is equipped with wheels motors, an accelerometer, calibrated gyroscope and a bumper for collision detection.

2.1.1 The Embodied Agent

In this thesis, we focus on real-world indoor navigation. Moreover, the agent is a mobile robot (Figure 2.2a) in a physical indoor environment (e.g., house, hotel, hospital). Since the robot has to interact with the environment (Figure 2.1) to perform a specific task autonomously, it needs to be equipped with sensors and actuators. Sensors help the robot to get observations related to the current state of the physical environment (e.g., camera, LIDAR and accelerometer). Furthermore, actuators help the robot perform a certain action in the environment (e.g., wheel motors, servo motors and robotic arms). Depending on available sensors and actuators, we can define the observation space and the action space of the agent, respectively. In this section, we define possible observation space and action space of an embodied agent, and we describe how the physical sensors and actuators of common mobile robotic navigation platform works in the real-world.

Observation spaces The observations related to the state of the environment are presented to the robot as tensors. The observations are provided to the robot to process them and take actions based on them. When it comes to visual navigation,

(a)



(b)

Figure 2.3 – **RealSense Depth Camera**: (a) Intel RealSense Depth Camera mounted on LoCoBot (Figure 2.2a). It's composed of four main modules (from left to right): Right Imager, IR Projector, Left Imager and RGB Module.

robots typically have access, at time t, to the current 3D RGB tensor $I_t \in \mathbb{R}^{3\times H\times W}$ of the egocentric front view of the scene. Along with the current image, they should have a sense of depth, and this can be provided as a 2D depth map $D_t \in \mathbb{R}^{1\times H\times W}$ tensor, often aligned with the RGB tensor. They should be able to estimate the change in their position and orientation over time (Odometry) relative to a starting location. The robot movement state can be represented as a 1D tensor $P_t \in \mathbb{R}^6$ of the change in displacement (in x, y and z directions) and rotation angle (roll, pitch and yaw) in the 3D space over time.

Sensors With the recent advancements in technology, most of the current mobile robotic platforms can be equipped with a depth camera (Figure 2.3a) easily. Such cameras, like the Intel RealSense depth camera series, are composed of multiple modules: RGB camera, Left Imager, Right Imager and Infrared Projector (Figure 2.3b). In a stereo setup, the left and right Imagers get the current left and right front images, respectively. The depth map is then calculated using epipolar geometry, which estimates the 3D position of the corresponding pixels by performing triangulation (Hartley et al. 2004). Since all four modules are calibrated together on the RealSense, the RGB image and depth map can be aligned together. The IR projector is used to detect the depth of the smooth surface, such as walls, by projecting Infrared rays. Usually, accelerometer and calibrated gyroscope sensors are provided. The accelerometer detects the change in linear motion (displacement), while the gyroscope detects the change in angular motion (orientation). By keeping integrating the displacement and orientation information with the initial position of the robot, the robot can keep track of its current position and orientation with respect to a reference point. This process is

called **dead reckoning** and is subject to drift and accumulates error in estimating odometry.

Action spaces A typical robot with a differential drive system can perform various fundamental actions by rotating their base's wheels (moving forward/backward and turning left/right) or moving their camera (looking up/down/left/right). In the physical environment, defining the embodied actions space in navigation depends mainly on the available motors on the deployment robot. Motors can give the robot the access to greater degrees of freedom in the space: e.g. a moving forward/backward action with a specific speed on the up-front-axis of the robot. Therefore, the robot sets the rotation velocity of the motors to a specific value which is continuous. In order to specify the motor speed, we need to map it from desired linear and angular velocities. These velocities are calculated or estimated using the navigation algorithm and they mapped and maintained by a low-level control algorithm. When designing a navigation algorithm, an embodied agent might want to make predictions in a different action space. For example, in recent learning-based algorithms, training a navigation agent that regresses continuous action values (velocity values) tends to result in poor performance or leads to instability and random behavior (Seyde et al. 2021). Therefore, most of the existing approaches discretize the action spaces by defining a set of possible actions than can be conducted in the environment (e.g. move forward 25cm, turn left 25deg) and turn the problem to a classification problem (Tang et al. 2020).

Real-world challenges From the above description if how sensors and actuators work, we can deduce that robot interactions are prone to various incorrect behaviors. The erroneous behaviors are possibly due to errors in sensor readings (e.g. incorrect triangulation for depth - inaccurate accelerometer and gyroscope), obstructions on actuators (e.g. floor friction on wheels) or an unexpected latency in communication, specially in asynchronous communication between sending robot actions and receiving observations.

2.1.2 Navigation Tasks

2.1.2.1 Point-Goal Navigation

Definition This is a point-to-point navigation where the agent's goal is to navigate to target coordinates in the environment. The point coordinate can be presented as input in many coordinate frames. It can be given with respect to the agent's **starting** position, to a **global frame** in the environment or to the agent's **current** position. We focus on a specific Point-Goal task which is known under the name *PointNav* (Anderson et al. 2018a) on which most of the current benchmarks are evaluated. The agent starts in a specific position and orientation. It's given a

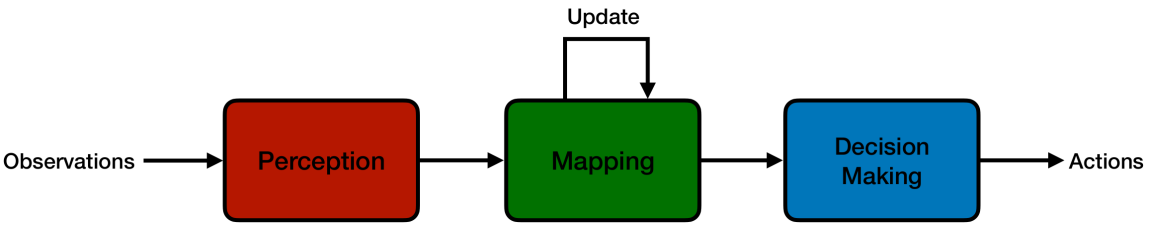


Figure 2.4 – **Agent Reasoning**: Conceptually, an embodied agent processes the given observations of the environment in three steps: (1) Perception, where the agent understands what is inside the observations and how they can be used. (2) Mapping, where the agent maps the perceived information into an internal representation. (3) Decision making, in this step, the agent uses the current knowledge (internal representation) to plan for the best strategic action to reach its goal.

point target which is a position relative to its starting position (e.g. Go to 3m South, 7m west) and then the target gets updated with respect to agent's current position. We assume in this task that the agent doesn't have access to the ground-truth map of the environment, otherwise, the task will be effortless to be optimally solved. Instead of a predefined map, the agent has access to current egocentric observations of the environment (RGB and depth observations). Initially, the target sensor in *PointNav* input was defined in the community (Deitke et al. 2022) as the current relative 2D polar vector of the goal (distance and Azimuth direction to the goal) with respect to the **current** position of the agent (usually, referred to as PointGoal GPS+Compass Sensor). But recently, the target input has been changed to be always a polar vector with respect to the **starting** position of the agent. This change raises the need of performing visual-based localization only and also removed the strong assumption made previously that the integrated odometry of the current position of the agent is perfect, which is not the case in the real environment.

Evaluation Setup In order to evaluate and benchmark the performance of different agents on *PointNav*, the community has provided multiple datasets which consist of *PointNav* episodes that can be run in simulators. An episode is defined by a starting position of the agent and a fixed PointGoal position: the agent will then have a fixed budget of 500 action steps to execute. The action space of the agent is discrete and consists of: *Move Forward* 0.25m, *Rotate Right* 30°, *Rotate Left* 30°, *STOP* (no move). The episode is considered terminated when the agent executes the action *STOP* or the time step budget is reached. There are three main metrics to evaluate the performance of an agent on a given dataset of predefined *PointNav* episode (Anderson et al. 2018a):

- *Success Rate* (SR) is the binary indicator of an episode to be successful. The agent must call `STOP` with the final distance d_k to the goal being lower than a predefined threshold d_{success} which is twice the radius of the agent:

$$\text{SR} = \mathbb{1}_{d \leq d_{\text{success}}} \tag{2.1}$$

- *Success weighted by Path Length* (SPL) is the SR with each success weighted by the ratio between the optimal path length l^* and the distance travelled by the agent l (Zhu et al. 2016):

$$\text{SPL} = \text{SR} \cdot \frac{l^*}{\max\{l, l^*\}} \tag{2.2}$$

- *Soft SPL* (sSPL) is a softer version of SPL where the boolean success value is replaced by a continuous measure of the progress made towards the goal:

$$\text{sSPL} = \max\left\{0, 1 - \frac{d}{l^*}\right\} \frac{l^*}{\max\{l, l^*\}}. \tag{2.3}$$

In the community, this task is now considered solved in simulation, but, in real environments the results are still far from being solved due to the sim2real gap. This gap comes from multiple sources: visual input from the camera, the actuation noise from the robot's base and the noise on the agent's current position estimate.

2.1.2.2 Object-Goal Navigation

Definition The task is also known under the name *ObjectNav* (Batra et al. 2020). In this task, the agent is asked to navigate to one of a fixed set object categories in the scene (e.g., find a bed in the house). The agent doesn't have any knowledge about its location, it only has access to the goal class label. Similar to *PointNav*, *ObjectNav* shares the same evaluation metrics. This task is considered being harder than *PointNav*, therefore, d_{success} is tolerated with a larger threshold on the distance to the goal compared to *PointNav*. In addition, it makes sense that the agent is supplied with two more discrete actions (*Look Up* 30°, *Look Down* 30°) which correspond to a change in the camera tilt to help the agent observe all the scenes. Unlike *PointNav*, in order to consider the episode successful, one extra condition needs to be satisfied: the target object must be visible to the agent in the last frame when the agent executed *STOP*. In other words, the object needs to be in the field of view of the camera at the end of the episode. This condition is significant because it ensures that the agent understands exactly what the object is and that it does not confuse it with another object that was by chance next to the target object.

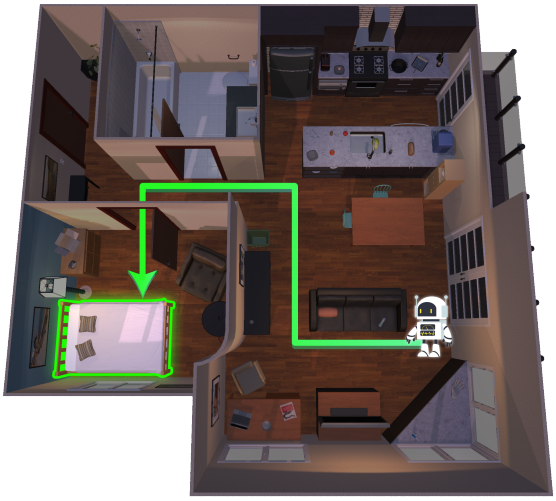
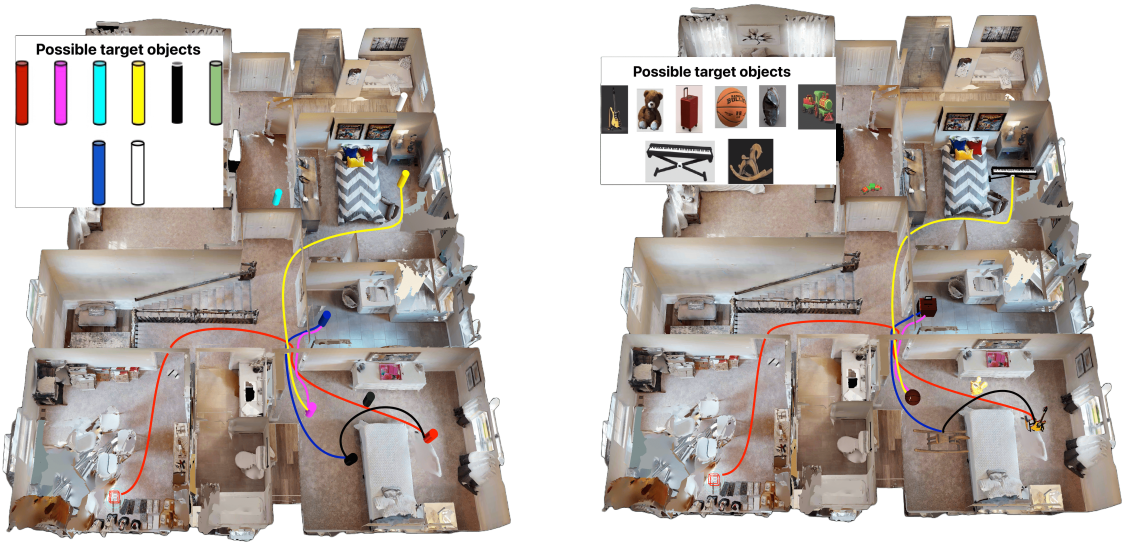


Figure 2.5 – **ObjectNav Task** The agent starts the episode in the living room and has 500 steps to find the bed. This figure is reproduced from (Deitke et al. 2022).

2.1.2.3 Multi-Object-Goal Navigation

Definition Multi-Object-goal Navigation, also known as *MultiON* (Wani et al. 2020), is an extended version of *ObjectNav*. Concretely, the agent is asked to find multiple objects in a predefined ordered sequence. The agent doesn't have access to the whole sequence of requested objects from the beginning of the episode. Therefore, each time the agent finds an object, a new object is requested and so on. Principally, the task's main goal is to evaluate the location-based memorization capacity of the agent. The task requires the agent to learn memorization and mapping of the location of previously potential seen objects in case it was asked to navigate to them in a future request. Like *PointNav*, the action space of the agent includes the same four discrete actions, except that action *STOP* is renamed to *FOUND*. *FOUND* is used when the agent reached a target.

Evolution of the task in the community Similar tasks have been introduced in the community. At the beginning, the target objects sequence was fixed and predetermined (Fang et al. 2019; Beeching et al. 2020a). Wani et al. (2020) introduced the task in a photo-realistic 3D simulator with a different target sequence per episode. In addition, the task doesn't rely on regularities between object semantics and scene semantics, as *ObjectNav* does (e.g. in *ObjectNav*, bed is associated with bedroom, and TV with living rooms). This addition is an important aspect to decouple objects from rooms, which makes the task a realistic and practical real-world scenario to find missing objects (e.g.keys, knife and



(a) (b)

Figure 2.6 – **MultiON Task**: (a) Episode with 5 target cylinder objects in a specific order (Red → Black → Blue → Pink → Yellow). Some unneeded cylidners exist as distractors (Cyan, White, and Green).
(b) Episode with 5 target real objects in a specific order (Guitar → Horse Toy → Travel Bag → Basket ball → Piano). Some unneeded objects exist as distractors (Train Toy, Teddy Bear and Backpack). This figure is reproduced from (Deitke et al. 2022).

toys). While in *ObjectNav* the agent exploits regularities in environment layouts, in *MultiON* the agent needs to learn to map regardless of any structure or regularities in the environment. In 2020, the task was introduced with abstract target objects (Figure 2.6a) and realistic objects were introduced later to make the task closer to real scenarios (Figure 2.6b). The task was evaluated in various N objects scenarios, called N-ON (e.g. 1-ON is equivalent to one object in an episode as *ObjectNav*). This task is still challenging in simulation setups, but we are interested in identifying its limitations in real-world setups. To our best knowledge, in this thesis, we are the first to evaluate this task in a real environment.

Evaluation Setup The agent is evaluated using similar metrics to *PointNav* and *ObjectNav*. The metrics are extended to consider partially successful episodes, when the agent navigates to a subset of the required targets.

- *Success Rate* (SR). The episode is successful (SR $= 1$) if the agent navigates to all goals in correct order, calling *FOUND* at each goal, within the budget of allowed steps. During the search for any goal, if *FOUND* is called at a wrong goal (not the current goal to find) or at a distance higher than a predefined

threshold d_{success} to the current goal, the episode terminates immediately with SR $= 0$.

- *Progress Rate* (PR) is the percentage of objects that are successfully *FOUND*. This metric is equal to *Success Rate* for one-goal scenario.
- *Success weighted by Path Length* (SPL) is the extended version of SPL in *PointNav* and *ObjectNav* tasks that takes into consideration multiple sequential goals:

$$l^* = \sum_{i=1}^{n} l^*_{i-1,i} \tag{2.4}$$

$$\text{SPL} = \text{SR} \cdot \frac{l^*}{\max\{l, l^*\}} \tag{2.5}$$

Here, l is the total distance traveled by the agent, l^* is the total geodesic shortest path distance from the agent's starting point through each goal position in order with $l^*_{i-1,i}$ indicating shortest geodesic distance from goal $i-1$ to goal i, and $i = 0$ being the starting point.

- *Progress weighted by Path Length* (PPL) is a partial version of SPL based on progress instead of success. PPL is equal to SPL for a one-goal scenario:

$$\bar{l} = \sum_{i=1}^{k} l^*_{i-1,i} \tag{2.6}$$

$$\text{SPL} = \text{PR} \cdot \frac{\bar{l}}{\max\{l, \bar{l}\}}; \tag{2.7}$$

with k being the number of objects found, and l and $l^*_{i-1,i}$ are defined as before.

2.1.2.4 Exploration

Definition & Motivation Navigate to explore can be a principal objective in its own right, for instance for a robot performing autonomous mapping, or it can be considered as an auxiliary task for target-driven navigation tasks (*PointNav* ... etc.). An agent can learn to explore while the target is not yet observed. The main goal of the agent is to explore the surrounding environment efficiently. In other words, the agent has to observe a maximum area of the environment in a given budget of steps. A second objective is that when an agent has already explored the environment, it can perform downstream tasks for goal-specific navigation in a more efficient way. A concrete example can in be found in classical robotics (Section 2.2), when an agent is asked first to do a fast scan of an environment and map it (Section 2.2.1). This agent can reach upcoming goals more efficiently than

a map-less agent. On the other side, for learning-based agents (Section 2.3), we conjecture that pretraining an agent on an exploration task and then fine-tuning on a downstream task can boost the performance of the agent (Chen et al. 2019).

Evaluation metrics A common metric for the exploration task is coverage, which measures the maximum area explored by the agent. Coverage can be expressed in m^2 or as a ratio between the explored area and the total ground-truth explorable area in the environment. The total explorable area in the environment map is known to be traversable or non-traversable one. We consider an area to be explored if it's in the current field of view of the agent and within a certain predefined frontward distance range.

After we presented the common tasks in embodied navigation, we focus in this thesis on three tasks: *PointNav*, *MultiON* and exploration tasks. We are interested in evaluating and contributing to these tasks in real-world scenarios instead of focusing on simulation evaluation only (Section 2.3.1).

2.1.3 Decision Making

After the agent processes the different observations from the environment and maps them into an internal presentation (Figure2.4), the agent uses this representation as knowledge in order to make decisions toward the environment to reach its goal. The process of decision making can be tackled by leveraging classical planning algorithms (Section 2.1.3.1) or by formulating the decision-making problem as a learning problem (Section 2.1.3.2).

2.1.3.1 Analytical planning

Motivation and definition Given an existing map of the environment (Section 2.2.1), a robot should be able to exploit this map effectively in order to identify an efficient path to navigate from a starting point to a destination point. We consider a path efficient when it is collision-free and has the minimum travelling-cost (e.g. minimum distance). The time complexity to find the shortest-path increases with the increase in the size of the environment (map) and with the degrees of freedom of the robot/agent (e.g. In a maze, an agent that can move vertically, horizontally and diagonally has a larger pool of paths to explore than an agent with vertical and horizontal moves only). Moreover, an effective path planning algorithm must consider two criteria: (1) the algorithm should always find the optimal path in realistic static environments but should also be extendable to dynamic environments. (2) It must aim to minimize the memory and time complexity. In this section, we show the fundamental algorithm of analytical planning and its variants. For the rest of the section, we will assume that an environment

topological map $\mathcal{G}$ is provided. It consists of N nodes (vertices) and V weighted edges. The goal for each planning algorithm is to find a path to follow on the map. The path starts from a source node n_s (which is the agent's current position, in our navigation context) to a goal node n_g, the goal node can be the final target navigation goal or intermediate sub-goals.

Dijkstra The fundamental algorithm (Dijkstra 1959) relies on a greedy approach for path planning. The common version finds the shortest path from a source node n_s to all other nodes by building and storing a shortest-path tree. This tree is then used to find the shortest path to a specific given goal node. Dijkstra maintains a cost function $g(n)$ that represents bounds on the cost of navigating from the source node n_s to every other node n. The core building block of the algorithm is to take the greedy decision on the sub-problems between a node n_i to all its nodes n_j where $n_j \in Neighbors(n_i)$. The goal is to find the weights $g(n_j)$ for all the neighbors by following the bound update equation:

$$g(n_j) = min(g(n_j), g(n_i) + d_{i,j}) \tag{2.8}$$

where d_{ij} is the travelling cost from n_i to n_j. The pseudo-code algorithm of dijsktra is provided below.

In order to guarantee optimality, the bound updates need to be performed in the right order, selecting node n_i with minimum current bound. This is usually done with a priority queue.

Dijkstra works on graphs of static environments and assumes that all node edges are positively weighted. An improved algorithm is introduced: The Floyd algorithm (Kang et al. 2008). Floyd is known to find the shortest path in a generic graph that can be weighted positively or negatively, but it provides only a point-to-point shortest path and not like the traditional Dijkstra that can provide a point-to-all shortest path tree. More generally, Dijkstra can be considered as a reliable algorithm although it heavily consumes memory for larger maps. Also, since it relies on computing all the path possibility in order to get the shortest path, its computation complexity is O(n^2) slightly lower with a priority queue. To overcome the memory limitations, Fadzli et al. (2015) introduced a new memory scheme, the multilayer dictionary. The latter was also used for dynamic environments (Silva et al. 2010)

Dijkstra extension - A* The A^* algorithm (Ferguson et al. 2005) can be considered as an enhanced version of Dijkstra since it functions similarly to Dijkstra. It potentially can save the amount of computation time. Similar to Dijkstra, $A*$ traverses the graph to find the minimum-cost path tree from the starting point, n_s except that it adds a heuristic-based function to guide its search. The main advantage of using heuristic is that it can quickly converge the algorithm toward

the expected results (Zhang et al. 2014). Therefore, the final cost function $f(n)$ is represented as follows:

$$f(n) = g(n) + h(n) \tag{2.9}$$

Where $g(n)$ is the same cost from node n to the starting node n_s, as in Dijkstra. and $h(n)$ is the heuristic cost of the optimal path from n to the goal node n_g. An appropriate heuristic functions $h(n)$ need to be decided. An appropriate heuristic function should be consistent. Moreover, $h(n) = 0$ if n is the targeted node ($n = n_g$), and for a given node n, and all its successors $n^{'}$, the heuristic cost from node n to target node n_g is not greater than the cost of moving from n to node $n^{'}$ plus the heuristic cost from $n^{'}$ to n_g, in other word can be represented in the following inequality:

$$h(n) \leq c(n, n^{'}) + h(n^{'}) \tag{2.10}$$

Furthermore, the real cost from a node n to goal node n_g, $h^*(n)$ should be an upper bound of the chosen $h(n)$. Some common heuristic functions are used, such as Manhattan distance, Euclidean distance and Octile distance (Yao et al. 2010; Chen et al. 2018). In some real applications, the number of steerings (turnings) should be minimized as much as possible and to avoid sudden turns and improve the smoothness of the final path. Hence, a penalty factor $p(n)$ is added to the final cost $f(n)$ (Yijing et al. 2018). Cheng et al. (2014) introduced a weighted-sum on the cost terms in $f(n)$.

2.1.3.2 Learning

Similar to the definition of embodied agent in section 2.1.1, a learning-based agent interacts with an environment by following a learned policy $\pi(a|s)$. Given a current state s_t, it takes an action a_t based on the policy prediction and as a result, the agent transits to a new state s_{t+1}. Generally, this sequential decision-making process can be formulated as a Markov Decision Process (MDP) (Bellman 1957) (Figure 2.7). Formally, MDP are defined as a 5-tuple $\langle \mathbf{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$ where:

- $\mathcal{S}$: state space, all possible states in the environment.
- $\mathcal{A}$: action space, all possible actions that the agent can perform.
- $\mathcal{T}$: conditional state transition model ($T(s'|s, a)$).
- $\mathcal{R}$: reward model (R(s, a)).
- $\gamma \in (0, 1]$: a discount factor.

From their name, MDPs satisfy the Markov property: the transition to the next state s' depends on the current state s and the action a only and is conditionally independent of all previous states and actions. In another word, a given state

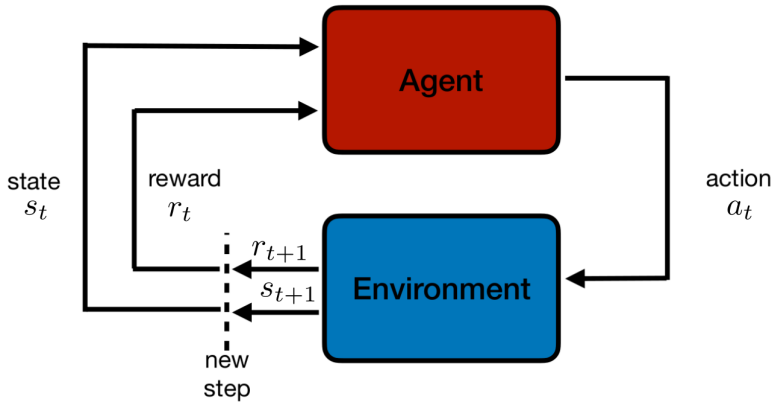


Figure 2.7 – **Markov decision process**: Similar to the agent interaction in (figure 2.1), The agent interacts with an environment by performing an action a_t that transitions its state in the environment from s_t to s_{t+1}. Additionally, the agent receives a reward r_{t+1} as feedback of its interaction with the environment. In Markov Decision Processes (MDP), the agent has access to the full state of the environment and not just to observations of the current environment state.

s contains all the information about the environment, thus, the agent can act optimally. In real-world problems, this assumption can hold in some scenarios, like playing a Chess game. In the game, a player has full access on the board, and he knows the full state of the environment. It doesn't matter what were the previous states in order to take a decision. Now, he should think about the current state to choose the best move in order to transit to a better state with a higher reward. On the other hand, in embodied navigation, the agent doesn't have access to the full state of the environment. As shown in figure 2.1, the agent can partially observe the environment through sensor inputs. It has access only to a 2D projection of the current egocentric view of the 3D environment with the addition of extra sensors' information. Therefore, unlike a chess game, the MDP formulation of navigation problem can't hold, because the agent's current observations do not contain all the current information about the environment. In that case, Partially Observable Markov Decision Processes (POMDP) come to place. POMDPs are a generalization of MDPs, where it is assumed that the agent cannot directly access the full state of the environment. The agent has to maintain a sensor model of the probability of observations given a state. Hence, POMDPs are described as a 7-tuple $\langle \mathbf{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma \rangle$: In addition to the description of MDP, two more attributes are added:

- Ω: Observation space
- $\mathcal{O}$: Observation model (P(o | s, a))

As previously mentioned, in navigation, we have access to observations only o_t instead of the full state s_t. Therefore, in order to apply an estimation technique we need our agent to incorporate the different observations seen to store important information about the environment. This can help to expand increase and its understanding about the current full state of the environment. Usually, this is incorporated by explicit memories as maps (see sections 2.2.1 and 2.3.3) or implicit memories learned with Recurrent Neural Networks (RNN) (Hochreiter et al. 1997; Cho et al. 2014) (see section 2.3.2), or more recently with self-attention over time.

Reinforcement Learning In the Reinforcement Learning (RL) framework, the goal is to make the agent learn a good strategy by performing trials with feedback (rewards). In other words, the main objective is to learn a policy π that maps the observations o_t to actions a_t to maximize the cumulative reward received during a horizon T of interactions between the agent and the environment (Figures 2.1 and 2.7). Learning is easier when dense rewards are chosen (see section 2.3.2 for details on reward shaping in navigation) and not sparse ones (which means the agent is able to receive a reward after each step), which decreases the Credit Assignment Problem (CAP). This problem refers to the problem that measure the impact of an action on the future outcomes (Minsky 1961). It arises when an agent receives a reward for a particular action, but the agent must determine which of its previous actions led to the reward. The problem arises when an agent must determine which of its previous actions led to the current received reward. Is it the last chess move or the group of k last moves? Ideally, when reaching an optimal strategy, the agent has an optimal policy π^* that maximizes the expected sum of cumulative rewards from an initial state s_0 to s_T: $\mathbb{E} = [\sum_{t=0}^{T} r_t]$. We focus during this thesis on finite horizon interactions, where we expect a given interaction episode to end if one of the termination condition occurs: The episode reached a success state (robot reached its target) or the maximum allowed budget of interactions (maximum number of actions) is reached by the agent.

For the optimization problem in RL, under a particular policy $\pi(a|s)$, we sample a trajectory τ of experience (from a given step t for a length of T). The objective is to optimize the discounted reward criterion G_t, for each trajectory independently:

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r(s_t, a_t) \tag{2.11}$$

G_t can be described as the sum of all rewards received from the step t over a horizon of size T discounted by the γ factor. The role of the discount factor is to penalize the future rewards and favor immediate rewards over the long-term rewards besides that long-term rewards may have higher uncertainty (In real life, we value more the immediate certain benefits than the long-term uncertain benefits). Also, in case we tackle the problem in infinite-horizon setting, the discount factor limits the effect of long-term rewards when calculating G_t. Algorithms in

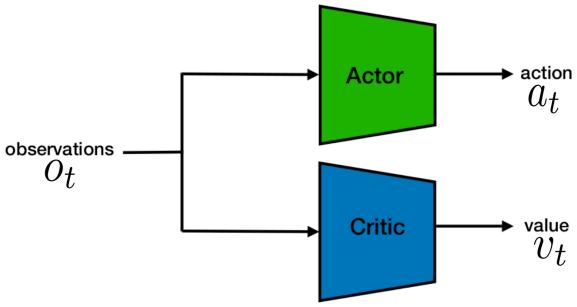


Figure 2.8 – **Actor-Critic Network**: A concept of an actor critic network. A sub-network, the Actor, receives the observations and predict an action, while the other sub-network, the Critic simultaneously receives the same input and predicts a value estimate for the action taken. In this illustration, each network has its unique parameters but they can share parameters in different configurations.

RL aim to optimize for G_t. In order to show some of these algorithms, we first explain two key functions:

$$V^{\pi}(s) = \mathbb{E}_{\pi}[G_t|s] \tag{2.12}$$

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}[G_t|s,a] \tag{2.13}$$

The state-value function, $V^{\pi}(s)$, estimates the expected return to receive if we follow a certain policy π, and we are at a state s. Similarly, the action-value function, $Q^{\pi}(s,a)$ (known also as Q-value) tells what can be the return if we perform action a at state s. We can see that both functions are related. We can recover the state-value function by aggregating the probability distribution over all possible actions and $Q^{\pi}(s,a)$. In that sense, an optimal policy would be the one that achieves the optimal value functions. Some techniques focus on approximating one of the two functions by learning it. Therefore, these function can then be used to sample the best action that maximizes $Q^{\pi}(s,a)$. Another derived function that provides an estimate of the relative improvement of an action in a given state is the advantage function (known also as A-value):

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s) \tag{2.14}$$

It quantifies how much better (or worse) an action is compared to the average expected reward the agent would receive if it followed its current policy.

RL optimization algorithms Many algorithms exist for RL. Some of them are model-based, where a model of the environment is provided or learned simultaneously with our target policy. Other algorithms are model-free, where we focus on learning a policy under the absence of the environment model. In this thesis, we focused on using algorithms in the model-free category, in particular

the policy gradient variants. These algorithms target modeling and optimizing the policy directly with a parameterized function $\pi_\theta(a|s_t)$. We describe three core algorithms that show some variations and tricks used to learn π.

One of the first algorithms in policy optimization is REINFORCE (Williams 2004), also known as Monte Carlo policy gradient. The algorithm updates a parameterized policy distribution using full sample trajectories. Simply, the algorithm samples a trajectory and performs a gradient ascent update step using the current loss function:

$$\mathcal{L}_{REINFORCE} = \sum_{t=0}^{T} G_t \log(\pi(a_t|s_t)) \tag{2.15}$$

The algorithm is simple and easy to understand. It directly estimates the gradients of the expected cumulative reward with respect to the policy parameters. A main drawback of REINFORCE is that it can have high variance and slow convergence, especially in environments with sparse or delayed rewards. To minimize the effect of variance in gradient estimate, a widely used variation in $\mathcal{L}_{REINFORCE}$ is to subtract a baseline value b (it can be calculated as the average of the current discounted rewards) from the return G_t:

$$\mathcal{L}_{REINFORCE} = \sum_{t=0}^{T} (G_t - b) \log(\pi(a_t|s_t)) \tag{2.16}$$

A second algorithm is Advantage Actor Critic (Mnih et al. 2016), also known as A2C or its asynchronous version A3C that focus on parallel training. In, the Actor-Critic approach, the algorithm focuses on learning two parameterized components: the policy (Actor) and the value function (Critic). These models can have separate parameters (figure 2.8) or optionally share some. Depending on the algorithm, the modeling of the value function could be the Q-value or the state-value function. The objective for the critic would be:

$$\mathcal{L}_{critic} = \frac{1}{|\mathcal{D}_k|\, T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_\phi^\pi(s) - G_t)^2. \tag{2.17}$$

As shown in the above equation, the algorithm collected first a batch of trajectories $\mathcal{D}_k$ before the update k on the network, which is not the case in REINFORCE that samples one trajectory per update. The actor is updated in the direction suggested by the critic using the following equation:

$$\mathcal{L}_{actor} = \frac{1}{|\mathcal{D}_k|\, T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} A_\phi^\pi(s,a) \log(\pi_\theta(a_t|s_t)). \tag{2.18}$$

To avoid high variance, the advantage function could be a good option instead of subtracting an average baseline b as in REINFORCE. It forces the update to

go in the direction suggested by the value function (the critic). At the end, the two losses are combined, and we can backpropagate the gradients over the two networks. Optionally, a third loss, the entropy can be added next to the two losses. It helps to limit high drift of the newly updated policy with respect to the old policy:

$$\mathcal{L}_{entropy} = -\frac{1}{|\mathcal{D}_k|\,T}\sum_{\tau\in\mathcal{D}_k}\sum_{t=0}^{T}\sum_{i=0}^{n}\pi_\theta(a_i|s_t)\log(\pi_\theta(a_i|s_t)). \tag{2.19}$$

$$\mathcal{L}_{actor-critic} = \lambda_{actor}\mathcal{L}_{actor} + \lambda_{critic}\mathcal{L}_{critic} + \lambda_{entropy}\mathcal{L}_{entropy} \tag{2.20}$$

Here, λ coefficients are hyper-paremeters. Since, by default, we perform a gradient ascent update on $\mathcal{L}_{actor}$, usually we attribute negative value to λ_{actor} to perform a single gradient descent using the total loss $\mathcal{L}_{actor-critic}$.

Another variant is Proximal Policy Optimization (PPO) (Schulman et al. 2017b). It seeks to mitigate some issues in basic policy gradient methods. It tries to avoid parameter updates that change the policy too much at one step. PPO introduces the "proximal" zone for updates. A previous algorithm by Schulman et al. (2017a) has already introduced a similar concept of "Trust Region" for policy optimization (TRPO). The drawback with TRPO is its computationally complexity. It needs to solve constrained second-order optimization problems in each iteration by enforcing a KL divergence constraint on the size of the policy update at each iteration. PPO provides a simpler alternative to this step. It applies a clipping function to $\mathcal{L}_{actor}$ (equation 2.21) as shown in algorithm 2.1 - line 6. In addition, PPO uses the same batch of trajectories to make several updates. This improves the sample efficiency while empirically demonstrating policy stability and better performing compared to the former policy gradient algorithms. A decentralized and distributed version of PPO has been proposed under Wijmans et al. (2019) called DD-PPO to solve the task of *PointNav*, which leads to our knowledge to state-of-the art performance in this task in simulation-only benchmarks.

Imitation Learning is a framework of learning a policy π_θ from demonstrations of an expert policy π^*. Most of the time, the demonstrations are presented in the form of sequence of state-action pairs (o_t, a_t) for every episode trajectory τ. Each pair indicates the action a_t to take when observations o_t are received. One simple form of imitation learning is to use supervised learning. This is known as Behavior Cloning (BC). Given the expert's demonstrations, we treat all pairs as i.i.d (Independent and Identical Distribution) and we try to learn in a supervised manner a generalized function that maps a given observation o_t to its corresponding action a_t. Therefore, we treat the action as the target label for each state (observation). Hence, for a given dataset of demonstrations

Algorithm 2.1 Proximal Policy Optimization PPO

1: Input: initial policy π_0 (actor) with parameters θ_0 and intial value function V_0 (critic) with parameters ϕ_0.
2: **for** k = 0, 1, 2,... **do**
3: Run actor π_k in the environment to collect dataset of N trajectories $\mathcal{D}_k = \{\tau_1, \tau_2, ...\tau_N\}$.
4: Compute cumulative rewards G_t.
5: Based on the value function V_k, compute advantage estimates $\widehat{A}_t$.
6: Calculate PPO-Clip objective (actor loss):

$$\mathcal{L}_{actor} = \frac{1}{|\mathcal{D}_k|\, T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left[\min \left(r_t(\theta)\hat{A}_t, \mathcal{C}(r_t(\theta), \epsilon)\hat{A}_t \right) \right]. \tag{2.21}$$

where $\mathcal{C}(r_t(\theta), \epsilon) = \text{clip}\,(r_t(\theta), 1-\epsilon, 1+\epsilon)$, $r_t(\theta) = \frac{\pi_k(a_t|s_t)}{\pi_{k-1}(a_t|s_t)}$. s_t and a_t are, respectively the state and action at time t within the trajectory.
7: Calculate the value regression objective (critic loss):

$$\mathcal{L}_{critic} = \frac{1}{|\mathcal{D}_k|\, T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} (V_k(s_t) - G_t)^2. \tag{2.22}$$

8: Update actor by $\mathcal{L}_{actor}$ using gradient ascent and critic by $\mathcal{L}_{critic}$ using gradient descent
9: **end for**

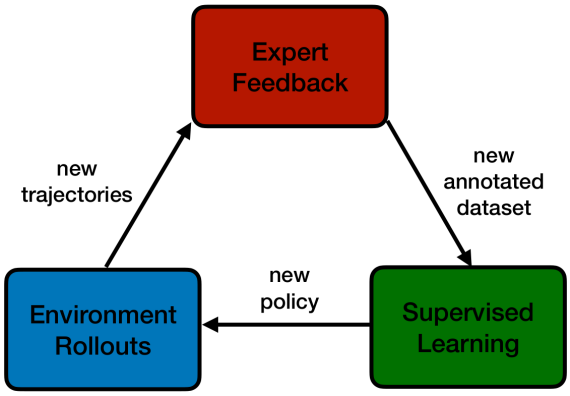


Figure 2.9 – **Direct Policy learning** At training time, an interactive demonstrator that can provide training data from the expert policy feedbacks (decisions) on the rollout trajectory done by the learning policy in the environment. This loop continues until the learning policy converges.

$\mathcal{T} = \{\tau^{(i)}\}_{i=1}^{N}$, the optimal parameters θ^* should minimize a cross-entropy loss as in the following equation:

$$\theta^* = \arg\min_\theta \sum_{i=1}^{N} \sum_{(s_t, a_t) \in \tau^{(i)}} \log(\pi_\theta(a_t|s_t)) \tag{2.23}$$

Although, the vanilla BC framework is simple, it can still be quite problematic, specially for applications that require long-term planning and in long-horizon tasks. The main reason for that is the i.i.d assumption breaks the assumption of MDP that a next state is the result of a previous state given a certain action. Moreover, if an error is made at a certain state, it will add up. An agent can easily put itself into a state that it has never been visited by the expert, in another word, the agent's policy has never seen this example during training. In such situation, the behaviour is considered unknown and can accumulate with time which can lead to catastrophic behavior. An improved version of BC is called Direct Policy Learning (DPL) (Figure 2.9) where an interactive method is used for generating the demonstration. At the beginning an initial policy based on the expert is used to collect the demonstrations. This dataset is used to train the policy π_θ using supervised learning, then at each iteration, we collect new trajectories by rolling out the current version of the policy π_θ and collecting the feedbacks from the expert to know what would be the optimal action to take in the same state and we train with this new dataset. Some known variation is used as Data Aggregation (DAgger) (Ross et al. 2010), where at each iteration the policy is trained on the new dataset in addition to all the previous dataset.

Another possible way to leverage the training from demonstrations is to combine both frameworks. One option can be done by pretraining the policy using demonstrations and finetuning it using a reinforcement learning algorithm (Ramrakhya et al. 2023). Another option can be to reward-labelling the transition action of successful demonstrations and use them in an RL training (Martin et al. 2022).

Inverse Reinforcement Learning Another form of Imitation Learning (IL), is Inverse Reinforcement Learning (IRL). It treats the expert's demonstrated actions as a sequence of decisions, hence, it drops the i.i.d strong assumption. The main idea is to learn an optimal parameterized reward function r_ϕ of the environment under which the demonstrated actions are optimal. Then, this reward is used to learn a policy using RL. The process of tuning the parameters ϕ of the reward function stops when reaching a satisfactory policy π that has a close behavior to the expert policy π^*.

2.2 Classical robotics in Navigation

Classical robotics techniques tend to adopt a modular approach by dividing the task of autonomously navigating an environment into sub-skills: Perception, Mapping, Localization and Planning. They try to solve for these skills individually. The underlying algorithms are often based on explicit models and representations as well as optimization algorithms. A large body of work is based on probabilistic models (Thrun et al. 2005) in an analytical and probabilistic way. In this thesis,

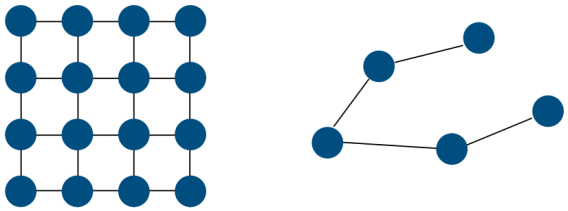


Figure 2.10 – **Map representations** Two widely used representations: Metric (left) and Topological (right) maps. The metric variant has a unique structured representation, as a grid, where each pair of nodes are distant by a predefined distance. Nodes are commonly named "grid cells" in that case. While, in topological representation is more sparse where each pair of nodes can have a unique distance between each other, or they may not be connected at all.

we are interested in the classical algorithms that perform mapping and planning for a navigation agent in order to reach a certain target. Therefore, in this section we describe first how, classically, the robot represents the environment as map. Second, how the map is built and finally leveraging this map, how the robot uses it to reach a certain goal by planning.

2.2.1 Maps

Mapping the surrounding world for an embodied agent is crucial for an efficient navigation. Hence, predefining the map representation and storage can lead to performance increase on the task level and computation level. Therefore, in order to define a map, we need to define its representation (how the map is presented and stored) and its storage (what information stored inside the map).

Map representation Two widely used variants to represent the environment are metric and topological representations (Figure 2.10). Metric maps can be presented as a 2D or 3D tensor of grid cells, $M \in \mathbb{R}^{H \times W}$. Every element (cell) in the matrix represents a precise location in the environment. The precision of the map (map resolution) corresponds to the size (in m^2) of a single cell on the grid (a node). The map resolution is inversely proportional to the cell size. A low-resolution map is presented by less cells because each cell represent a bigger area in the real environment. In that sense, the choice of map resolution affects the robot navigation while planning (Section 2.1.3.1). Higher resolution offers better paths, at a cost of a higher computation complexity. On the other side, in order to tackle the problem of computation time, topological maps were proposed. Topological maps are presented as graphs $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ is the set of chosen nodes representing specific locations of the environment and $\mathcal{E}$ is the set

of valid connections (valid movement/straight-path) between nodes. Topological maps define a set of distinct places and a set of quantitative relations between them. A common claim states that humans often use topological information for navigation. For large environments, topological maps tend to be the best option, since they limit the computation time for planning to a specific set of locations. Since topological maps don't represent the whole environment location, they don't guarantee that the planned path is the optimal path for the given explored area, comparing to the grid maps representation for the same area. But, with the rise of high computing resources, the metric map are also being used.

Map storage Whether the map is grid-based (metric) or node-based (topological), the data saved inside the grid/node can vary. It depends on the available inputs from the environment, the required data to be stored and the current limitations of storage and computation capabilities. Information can be raw data or processed data, also it can be geometrical or visual. Spatial coordinates in the environment are considered as raw geometrical data. By definition, a grid-based map implicitly contains the coordinates of the location while in node-based representations, positional/metrical information should be stored inside the node if needed (Angeli et al. 2009). One frequent and important information that is stored inside grid-based map is occupancy: in every cell, a boolean or a probability is assigned to specify whether a location is free or occupied. In the case of a boolean occupancy map, the map can be visualized as a gray scale image $M \in \mathbb{R}^{1\times H\times W}$ and the occupancy information can be processed from the input sensory data from depth camera (Figure 2.3a) or LIDAR (Figure 2.2b). Additionally, featured-based information such as SIFT features (Lowe 2004) or neural embeddings (learned-based) (Beeching et al. 2020a; Wani et al. 2020) can be stored.

2.2.2 Simultaneous Localization and Mapping (SLAM)

Motivation The SLAM problem (Figure 2.11) addresses a main challenge of a robot navigating in an unknown environment. The robots need to build a map for this environment, while at the same time it needs to localize itself on the built map. Although the pose estimate of the robot can be maintained through **dead reckoning** (see sensors description in section 2.1.1), SLAM techniques can offer a more robust estimate of the current pose of the robot. Over the years, various SLAM variants have been developed to address different challenges and requirements. First, we roughly present the different categories and assumptions taken in order to build a SLAM system. Then we present two predominant categories in SLAM: Feature-based and Dense SLAM. Then, we provide a more detailed description of widely used probabilistic formulations with different variants.

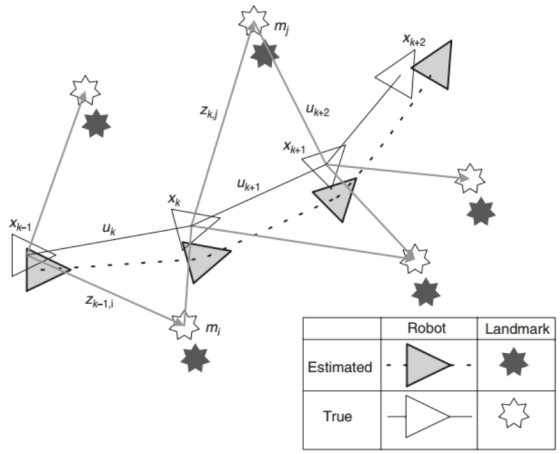


Figure 2.11 – **SLAM - foundation problem:** A SLAM algorithm estimates simultaneously the position of landmark locations (**Mapping**) and the robot's position (**localization**). The true locations (in white) are not measured directly, and the robot can't have access to them. The figure is reproduced from Durrant-Whyte et al. (2006).

Different assumptions Some assumptions are underlined explicitly when designing a SLAM algorithm. *Dense Versus Feature-Based*: some algorithms sample the map in a high resolution to enable a photorealistic reconstruction of the environment, requiring higher computational complexity. On the other side, feature-based techniques are more efficient since they rely on sparser features from the sensors. *Static Versus Dynamic*: Static methods assume that the environment doesn't change over time, while dynamic methods accept changes in the environment. The latters don't treat the dynamic effects as outliers and tend to be more robust. *Active Versus Passive*: In passive algorithms, an external module is responsible for controlling the robot to explore and observe the scene. In active algorithms, the robot tries to explore the environment efficiently to map the environment. Such algorithms are harder to design, but yield more accurate maps in less time. *Topological Versus Metric*: This assumption is related to the map representation (discussed in section 2.2.1). Another assumption about the way of processing of the stream of inputs: *full versus online*. Full SLAM version estimates the robot poses for the entire robot path along with the map. On the other side, online version focuses on recovering the current robot location and updating the map. Online algorithms are usually incremental and can process one data item at a time.

Feature-based SLAM These methods operate by detecting and tracking distinctive features or keypoints in the environment, such as corners, edges, or distinctive points. These features are typically extracted from sensor data like images or laser scans. One of the well-known feature-based SLAM systems is ORB-SLAM (Mur-Artal et al. 2015; Mur-Artal et al. 2017), which stands for Oriented FAST

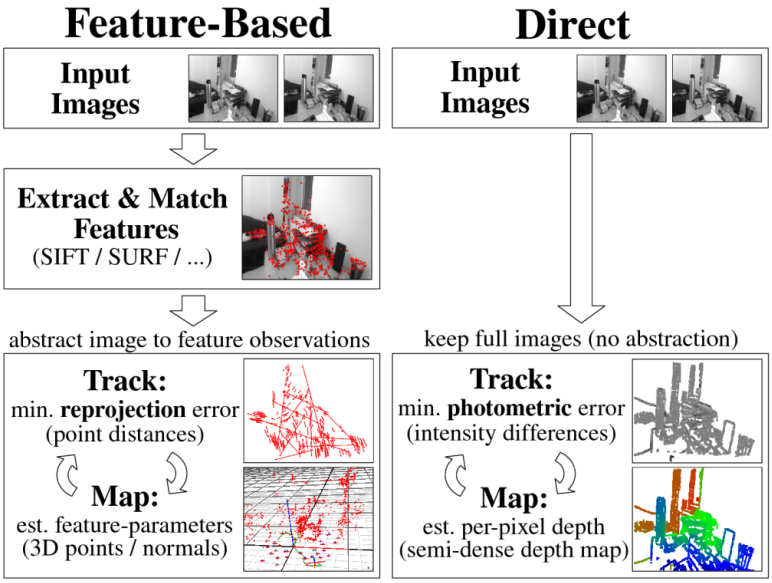


Figure 2.12 – **Feature-Based Vs Direct SLAM:** Differences between both methods. Feature-based techniques add an extra step of extracting and matching features. Also, they use difference tracking and mapping techniques. The figure is reproduced from Schöps et al. (2014).

and Rotated BRIEF SLAM. ORB-SLAM starts by extracting distinctive features, also known as keypoints, from the camera images. These keypoints are chosen based on their high repeatability and distinctiveness across frames. The system employs the FAST feature detector (Rosten et al. 2005), which identifies potential keypoints based on pixel intensity differences in corners. It also uses the BRIEF descriptors (Calonder et al. 2010) to track keypoints in the camera images. BRIEF are binary descriptors that capture the appearance and geometry of the keypoints. Therefore, the SLAM system initializes itself by detecting enough keypoints in the initial frames and identifying their corresponding matches. It estimates the initial camera pose and starts building a sparse map. As the camera moves, ORB-SLAM continues to track keypoints and estimates camera poses relative to the initial frame using bundle adjustment (Triggs et al. 1999). In addition, ORB-SLAM has the ability to detect loop closures. A loop closure occurs when the camera captures a previously visited location. The system detects loop closures by recognizing that previously observed keypoints reappear in the current frame. This step helps correct accumulated errors in the map and the poses trajectory and improves accuracy.

Dense SLAM aims, on the other hand, to create a dense representation of the environment by estimating the depth or distance values for every pixel or point in the sensor data. Their optimization techniques rely on minimizing the

photometric error per pixel. Unlike feature-based methods that focus on distinct keypoints, dense SLAM methods operate on a per-pixel basis, providing a more detailed map of the environment. Dense methods typically appear in applications where accurate geometric information is required, such as augmented reality, virtual reality, and 3D reconstruction. In this category we can find systems like KinectFusion (Newcombe et al. 2011a) and DTAM (Dense Tracking and Mapping) of Newcombe et al. (2011b).

Problem formulation Based on the handbook of robotics (Siciliano et al. 2007), the problem is best described in probabilistic terminology. Let the robot pose be denoted as x_t at a time t. x_t is usually a three-dimensional vector in $\mathbb{R}^3$. It contains the two-dimensional coordinates of the robot's plane, in addition to an angle value for the robot's orientation. Then, the whole sequence of poses (path) in a horizon of length T (T might be ∞) can be given as:

$$X_T = x_0, x_1, x_2, ..., x_T. \tag{2.24}$$

where x_0 is often referring to an initial reference point for the estimation of subsequent unknown poses. One of the goal of SLAM algorithms is to estimate them.

Let u_t denote the odometry that represents the relative motion between time $t-1$ and time t. As mentioned before, such information can be obtained from the robot's motion sensors or the input control given to the wheels' motors. Therefore, the odometry sequence for T steps, is given as:

$$U_T = u_1, u_2, ..., u_T. \tag{2.25}$$

In a perfect world, with noiseless motion, having U_T should be sufficient to recover the sequence X_T. However, measuring odometry is usually imperfect and leads to drift.

The second subproblem, is to map the environment. Therefore, let M denote the map of the environment. Different representations of the map are possible (see section 2.2.1). The robot gets sensory information at each x_t. These measurements can be denoted as z_t, they represent the information between features in M and the current pose x_t. As before, the sequence of measurements is given as:

$$Z_T = z_1, z_2, ..., z_T. \tag{2.26}$$

Now, the SLAM problem is to recover the map M of the environment (M can be represented as a graph of landmarks or a metric representation – see section 2.2.1), and the sequence of robot poses X_T using the two available readings: Odometry U_T and sensory measurements Z_T. Therefore, we can now model the recovering problem as a problem of estimation of the posterior probability over the robot path together with the map

$$p(X_T, M | Z_T, U_T) \tag{2.27}$$

By writing the posterior probability in this way, we target the *full* SLAM problem, which estimates the *entire* path. In this case, the proposed approach often processes the data by batching (all data available at the same time). As previously discussed, the alternative problem, is the *online* version of SLAM. It seeks to recover the current robot pose instead of the entire path. In this setup, the algorithms are usually incremental and process the data readings one by one at a time. Therefore, the posterior probability is modeled as follows:

$$p(x_t, M|Z_t, U_t) \tag{2.28}$$

In both cases, whether full or online setup, we can notice that the condition variables are all directly observable to the robot, which facilitates the estimation of the targeted variables on the left. But, in order to solve the problem using Bayes rule, we need to have two more mathematical models:

$$p(x_t|x_{t-1}, u_t) = g(x_{t-1}, u_t) + \epsilon_x \tag{2.29}$$

$$p(z_t|x_t, M) = h(x_t, M) + \epsilon_z \tag{2.30}$$

The first model (equation 2.29) is called the motion model that relates to u_t, the previous pose x_t and current robot poses x_t. The model is derived from a kinematic model g of robot motion. The second model (equation 2.30) relates z_t to the environment M and the current robot pose x_t. The model is derived from a certain measurement function h that relies on sensor readings. Both probability distributions peak at the noise-free case when the noise models, ϵ_x and ϵ_z are equal to zero.

Every SLAM algorithm can model the problem differently while keeping the same relations between observable variables and estimated variables. The main target remains the estimation of noises ϵ in order to correct the final estimated values for the robot poses and the map.

Different optimization approaches Historically, there are three basic SLAM lines of work that try to solve the problem (solve for 2.27 or 2.28). Most of the works done are derived from these approaches:

1. **Extended Kalman Filters (EKF) SLAM**: It was introduced in (Smith et al. 1987; Moutarlier et al. 1989). EKF formulation is the earliest and most influential in the problem. Basically, it proposes the use of a single state vector to estimate robot poses and a set of N representative features of the environment (N observable landmarks in the map), with an associated error covariance matrix $\mathbf{\Sigma_t}$. The covariance matrix represents the uncertainty in the estimation, alongside the correlations between the features and the robot poses. During the navigation of the robot in the environment, the robot takes the measurements and the system state vector with the covariance matrix

are being updated using the extended version of the Kalman filter (Kálmán 1960; Jazwinski 1970) by performing a single linearization step on the g and h functions (equations 2.29 and 2.30) using Taylor series expansion. This is a strong assumption. Moreover, the robot estimation can be represented as a multivariate Gaussian:

$$p(x_t, M|Z_t, U_t) = \mathcal{N}(\mathbf{u_t}, \mathbf{\Sigma_t}) \tag{2.31}$$

where the vector $\mathbf{u_t}$ is the robot's best estimation of its current pose x_t and the location of the features in the environment. The dimension of $\mathbf{u_t}$ is $3 + 2N$ (3 for robot pose and $2N$ for the features locations in the 2D map). Hence, $\mathbf{\Sigma_t}$ should be of size $(3 + 2N)$x$(3 + 2N)$. One main limitation of the EKF technique comes from the approximation to a linear model (linearization step), but another concern is related to the computation complexity as the size of $\mathbf{\Sigma_t}$ grows quadratically as new features are observed. Follow-up works proposed some extensions to enhance the vanilla EKF. One way to tackle the quadratic growth nature of the covariance matrix is through a submap decomposition of the environment (Leonard et al. 2000; Williams et al. 2002). Other work (Thrun et al. 2004; Walter et al. 2007) involves the Extended Information Filter, which deals with the inverse of $\mathbf{\Sigma_t}$. The key motivation is that $\mathbf{\Sigma_t}$ is a dense matrix, while the information matrix is sparse when the full robot has the trajectory maintained. This modification leads to more efficient algorithms.

2. **Particle Methods** SLAM: This line of work is based on particle filters (Metropolis et al. 1949). The basic idea behind the particle filters is to represent a posterior estimation through a set of guesses (particles). Each particle k can be thought as a concrete guess to what is the value of a specific state $\mathbf{u_t^k}$. In perfect conditions, if the number of particles tends to infinity, the particle filter approaches the true posterior distribution for state estimation by representing a weighted multimodal distribution. Doucet et al. (2000) were the first to introduce particle filters into SLAM followed by FastSLAM (Montemerlo et al. 2002). FastSLAM maintains K particles, each particle contains an estimate for the robot pose $X_t^{[k]}$ and a set of N 2-D Gaussians, one for each environment feature (landmark):

$$X_t^{[k]}, \mathbf{u_{t,1}^{[k]}}, ..., \mathbf{u_{t,N}^{[k]}}, \mathbf{\Sigma_{t,1}^{[k]}}, ..., \mathbf{\Sigma_{t,N}^{[k]}} \tag{2.32}$$

When a new odometry measurement is received, FastSLAM generates new pose variables stochastically for each particle using the distribution based on the motion model. This step called *transition*.

$$x_t^{[k]} \approx p(x_t|x_t^{[k]}, u_t) \tag{2.33}$$

The next step is the *evidence* step. After a new measurement z_t is received (new evidence), the importance of particles needs to be adjusted. A weight $w_t^{[k]}$ is assigned to each particle to measure its importance. Hence, $w_t^{[k]}$ is called *importance weight*.

$$w_t^{[k]} = \mathcal{N}(z_t | x_t^{[k]}, \mathbf{u}_{\mathbf{t,N}}^{[\mathbf{k}]}, \mathbf{\Sigma}_{\mathbf{t,N}}^{[\mathbf{k}]}) \tag{2.34}$$

where $w_t^{[k]}$ represents the importance of a particle k at time t in the presence of a new measurement. All importance weights are then normalized to sum to 1. Then, FastSLAM proceeds in a new step (the *resampling* step), by replacing from the set of existing particles with new particles using the importance weights. Finally, the mean $\mathbf{u}_{\mathbf{t,n}}^{[\mathbf{k}]}$ and covariance $\mathbf{\Sigma}_{\mathbf{t,n}}^{[\mathbf{k}]}$ for the new set of particles are updated using the standard EKF updates.

FastSLAM approximately solves the full SLAM posterior. It applies the Rao-Blackwellization technique Blackwell (1947). It samples from the poses posterior $p(X_t^k | U_t, Z_t)$ and represents the map posterior as $p(M | X_t^k, U_t, Z_t)$ in Gaussian form.

3. **Graph-Based SLAM**: Graph-based techniques are based on solving the problem using nonlinear sparse optimization. They represent the robot poses as nodes in a graph. Edges between a pair of robot poses x_{t-1} and x_t represent the odometry readings u_t. On the other hand, landmarks m_i can be represented as nodes with an estimated pose that depends on sensed measurements z_t at x_t, in that case an edge is added between m_i and x_t. Most of the graph-based methods are offline methods. Therefore, once a graph is constructed, an optimization technique is applied to find the best X_T^* and m^* that maximize $\log p(X_T, m | Z_T, U_T)$. Under the Gaussian noise assumption, this logarithm can be represented in a system of equations:

$$\begin{aligned} \log p(X_T, m | Z_T, U_T) &= const + \sum_t \log p(x_t | x_{t-1}, u_t) + \sum_t \log p(z_t | x_t, m) \\ &\geq const + \sum_t [x_t - g(x_{t-1}, u_t)]^T R_t^{-1} [x_t - g(x_{t-1}, u_t)] \\ &+ \sum_t [z_t - h(x_t, m)]^T Q_t^{-1} [z_t - h(x_t, m)] \end{aligned} \tag{2.35}$$

In this quadratic form, the first sum of squares error term represents the error in odometry readings, while the second term represents the error in measurements readings. The quadratic form can be solved using multiple optimization techniques such as iterative methods as gradient descent, conjugate gradients or direct methods as QR decomposition and sparse Cholesky. Graphical-based methods show advantage over EKF. In EKF the covariance matrix consumes space and time quadratically given the size of

the map, while in graphical methods, the memory size is linear and the updating the graph is constant. This advantages yield to scale graphical to high-dimensional maps. Historically, the graph-based method has generated some of the largest SLAM maps ever built in big outdoor environments.

2.3 End-to-End Neural Agents

The navigation task can be framed as a learning problem. Using a large amount of data, a learning-based agent leverages the pattern recognition and correlation abilities of deep neural networks to extract knowledge and regularities that can help the agent to take optimal decisions toward optimizing for a certain goal in certain environments. Formulating the navigation task as a learning problem can be done through IL or RL. Both settings require access to large scale data. Therefore, in this section, we introduce first the different types of existing datasets (Section 2.3.1). Second, we present how learning-based techniques are used to train navigation agents (Section 2.3.2). Third, we show how maps can boost the performance of End-to-End agents (Section 2.3.3). Forth, we present different extensions to improve the training strategies (Section 2.3.4).

2.3.1 Training Data

Training and testing robots in real environments is not cost-efficient or even quite expensive, dangerous (robots can be broken or injure others in some scenarios), time-consuming (it cannot be faster than real-time and impossible to parallelize) and hard to reproduce (hard to replicate exactly specific conditions, hence hard to conduct fair comparisons). In addition, collecting data from real environments and using it offline, despite that it doesn't solve always the slowness problem, is nor practical neither scalable for generalization and cannot cover all the various downstream tasks in navigation. Hence, existing work tries to leverage data from two main sources. Data rendered in simulators, especially photorealistic simulators (Savva et al. 2019; Xia et al. 2018) that are visually close to the real setup, and real egocentric videos that are available on the internet (Chang et al. 2020).

Photorealistic simulators A simulator consists of (1) a *Rendering engine* (e.g. Magnum (Vondrus et al. 2020), PyRender (Matl 2020)) that generates observations o_t from a current states s_t in an environment and (2) a *Physics engine* (e.g. PyBullet (Coumans et al. 2016), MuJoCo (Todorov et al. 2012))that simulates the physical evolution of objects from an environment state s_t to s_{t+1}. Some simulators integrate the existing rendering and physics libraries (Xia et al. 2020; Xiang et al. 2020; Shen et al. 2020) or are built on top of game engines (Kolve et al. 2017; Yan

et al. 2018; Puig et al. 2018). Relying on game engines is not necessarily an optimal solution since they are more optimized for human usage that requires low frames per second (=60 FPS), display dependent and high image-resolution. On the other hand, in Embodied AI, the needs (Choi et al. 2020) tend to favor high FPS (10k+ FPS), headless deployment (for training and evaluation on remote clusters), and relatively low-resolution images. In indoor navigation, we focus on photorealistic simulators that render indoor environments from large-scale 3D scene datasets of meshes (Armeni et al. 2016; Chang et al. 2018; Xia et al. 2018; Yan et al. 2018). Such high-speed photorealistic simulators can help to solve the challenges of training mentioned above. They can run by high order of magnitude faster than real-time, up to 10k times faster for some simulators (Szot et al. 2021). They can be parallelized over clusters. In addition, resource-wise, training in simulation is cheap and safe. Also, it can enable reproducibility where we can conduct benchmarking and fair comparison among different navigation algorithms.

Habitat Simulator Habitat AI (Savva et al. 2019) is currently one of the most widely used simulators for the introduced navigation tasks (Section 2.1). It is used in most of the recent work that relies on learning-based modules due to its high frame rate, which can go up to 1k FPS. Habitat excels also in its flexibility and modularity. The simulator decouples tasks, simulation, and task/episode management, making it agnostic to 3D scene datasets and tasks. Therefore, agents can be transferred from one task to the other easily and to switch seamlessly between different datasets on the same training and evaluation procedure. It gives a parameterized flexible control on the agent configurations and environment states allowing the users to specify specific embodied agent characteristics (geometry, physics and actuation) and apply changes in the environment state. For example, in the simulator level, noises can be applied on the environment and agent actuators.

2.3.2 End-to-End Training

Navigation agents can be trained end-to-end by optimizing a neural policy π that takes the observations o_t and predicts the best current action to perform a_t in order to reach its current target. For our visual navigation problem, a typical end-to-end action prediction can be described as follows:

$$p(a_t) = \pi_\theta(v_\gamma(o_t), g_t, a_{t-1}) \tag{2.36}$$

Where v_γ is the visual encoder the agent's observations, g_t is the current target and a_{t-1} the previous action. This is a typical configuration for a reactive (memoryless) agent. The agent doesn't store any information related to the history of its

current episode. In order to add memorization capacity to the agent, a recurrent neural network $f_{recurrent}$ can be added as follows:

$$h_t = f_{recurrent}(h_{t-1}, v_\gamma(o_t), g_t, a_{t-1}) \tag{2.37}$$
$$p(a_t) = \pi_\theta(h_t) \tag{2.38}$$

where h_t is the current hidden state of the recurrent neural network. Another alternative is to apply transformers as a self-attention module over time on the history of observations (Pashevich et al. 2021). The end-to-end policies can be trained with reinforcement learning (Zhu et al. 2016). In that case a good reward function has to be designed. Also, the policy can be trained using imitation learning by collecting expert shortest-path demonstrations for every training episodes using rollouts inside the simulator.

Reward shaping for RL is the design of a reward function that provides a representative and frequent feedback signal on agent behavior. Reward shaping is crucial for RL training to converge. A noisy reward signal can destabilize training. Also, a sparse signal can make the training longer because the agent receives its signal after a potentially long sequence of actions (usually, when the episode terminates) therefore the agent needs more training time to explore the best strategies to reach its goal. In the context of navigation, a common reward function is shown in eq. 2.39

$$r_t = \mathbb{1}_t^{\text{reached}} \cdot r^{\text{goal}} + r_t^{\text{geo-dist}} + r^{\text{time-penalty}} \tag{2.39}$$

where r^{goal} is a constant reward that is gained if and only if the agent reached the navigation target, it's obviously a sparse reward that the agent received once at the end of the episode. The dense reward $r_t^{\text{geo-dist}} = d_t - d_{t-1}$ represents the difference in reward on approaching the goal, by calculating the difference in geodesic distance to the goal d between step $t-1$ and t. $r^{\text{time-penalty}}$ is a negative slack reward that encourages the agent to reach the goal faster.

Another reward shaping is done in exploration task (Chen et al. 2019; Chaplot et al. 2020b; Beeching et al. 2020c) to represent the explored area of the scene at given step t:

$$r_t = r_t^{\text{cov}} + r_t^{\text{col}} \tag{2.40}$$

where $r_t^{\text{col}} = -Bump(t+1)$ and $Bump(t+1)$ denotes if a collision occurred while executing action a_t. $r_t^{\text{cov}} = C(M_{t+1}) - C(M_t)$ denotes the coverage reward where the coverage $C(M_t)$ is defined as the total area in the map that is explored at time t and currently known as traversable or non-traversable. The difference in coverage is the actual reward we gain, since it measures how much we gain in coverage after each action taken for exploration.

Leverage IL with human demonstrations Prior works explored the usage of IL (specifically BC) from expert demonstrations in multiple interactive tasks such as videos games (Kanervisto et al. 2020b; Kanervisto et al. 2020a) and autonomous driving (Samak et al. 2021; Codevilla et al. 2019; Bojarski et al. 2017). In *ObjectNav* navigation, the Habitat-Web initiative from Ramrakhya et al. (2022) performs large scale training from human demonstrations instead of shortest-path expert navigators. In this work, they develop the Habitat-WebGL infrastructure. A virtual tele-operation data-collection infrastructure that connects Habitat simulator (Section 2.3.1) running in a web browser to Amazon Mechanical Turk (a crowdsourcing website). The infrastructure allows remote users to tele-operate virtual agents at scale. They were able to collect 80k demonstrations for the *ObjectNav* task (to our knowledge, the largest human demonstrations' dataset for robot navigation tasks with 29.3M actions). As a result, the Habitat-Web IL agent trained with 70k demonstrations outperforms the RL agent trained on 240k trajectories. Although the average of the human demonstrations has lower performance than the shortest-path expert (they perform longer episode steps, and eventually lower SPL results), they show some advantages since the human navigation experiences leverage more exploration skills than shortest-path experts. Moreover, the statistics from Habitat-Web dataset show that the human demonstrations tend to have a more uniform action distribution and navigability coverage and sight coverage are respectively 3-4x and 3x higher than the one obtained by shortest-path experts.

In follow-up work, Ramrakhya et al. (2023) investigate the possibility of using both frameworks: IL and RL during training. They present a two-stage fine-tuning scheme of using BC pretraining on human demonstrations followed by RL-finetuning (BC $\rightarrow$ RL). The End-to-End agent is taken from (Yadav et al. 2022) and has an actor-critic architecture (Section 2.1.3.2). After the actor branch is trained using BC, the two-stage fine-tuning starts. In the first phase (Critic Learning phase), they focus on training the critic branch only. Since the critic wasn't trained in BC as the actor branch was, training the critic on successful episodes of human demonstrations would be overly optimistic. Another reason to train first the critic before jumping to full actor-critic training with a random weights for the critic, would potentially lead to drop in performance because of the poor value estimates that can affect the actor weights updates. Therefore, their solution was to provide rollouts of the BC pretrained agent to train the critic-head only until the loss plateaus. In the second phase (Interactive Learning phase), both,actor and critic, are trained. They adapted a specific learning rate schedule: decaying the critic learning rate while increasing the actor learning rate (starting from zero), both with the same rate until they reach the same value. Then, they keep both at this value until the end of the training. The key findings of their fine-tuning curriculum (BC $\rightarrow$ RL) is that applying BC $\rightarrow$ RL using human demonstrations outperforms BC $\rightarrow$ RL using generated shortest-path demonstrations and task-

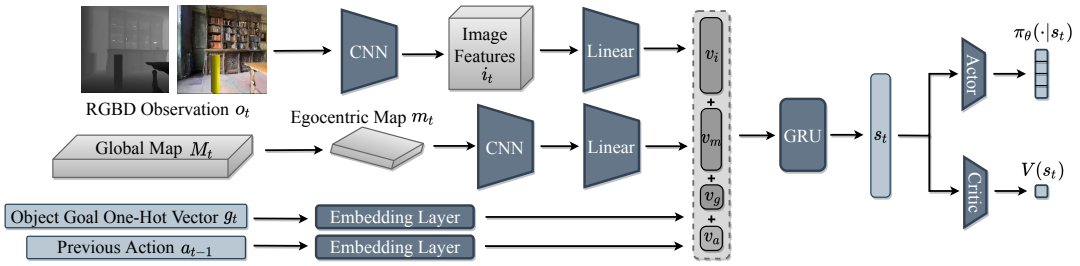


Figure 2.13 – **End-to-End training with mapping** During training with PPO, at each time step t, the agent encodes visual observation o_t, an egocentric view m_t of a global map M_t, the current goal g_t and the previous action a_t and concatenate the embeddings to be passed to a recurrent layer Gated Recurrent Unit (GRU). The output recurrent state is the passed to actor-critic architecture for action prediction and value prediction. The figure is reproduced from Wani et al. (2020).

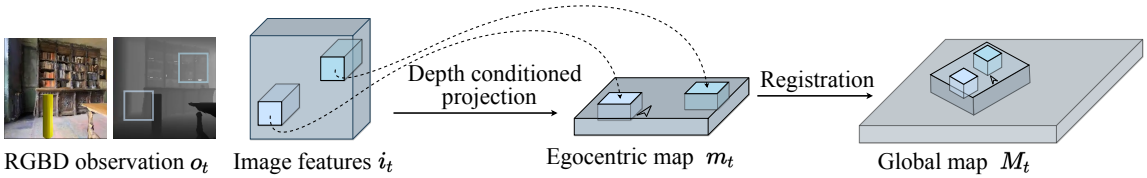


Figure 2.14 – **Features projection & registration module** RGB-D observation $o_t(i,j)$ are encoded into neural images features $i_t(i,j,.)$ which are then projected using the depth input $d_t(i,j)$ into an egocentric map m_t. m_t (agent is at the mid-bottom) is then registered onto the global map M_t via registration function R(m_t, M_t, p_t), where p_t is the agent's episodic position and orientation. The neural features are integrated into the map cells using element-wise max-pooling. The figure is reproduced from Wani et al. (2020).

agnostic frontier exploration demonstrations. Also, they found that increasing the size of the dataset used in BC-only phase doesn't have the same improvements on BC $\rightarrow$ RL fine-tuning. 90% of the best performance finetuned policy can be reached with less of 50% of the existing demonstrations in the experiments. The effect of using more demonstrations is not linear to the final performance of BC $\rightarrow$ RL finetuned policy.

2.3.3 Training with mapping

As shown in equation 2.37, the RL agent can be supplied with a recurrent memory that acts as an implicit memory. But, this example of recurrent memory has

no structure so far, and doesn't provide any spatial structure of the environment other than the one that automatically emerges from training. Therefore, on way is to provide a map m_t (e.g. ego-centric metric map) as a spatial cumulative memory of the current observed environment so far. This can be done by projecting the observed first person view and keep tracking of the motion of the agent in the environment:

$$m_t = T_{u_{t-1}}(m_{t-1}) + P^{-1}(v_\gamma(o_t)) \tag{2.41}$$
$$h_t = f_{recurrent}(h_{t-1}, v_\gamma(o_t), g_t, a_{t-1}, f_\phi(m_t)) \tag{2.42}$$
$$p(a_t) = \pi_\theta(h_t) \tag{2.43}$$

where m_t being an ago-centric map of the environment, $T_{u_{t-1}}$ is the transformation of the previous map m_{t-1} from the agent motion u_{t-1}, P^{-1} is an inverse projection function of image pixels or pixels features and f_ϕ is a learned function with parameters ϕ. By providing such a map, there's a possible that the performance can be increased since it stores more information about the environment and about its motion. These maps can be precomputed as this example, or it can be learned end-to-end as well. In the neural/latent form, map entries are latent states similar to the state of a recurrent network, but spatially organized. In this subsection, we will focus on showing both possibilities and how different setup of maps played a role in the agent performance.

Maps as memory Multiple works for embodied navigation have leveraged from building maps during an episode during RL training. The maps represent an form of explicit episodic memory. The episodic map can be spatial (Gupta et al. 2017a; Haarnoja et al. 2018) and topological (Chaplot et al. 2020d; Savinov et al. 2018). The spatial maps can be built as egocentric maps (Beeching et al. 2020a; Gupta et al. 2017a) or allocentric maps (Gupta et al. 2017b). Spatial maps (whether egocentric or allocentric) have been used in multiple navigation tasks. For task like exploration, Zhang et al. (2020) trained a deep RL model using prebuilt map with SLAM, while Chaplot et al. (2020b) worked on building a neural mapper for egocentric maps through mapping visual observations (first person view) directly by end-2-end neural module using superised learning. The output of the neural module is then integrated in an allocentric map for training an RL exploration agent. In object-related tasks like *ObjectNav*, Chaplot et al. (2020a) uses differentiable mapping module for semantic segmentation of a first person view and projecting the segmentation on a bird-eye-view map. On the other side, topological maps has been used in (Beeching et al. 2020c; Chaplot et al. 2020d; Kwon et al. 2021; Savinov et al. 2018) for *ImageNav* task where the agent provided the goal as an image (view or specific object), most of the works store landmark nodes that correspond to an input frame and connect the nodes with edges to the closest nodes (under a certain threshold). Every node stores its episodic

position and a unique representative embedding from visual input frame features. The visual embeddings are then compared with the visual goal embeddings (query embeddings). As a similarity function for comparing embeddings, cosine similarity function is a common choice. Once a close node to the goal is localized, the agent navigates directly to the goal using the nodes edges.

Study on Spatial Maps Wani et al. (2020) conducted an interesting and extensive benchmark with a systematic investigation of what beneficial information from spatial maps can boost the performance on navigation tasks. The *MultiON* task was a good candidate to test the memorizaton capacity of agents in a long-horizon task: the navigation agent were tested on 1-ON, 2-ON and 3-ON variants 2.6. Multiple RL recurrent agent (figure 2.13) were trained end-to-end with PPO using the *Habitat* simulator with a budget of 40M training steps. The benchmarked agents fall under three main categories of agents: **NoMap**, **OracleMap** and **LearnedMap**. Each agent differs in the type of map input. (1) **NoMap** refers, explicitly, to an agent that does not use explicit map information. This agent corresponds to a baseline recurrent agent that relies on the memory capacity of recurrent network (GRU). (2) **OracleMap** agents refer to agents that have access ground-truth from the scene, whether objects location (**OracleObjMap** agent), occupancy (**OracleOccMap** agent) or both (**OracleObjOccMap** agent). An egocentric perceptive version of the latter was provided (**OracleEgoMap** agent) where an explored area in the global map is progressively increased as the agent's exploration of the environment increases. (3) The **LearnedMap** agents use maps that are fully learned. Learned maps can be divided into two categories: maps that store neural image features (referred as **ProjNeuralMap** - figure 2.14) and maps that stores the predicted categories of the current visible goals by a classification network **ObjReconMap**.

The benchmark shows multiple interesting findings on maps: (1) The spatial maps are indeed useful for an end-to-end RL agent. Providing ground-truth maps helps significantly **OracleMap** agents over **NoMap** agent, especially, in longer-horizon tasks. Maps are more useful in 2-ON and 3-ON than for 1-ON. (2) Considering the **OracleMap** setup, it's much more useful to provide the object category information in the map than the occupancy information. One possible interpretation behind the usefulness of occupancy information is that the same information is already embedded in the depth sensor or in the additionally provided recurrent state. Also, by just adding the two informations (Object + occupancy) concurrently to the map, the training budget (40M steps) was not enough for a full convergence. (3) Regarding the **LearnedMap** agents, simply providing the recognized goal from visual features and integrating this information directly into a map can be more effective than accumulating images features into the map. (4) Obviously, in egocentric views (EgoMap), **OracleMap** agents are significantly better in performance compared to **LearnedMap** ones.

(5) Independently of the map used, the performance drops dramatically when increasing the number of goals. This observation raises the question of building new architecture of mapping agents that are agnostic and robust to the complexity of the visual reasoning required.

2.3.4 Extensions to End-to-End Training

3D photorealistic simulators open the door for end-to-end agents to be trained on auxiliary tasks, irrespective to the main navigation task. These tasks aim to focus on 3D visual understanding of the targeted environments. They can possibly enhance the final performance of the agent on the desired navigation task. Accessing the ground truth information in simulators helps in supervising these tasks without the need of external manual annotation of data. In this section, we focus on showing different approaches to extend the end-to-end training with extra self-supervised auxiliary losses.

2.3.4.1 Training with privileged Information

Auxiliary tasks Navigation agents can combine auxiliary tasks with the main downstream navigation objective. This combination extends the knowledge of the learning model by extracting useful representations of the scenes (Mirowski et al. 2016; Jaderberg et al. 2016a). In Mirowski et al. (2016), they extended the baseline End-2-End Neural agent (Section 2.1.3.2) with two geometric losses: a depth prediction loss $\mathcal{L}_D$ that aims to learn representations that help in obstacle avoidance and a loop closure loss $\mathcal{L}_{loop}$ from SLAM that supervises if the current location has been previously visited within a local trajectory. In photorealistic environments (Savva et al. 2019), Ye et al. (2020) a set of self-supervised auxiliary tasks have been proposed to significantly improve sample efficiency in learning *PointNav*, which sped up the training of DD-PPO (Wijmans et al. 2019) by 5.5x times. Distributed RL training is combined with three self-supervised auxiliary tasks: (1) the action-conditional contrastive predictive coding task (CPC|A) (Guo et al. 2018): by using a secondary GRU, future observation embeddings are contrasted from other observation embeddings at every timestep. (2) inverse dynamics estimation (Pathak et al. 2017) where the model uses the belief h_t (Section 2.1.3.2) and two egocentric observations embeddings to predict the action between them; and similarly, (3) the temporal distance estimation task where the model predicts the distance between two observations from a trajectory. In *ObjectNav* task, Ye et al. (2021) use the same configurations of (CPC|A) as in Ye et al. (2020). In addition, three auxiliary tasks are introduced. The first two are general-purpose inverse tasks - action distribution prediction (ADP) and general inverse dynamics (GID). Both tasks predict actions taken between two observations apart by k frames. ADP evaluates the KL-divergence between a

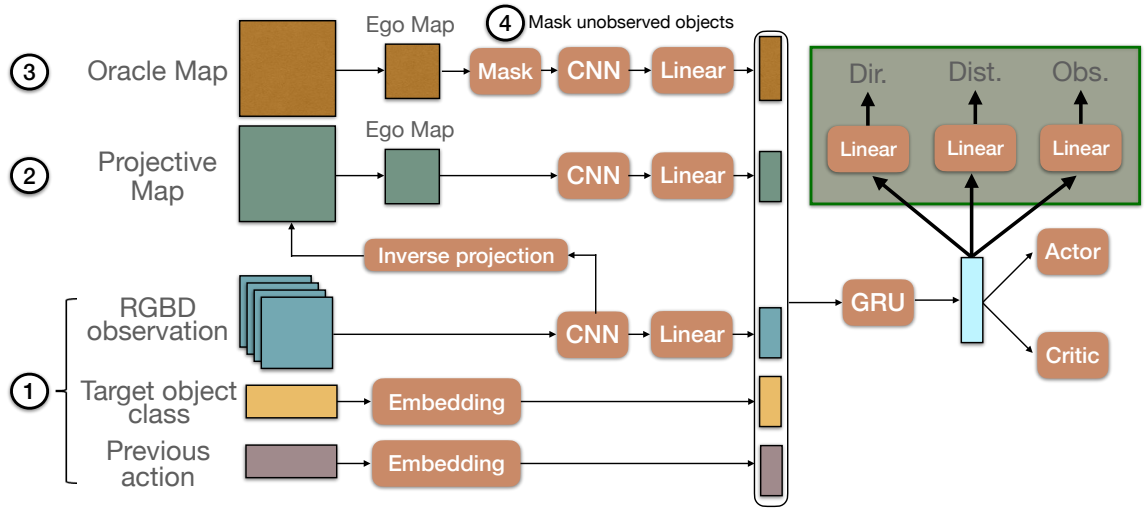


Figure 2.15 – **Learn to map with auxiliary tasks** The figure is reproduced from Marza et al. (2022). During RL training (with PPO), the *MultiON* agents (Figure 2.13) are extended with three auxiliary cross-entropy losses: Direction, Distance, Observed target (shown in the green rectangle on the right).

predicted action distribution and the empirical distribution of the next k actions while GID uses an extra GRU to estimate the individual action between the two observations. The third task is exploration-specific coverage prediction (CP) where the agent predicts the change in coverage at the next k frames using GPS sensor and the number of steps taken by the agent.

Spatial losses For more complex tasks such as *MultiON*, Marza et al. (2022) improved the training of the *MultiON* agents (Figure 2.13) with three auxiliary tasks (Figure 2.15) with the goal to encourage spatial reasoning. The three tasks are represented in the form of three spatial cross-entropy losses. We explain the three losses in detail since we will compare to this work in Chapter 4.

- **Observed target loss $\mathcal{L}_{obs}$:**

$$\mathcal{L}_{obs} = \frac{1}{|\mathcal{D}_k|\,T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T-1} -(\mathbb{1}_t^{\text{obs}} \log p(\hat{obs}_t) + (1 - \mathbb{1}_t^{\text{obs}}) \log(1 - p(\hat{obs}_t))) \tag{2.44}$$

  This binary cross-entropy loss favors learning whether the agent has previously seen the target object ($\mathbb{1}_t^{\text{obs}}$=1) or not ($\mathbb{1}_t^{\text{obs}}$=0) during the current episode. In another word, at time t, whether the target object has been observed within the agent's field of view for at least once or not yet. The model predicts the probability distribution over two classes $\hat{obs}_t$ given the

hidden GRU state $\mathbf{h}_t$ through an Multi-layers perceptron (MLP) (Figure 2.15) as $p(\hat{obs}_t) = f_{obs}(\mathbf{h_t}; \theta_{obs})$ with parameters θ_{obs}.

- **Direction loss $\mathcal{L}_\phi$:**

$$\mathcal{L}_\phi = \frac{1}{|\mathcal{D}_k|\, T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T-1} \left[-\mathbb{1}_t^{\text{obs}} \sum_{c=1}^{K} \phi_{t,c}^* \log p(\hat{\phi}_{t,c}) \right] \tag{2.45}$$

Similarly, the agent predicts the relative direction of the target object, only if it has been observed within its field of view during the episode ($\mathbb{1}_t^{\text{obs}}$=1) using the hidden GRU state $\mathbf{h}_t$ through an MLP (Figure 2.15) as $p(\hat{\phi}_t) = f_\phi(\mathbf{h_t}; \theta_\phi)$ with parameters θ_ϕ. ϕ_t^* denotes the ground-truth direction one-hot vector and the current ground-truth direction towards the goal is computed as,

$$\phi_t = \sphericalangle(\mathbf{o}_t, \mathbf{e}) = -\operatorname{atan2}(\mathbf{o}_{t,x} - \mathbf{e}_x, \mathbf{o}_{t,y} - \mathbf{e}_y) \tag{2.46}$$

where $\mathbf{e} = [\mathbf{e}_x\ \mathbf{e}_y]$ and $\mathbf{o} = [\mathbf{o}_{t,x}\ \mathbf{o}_{t,y}]$ are the coordinates of the agent (fixed coordinate at the center box) and the target object at time t, respectively. The angles are kept in the interval $[0, 2\pi]$ and then discretized into K bins (Figure 2.16 left), giving the angle class.

- **Distance loss $\mathcal{L}_d$:**

$$\mathcal{L}_d = \frac{1}{|\mathcal{D}_k|\, T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T-1} \left[-\mathbb{1}_t^{\text{obs}} \sum_{c=1}^{L} d_{t,c}^* \log p(\hat{d}_{t,c}) \right] \tag{2.47}$$

In addition, the agent predicts the Euclidean distance in the egocentric map between the position of the agent, and the position of target object that was observed during the episode ($\mathbb{1}_t^{\text{obs}}$=1) using the hidden GRU state $\mathbf{h}_t$ through an MLP (Figure 2.15) as $p(\hat{d}_t) = f_d(\mathbf{h_t}; \theta_d)$ with parameters θ_d. d_t^* denotes the ground-truth distance one-hot vector and the current ground-truth Euclidean distance towards the goal is computed as,

$$d_t = ||\mathbf{o}_t - \mathbf{e}||_2. \tag{2.48}$$

Again, distances are discretized into L bins (Figure 2.16 right), giving the distance label.

They three auxiliary losses are added to the main PPO loss (Section 2.1.3.2) as follows,

$$\mathcal{L}_{tot} = \mathcal{L}_{PPO} + \lambda_{obs}\mathcal{L}_{obs} + \lambda_\phi \mathcal{L}_\phi + \lambda_d \mathcal{L}_d \tag{2.49}$$

where the hyperparameters λ_{obs}, λ_ϕ and λ_d weight the relative importance of auxiliary losses. To summarize, each auxiliary task is responsible for increasing a specific sub-skill. The observed target prediction task increases memorization

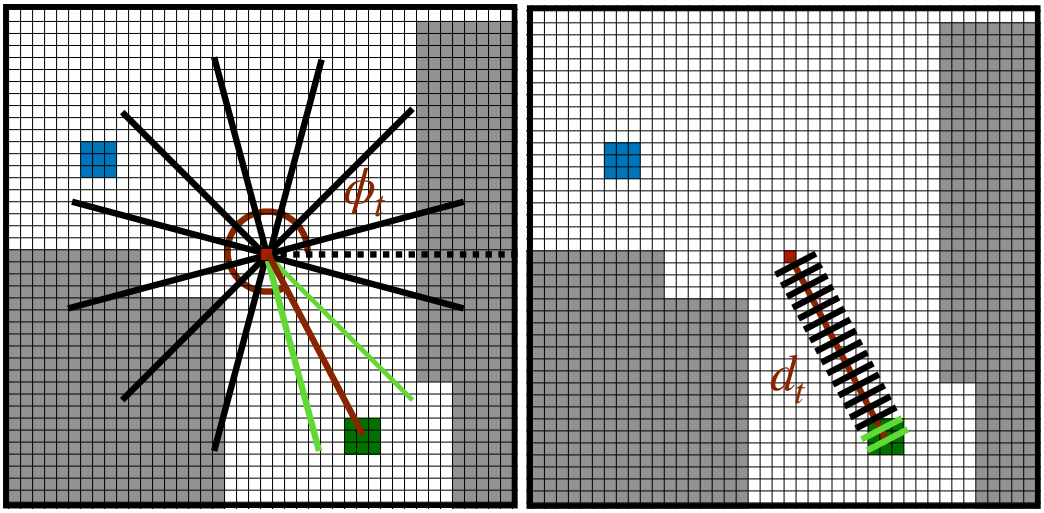


Figure 2.16 – **Auxiliary losses for MultiON agents**: For the *MultiON* navigation problem, two auxiliary tasks are visualized. One task is to predict, the direction ϕ_t (left) and the other task is to predict the distance d_t (right). Both predictions are with respect to the current target object (the green object), if it has been observed within the agent's field of view (green lines) at least once. Both values, ϕ_t and d_t are discretized with respect to the center of the egocentric map (i.e. with respect to the agent position). The figure is reproduced from Marza et al. (2022).

capacity of the agent, the Direction prediction increases the sense of direction, and the Distance prediction increases the sense of judgement of relative distance. By adding the three spatial losses, the performance is boosted for all *MultiON* agents. Even **NoMap** agent trained with the additional losses outperforms the **OracleEgoMap** agent. In that sense, the **ProjNeuralMap** agent trained with the additional losses becomes the state-of-the-art agent on the task.

2.3.4.2 Auxiliary losses to improve perception

Self-Supervised losses (SSL) with data augmentation During training, auxiliary tasks can be self-supervised by accessing privileged information from the simulator or by using contrastive learning with data augmentation techniques. The latter technique has been well studied in 2D computer vision (Chen et al. 2020a; Misra et al. 2019; He et al. 2019) and has been introduced in RL settings (Laskin et al. 2020) by combining both objectives: Contrastive and Reinforcement learning. Hansen et al. (2020a) use self-supervised representation learning to adapt RL policies during deployment only, while Stooke et al. (2020) decouple the RL training from representation learning. Hansen et al. (2020b) jointly train the RL policy with representation learning by applying data augmentation on

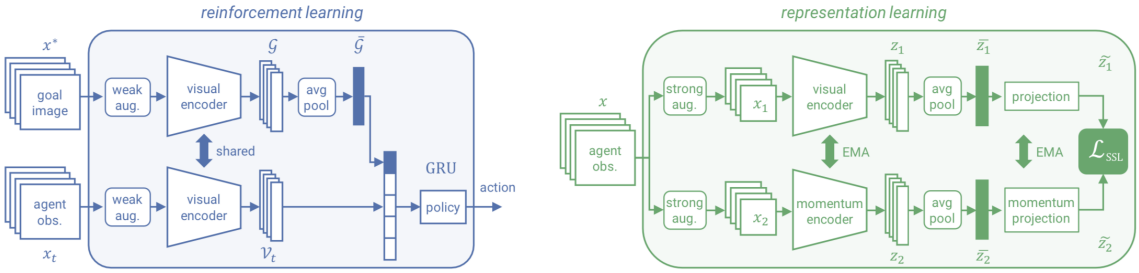


Figure 2.17 – **SSL with data augmentation:** a training framework that combines RL training of *ImageNav* agent using weak data augmentation (left) with SSL contrastive learning, using strong augmentation (right). The agent receives 4 observations x_t of the panoramic view and goal panoramic view x^* to be processed with a shared visual encoder and then fed to an actor-critic policy to estimate the current action. Using the same visual encoder, the panoramic view x is processed, average-pooled and projected to produce $\widetilde{z}_1$. Similarly, the view is processed with the same momentum encoder of (Chen et al. 2020b) (with parameters, updated as an exponential moving average (EMA) of the parameters from the visual encoder). The two models are trained jointly with the two objectives (RL and $\mathcal{L}_{SSL}$). The figure is reproduced from Majumdar et al. (2022).

the representation learning part. Mezghani et al. (2021) apply data augmentation the End-to-End RL policy for *ImageNav* visual navigation with panoramic view. A more recent work (Majumdar et al. 2022) extends (Mezghani et al. 2021) by studying the effect of applying different techniques of data augmentation during RL training to increase the variations seen during training. In addition, they jointly train two losses: the PPO base loss $\mathcal{L}_{PPO}$ (Section 2.1.3.2) and the self-supervised contrastive loss $\mathcal{L}_{SSL}$ from the InfoNCE work (Oord et al. 2018):

$$\mathcal{L}_{SSL} = -\log\left(\frac{\exp(\widetilde{z}_1 \cdot \widetilde{z}_2/\tau)}{\exp(\widetilde{z}_1 \cdot \widetilde{z}_2/\tau) + \sum_z \exp(\widetilde{z}_1 \cdot \widetilde{z}_n/\tau)}\right) \tag{2.50}$$

As shown in figure 2.17, $\widetilde{z}_1$ and $\widetilde{z}_2$ represented the average-pooled and projected features of independent sampled strong augmentations. $\widetilde{z}_n$ are sampled from other panoramas in the training batch and τ is a temperature parameter (Hinton et al. 2015).

During this study, they also conduct some experiments by dropping the dense reward $r_t^{\text{geo-dist}}$ from the reward equation (Section 2.3.2). They argue the dense reward shaping can lead to memorizing training environments as demonstrated in (Maksymets et al. 2021). Thus, it can penalize exploration and hinder generalization to new scenes. This study comes with important findings: (1) Training without dense reward dramatically decreases the performance of baseline training

but when SSL training is added to the sparse-reward-only solution, it outperforms the dense reward variants. (2) Surprisingly, SSL + dense reward shaping does not lead to similar gain if compared to sparse-reward-only solution. (3) Applying strong data augmentation directly within RL training decreases enormously the performance. (4) In previous works (Mezghani et al. 2021), data augmentation is applied independently to each instant of the episode to increase the variations seen during training. However, in the case of episodic data (navigation episode), they observe that using different inconsistent augmentation over time leads to unrealistic changes that do not reflect realistic conditions (e.g. changes in wall painting from one frame to the other). Therefore, by simply applying episodic-aware data augmentation during training, such agents outperforms the inconsistent data augmentation variant.

2.4 Hybrid Modular Agents

In this section, we visit the different approaches used to build a modular agent that can leverage multiple modules. Each module focuses on solving a sub-task in the main navigation task. A modular agent can have only learnable *Neural* modules or a mix of *Neural* and *Analytical* modules, which we call *Hybrid Agent*. During training, *Neural* modules can be learnt separately (Beeching et al. 2020c; Hahn et al. 2021), jointly (Chaplot et al. 2020d), and in end-to-end (Gupta et al. 2017a) by optimizing the same criterion or with the support of (Chaplot et al. 2020b).

We focus on presenting the different techniques used to achieve the reasoning steps (Figure 2.4). First, we present how the environment is perceived and mapped with various modules. Second, we show the decision making modules used by the agent to plan.

2.4.1 Perceive and Map

Occupancy Mapping A common module is to map directly the RGB observation of first-person view to a top-down map. One common way is to predict a 2D-map directly using an Encoder-Decoder (Gupta et al. 2017a; Chaplot et al. 2020b). Gupta et al. (2017a) (known as CMP) propose an ego-centric mapping module that predicts a 2-channel map to represent confidence channel $c_t^{'}$ and belief channel $f_t^{'}$ about the navigability of each grid position of the current ego-centric top-down view of the scene. CMP maintains an egocentric top-down map (EgoMap) of the current state f_t which is the combination of the estimated $f_t^{'}$ and the previous map f^{t-1} using an update function U. First, the current EgoMap is estimated using a function approximator ϕ that takes as input the current ego-centric image I_t. In order to perform the combination steps in Equation 2.51:

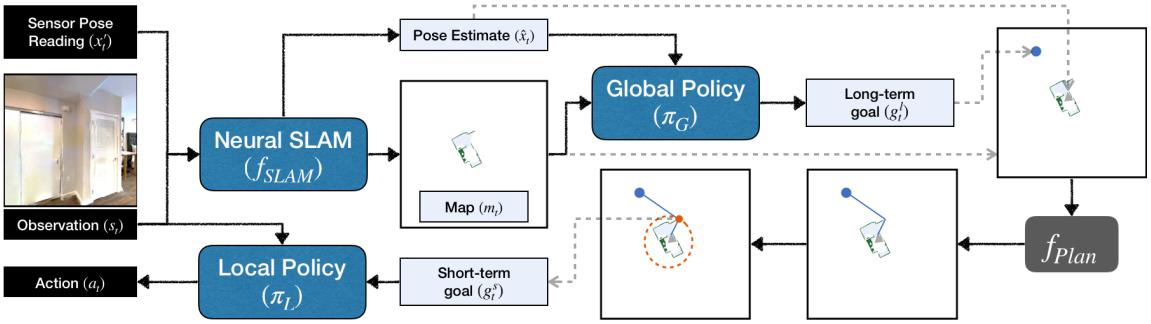


Figure 2.18 – **Active Neural SLAM - Overview:** From the current sensor readings, Neural SLAM f_{SLAM} predicts a map m_t and agent pose x_t. Then, a *Global Policy* π_G uses the map and the pose and predicts a long-term goal g_t^l which is converted to a short-term goal g_t^s by an analytical planner f_{Plan}. A local neural policy π_L (trained for *PointNav* task) is used to navigate to g_t^s.

$$f_t', c_t' = \phi(I_t) \tag{2.51}$$

$$f_{t-1}^e, c_{t-1}^e = W(f_{t-1}, c_{t-1}, e_t) \tag{2.52}$$

$$f_t = U(f_{t-1}^e, f_t') = \frac{f_{t-1}^e c_{t-1}^e - f_t' c_t'}{c_{t-1}^e + c_t'} \tag{2.53}$$

$$c_t = c_{t-1}^e + c_t' \tag{2.54}$$

where the previous EgoMap of $t-1$ should be in the same coordinate as the current one. Therefore, a differentiable module W is used that applies a warping operation. To ensure that this step is differentiable, a bi-linear sampling is applied in W to allow the backpropagation of gradients from f_{t-1} to f_t (Jaderberg et al. 2016b). This sampling technique has been used in optimization of neural networks in the context of view synthesis problems (Zhu et al. 2017). The only learnable function in this module is ϕ. The training is done using DAgger (Section 2.1.3.2), at each step the U function provides the map to a learnable planner based on Value Iteration Networks (Tamar et al. 2017). The planner predicts the next action to take which will be supervised by the optimal action on the corresponding ground-truth trajectory. The mapper part is not supervised explicitly or directly from ground-truth map. It has more freedom to write any information useful for the purpose of navigation, which is learned by directly optimizing the downstream navigation loss.

Following CMP, Chaplot et al. (2020b) extended CMP by proposing a Neural SLAM module,f_{SLAM}, which simultenously acts as mapper and localizer (Figure 2.19) inspired from classical SLAM (Section 2.2). f_{SLAM} is part of their modular agent (Figure 2.18), named Active Neural SLAM (ANS). f_{SLAM} takes the current

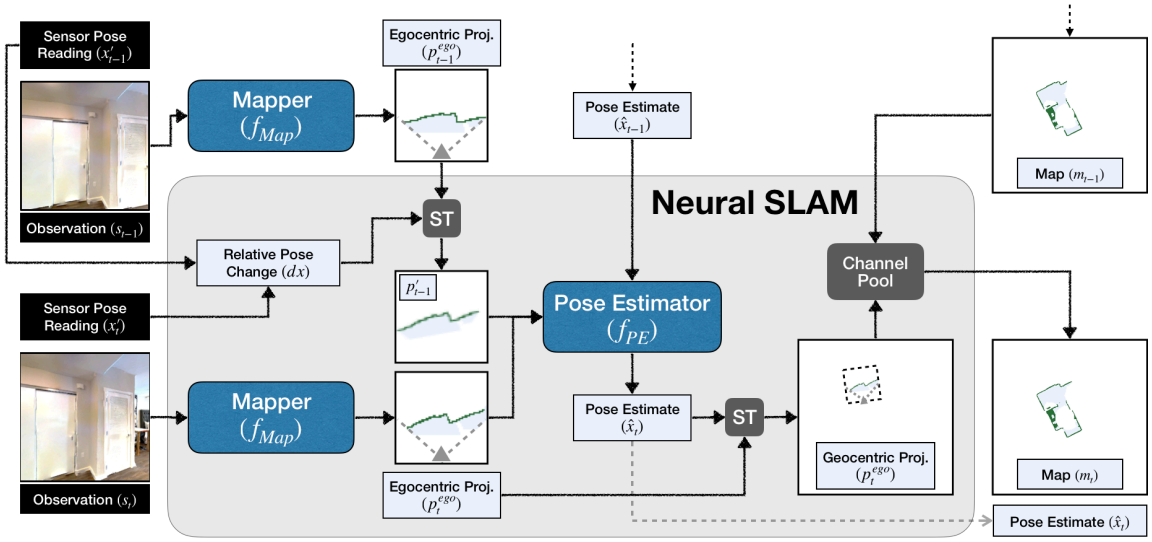


Figure 2.19 – **Neural SLAM Module**: As shown in figure 2.18, f_{SLAM} estimates simulatenously a map m_t and pose x_t. It consists of two trained modules, f_{map} (Mapper) and f_{PE} (Pose Estimator). The "ST" module represents a spatial transformation of the map given a pose change. The figure is reproduced from Chaplot et al. 2020b.

observation s_t, the current noisy pose x'_t, the previous geocentric map m_{t-1} and the estimated pose $\hat{x}_{t-1}$. In return, it outputs an updated spatial map m_t and a new estimate of the pose $\hat{x}_t$. It consists of two learned function approximators, a Mapper f_{Map} and a Pose Estimator f_{PE}. f_{Map}, designed as an Encoder-Decoder network, predicts a Spatial EgoMap p^{ego}_t which presents the explored area and obstacles in the field of view of the agent. Using the pair of prediction and p^{ego}_{t-1} and p^{ego}_t, the Pose Estimator f_{PE} predicts the correct relative pose $\widehat{dx}_t$. The intuition is that the estimator can learn the small transformation that align the two estimated EgoMaps. Then, the new $\hat{x}_t$ is equal to $\hat{x}_{t-1} + \widehat{dx}_t$. Like CMP, ANS uses the same differentiable module W (named ST in figure 2.19) to align maps. On the other hand, it combines previous and current maps using channel-wise addition and not on weighted average as in U (equation 2.51). Moreover, the full f_{SLAM} module is trained using supervised learning independently of the planning module.

Semantic Mapping ANS targets the exploration or *PointNav* tasks and does not perform object searching tasks. f_{SLAM} only maps the explored area and not the detected objects. In a follow-up work (Chaplot et al. 2020a), the mapping is extended to map the semantics in the first-person view (Figure 2.20). They adopted a hybrid approach. First, a pretrained segmentation model (MASK-RCNN is used) predicts a semantic segmentation from the first person RGB view. Second, the depth observation is used to compute a point cloud. The point cloud is associated

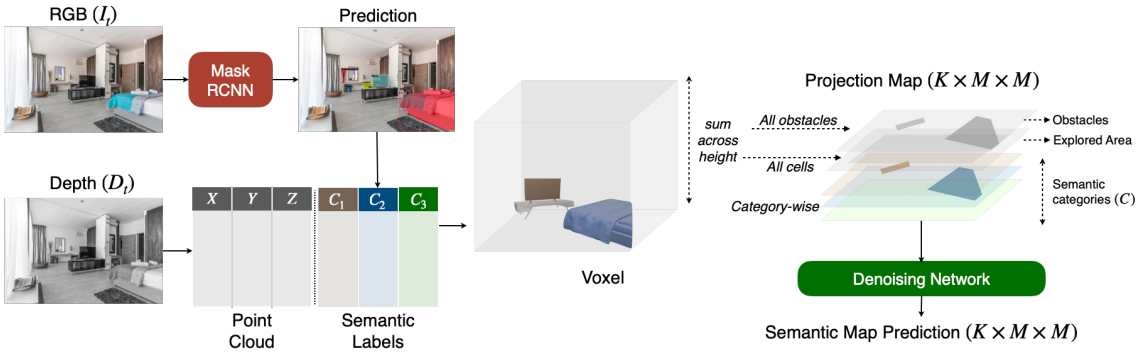


Figure 2.20 – **Semantic Mapping**. A semantic segmentation module (MASK RCNN in this example) takes an RGB observation I_t and predicts semantic mask for which is projected into voxels using the depth readings D_t. The voxels are reduced by summing across the hight to get a semantic top down map, aligned with an occupancy map. The figure is reproduced from Chaplot et al. 2020a.

Table 2.1 – The table is reproduced from (Luo et al. 2022). A description of the 7-channel map maintained by the agent in this work. The first three channels of the map are used to record obstacle-related information, while the rest of the channels are used to record goal-object-related information.

	Channel	Description
Obstacle Map	1	Obstacle Map Using Depth Input
	2	Pessimistic Obstacle Map using Collision Information
	3	Optimistic Obstacle Map using Collision Information
Object Identification Map	4	Total Number of Frames Appeared in View
	5	Sum of Confidence Score of Target Object
	6	Maximum Confidence Score of Target Object
	7	Maximum Confidence Score of Non-Target Object

with the segmentation mask. Using a differentiable geometric projection module, the point cloud is projected in the 3D space to a voxel representation. The semantic map is created by summing over the height dimension of the voxel presentation. The segmentation model can produce some errors in the segmentation which can lead to larger errors in the map after projection. To overcome this limitation, another learned module is added. A Denoising Network receives the projected semantic map and outputs a final denoised version. Therefore, the full semantic module is trained using supervised learning with two losses: a cross entropy loss on the semantic segmentation and a loss in the map space on the semantic map prediction. The ground-truth semantic map is generated from ground-truth semantic annotation in the simulation environment.

The semantic module is adopted in other hybrid agents (Ramakrishnan et al. 2022; Zhai et al. 2022; Luo et al. 2022). However, the final semantic map representation can differ, which can reflect on the decision-making part while planning using the map (Section 2.4.2). In the initial work by (Chaplot et al. 2020a), the final map is presented as an allocentric map with $C + 2$ channels. Each of the C channels represents an object category and the first two channels represent obstacle and explored area. On the other side, Luo et al. (2022) used a different representation in their hybrid agent (named STUBBORN). As shown in table 2.1, they used fixed 7 channels, independent of the number of categories C. The first channel is as in (Chaplot et al. 2020a), it stores the obstacle map from the depth information. The second and third represent any occurrence of "physical" collision that maybe not be represented in the depth observation (due to noises or artifacts in the simulator). The exact collision location cannot be observed, and the agent location is discretized, hence, it's impossible to exactly construct an occupancy map from collisions. They tackle this problem by maintaining two *multi-scale* channels. One channel represents a "pessimistic" map where collided region is marked with larger areas (25x25 cm^2) while an "optimistic" map marks smaller areas (15x15 cm^2). Alongside, they applied another rule to always mark areas along visited path as free. These manipulations reduce the chance that the agent trapped due to incorrectly marked obstacles.

2.4.2 Planning

Hierarchical Planning Many hybrid modular agents use a hierarchical planning approach for navigation. A *Global Policy* (High-Level Planner) proposes a waypoint. Then, a *Local Policy* (Low-Level Planner) performs point-to-point navigation to reach the waypoint. The *Global Policy* acts at a coarse timescale while the *Local Policy* acts a a fine time scale. At each time step, the Low-Level policy should replan to reach the waypoint. Global Policies can be learned (Chaplot et al. 2020b; Ramakrishnan et al. 2022; Zhai et al. 2022) or be analytical such as the classical algorithm of Frontier-Based Exploration (Yamauchi 1997).

In ANS (Chaplot et al. 2020b), the *Global Policy* is presented as a convolutional neural network that regresses the waypoint g_t^l (Figure 2.18). The policy takes as input the estimated allocentric map from the f_{SLAM} in addition to two extra channels concatenated on the map m_t. The third channel represents the current agent position and the fourth channel represented all visited locations until step t. In order to reach the g_t^l, ANS relies on two-staged Low-Level Planner: an analytical planner (The Fast Marching Method - FFM from Sethian (1996b)) that takes the goal g_t^l with the spatial map and plans the shortest-path. It proposes a shorter goal g_t^l on the planned path. Then an End-to-End Neural policy represented as visual-based Policy as in Savva et al. (2019), $a_t = \pi_L(s_t, g_t^s)$. Experimentally, they show that using the two policies doesn't add a significant gain to the global

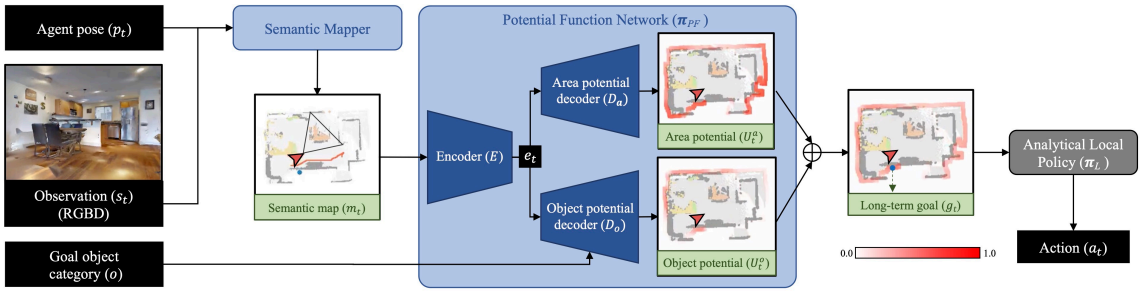


Figure 2.21 – **Potential Functions for navigation**. The figure of the architecture for PONI is reproduced from Ramakrishnan et al. (2022). It consists of a semantic mapper (similar to figure 2.20 version). The produced map is used by a potential function π_{PF}. First, the map is encoded into a vector e_t and then used with the current object goal category o_t to predict the area and object potential functions. The two potentials are averaged. A long-term goal g_t is sampled by taking the maximum location on the final potential area. The agent navigates toward g_t using a analytical path-planning *Local Policy* π_L.

performance of the modular agent. In future works (Chaplot et al. 2020a), FFM is used as a stand-alone Low Policy. The *Global Policy* is trained using RL to optimize the coverage of the scene (equation 2.40).

In *ObjectNav* task, a semantic Goal-Oriented policies (Chaplot et al. 2020a; Zhai et al. 2022; Ramakrishnan et al. 2022) are implemented similarly where an additional input is provided in order to represent the current target category. Intuitively, once the corresponding channel to the target has a non-zero element (which means the goal is segmented and mapped), it chooses the location as the current waypoint target for the *Local Policy*. If the goal is not observed, the policy needs to propose a long-term goal where the current target object is likely to be in the area. Therefore, the semantic-aware *Global Policy* differs from the Goal-Agnostic *Global Policy* of ANS on its target to learn semantic priors on the correlation and arrangement of objects and areas. One way of training is to use RL to optimize the geodesic distance reward to the nearest goal object (equation 2.39).

Potential regions Another approach to build a *Global Policy* is to train an Encoder-Decoder network that takes an incomplete map and generates proposal regions (areas) that are potentials for area exploration or goal finding. Ramakrishnan et al. (2022) build a hybrid agent (named PONI - Figure 2.21) with two U-shaped networks that share the same semantic map encoder (E) and predict two types of potential regions: the area potential U_t^a and the object potential U_t^o. Both regions are linearly combined and then a potential way-point is sampled. They illustrate that area potential is useful and has higher values at the beginning

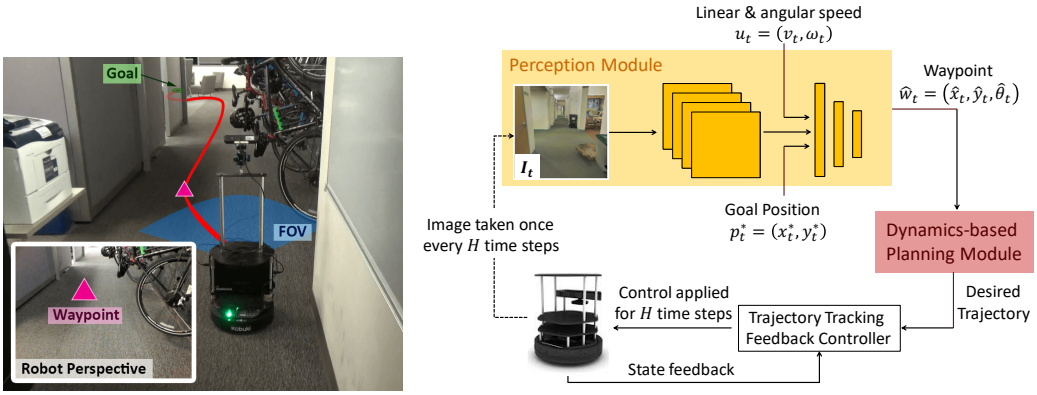


Figure 2.22 – **Optimal Control and Learning:** The approach consists of a perception module trained to predict waypoint $\hat{w}_t$ using the first-person RGB observation. A dynamic-based planning module uses $\hat{w}_t$ to control smoothly and regulates the robot until it reaches the waypoint. The figure is reproduced from Bansal et al. (2019).

of the episode, while the object potential starts to have more importance when the agent discovers more semantics in the scene. Thus, it is more able to estimate the potential regions of the target object. A follow-up work (Zhai et al. 2022) simplified the same potential regions *Global Policy* in their agent (named PEANUT) to train one single decoder for potential regions. This simple version surprisingly outperforms the PONI agent.

Combining with Control Optimization Most previous works abstract out the dynamics and work with set of discrete actions. Such neglection of dynamics produces a stop-and-go jerky behavior when deployed on a real robot. Therefore, another line of research builds hybrid planners that combine learned components with optimal control which is studied in multiple scenarios. In high-speed racing, Drews et al. (2017) introduce a vision-based model predictive control (MPC) for the task of aggressive autonomous driving. The goal is to learn cost functions (cost maps) from monocular video frames using convolutional neural networks. The cost functions are then used for online trajectory optimizations with MPC. In follow-up work (Drews et al. 2018), the hybrid approach is extended to combine learning-based road detections and particle filters (Section 2.2) with MPC. The neural network predicts local cost maps of the track in front of the vehicle. Particle filters are then used for state estimation while driving using MPC. In drone racing, Kaufmann et al. (2018) and Kaufmann et al. (2019) designed a hybrid drone agent that combines a learned global planner with a path-planning and control system. A Neural network directly estimates a waypoint and desired speed from raw images. This information is then used by the planner to generate a short trajectory (Mueller et al. 2013) that is tracked by a low-level controller (Faessler et al. 2015).

For indoor navigation, Bansal et al. (2019) (Figure 2.22) rely on building two submodules: perception and planning. The perception module acts as a *Global Policy* that predicts a waypoint based on the current First-Person-View observation. A planner with a feedback controller acts as a *Local Policy*. The process is shown in equations 2.55 to 2.58.

$$p_t^* = (x_t^*, y_t^*) \tag{2.55}$$
$$\hat{w}_t = \psi(I_t, u_t, p_t^*) \tag{2.56}$$
$$\{z^*, u^*\}_{t:t+H} = FitSpline(\hat{w}_t, u_t) \tag{2.57}$$
$$\{k, K\}_{t:t+H} = LQR(z^*_{t:t+H}, u^*_{t:t+H}) \tag{2.58}$$

At every H steps, the waypoint is predicted using the current position p_t^*, speed u_t and image I_t. Then, a trajectory spline-based planner ($FitSpline$) provides a desired state and control trajectories on a horizon of H steps. In order to track the generated trajectory, a Linear Feedback Controller (LQR) (Bender et al. 1987) is put in place to generate control commands. The control commands are executed over the horizon and then a new image is observed, and the process is repeated until the robot reaches the target position.

Similar approaches have been used in autonomous driving. Müller et al. (2018) designed a hybrid modular approach with end-to-end trained components. First, a segmentation model is used to generate segmentation from raw images. Another learned component is used to generate waypoints from segmentations. A PID controller used the waypoint to generate controll commands. Although these approaches, that rely on First-Person-View Global policies (ANS - Figure 2.18 or PONI - Figure 2.21), proved their capacity in their experimental setups, they are limited to proposing waypoints that are in the Field-Of-View of the agent. This lacks exploration capacity if deployed in such a setup. On the other side, Map-Based Global Policies can propose way-points out of the Field-of-View and potentially in an explored area.

2.5 Conclusion

In this section, we have visited the different concepts needed to understand the navigation problem and the different approaches used to tackle the problem. In the upcoming sections, we focus first on studying the end-to-end learning-based agents in real environments and to interpret the sensor usages by such agents in *PointNav*. Second, we present a new hybrid agent for complex tasks such as *MultiON*, also, in real environments. Third, we try to leverage both decision-making approaches, analytical and learned-based ones to build a new hybrid agent for *PointNav* in real environments.

CHAPTER 3

GENERALIZATION AND VISUAL REASONING OF ROBOTS NAVIGATING IN REAL ENVIRONMENTS

Contents

Chapter abstract

Visual navigation by mobile robots is classically tackled through SLAM plus optimal planning, and more recently through end-to-end training of policies implemented as deep networks. While the former are often limited to waypoint planning, but have proven their efficiency even on real physical environments, the latter solutions are most frequently employed in simulation, but have been shown to be able learn more complex visual reasoning, involving complex semantic regularities. Navigation by real robots in physical environments is still an open problem. End-to-end training approaches have been thoroughly tested in simulation only, with experiments involving real robots being restricted to rare performance evaluations in simplified laboratory conditions.

In this work we present an in-depth study of the performance and reasoning capacities of real physical agents, *trained in simulation and deployed to two different physical environments. Beyond benchmarking, we provide insights into the generalization capabilities of different agents training in different conditions. We visualize sensor usage and the importance of the different types of signals. We show, that for the PointGoal task, an agent pre-trained on wide variety of tasks and fine-tuned on a simulated version of the target environment can reach competitive performance* without modelling any sim2real transfer, *i.e. by deploying the trained agent directly from simulation to a real physical robot.*

The work in this chapter has led to the publication of a conference paper:

- Assem Sadek, Guillaume Bono, Boris Chidlovskii, and Christian Wolf (2022b). "An in-depth experimental study of sensor usage and visual reasoning of robots navigating in real environments". In: *ICRA 2022*;

3.1 Introduction

The design of mobile robots capable of performing visual navigation tasks in real physical environments has been classically addressed with geometric pipelines creating maps from LIght Detection And Ranging (LIDAR) or visual input, localizing themselves on these maps (often simultaneously — SLAM), and using symbolic planners to navigate to a specified target position. In recent years, machine learning has had a deep impact on these problems either by extending the classical pipelines with semantic information of the scene, complementary to geometry, or through end-to-end learning of navigation policies directly mapping observations to actions and usually trained in simulation by Reinforcement Learning (RL), Inverse RL, Imitation Learning, or a combination of objectives.

While classical tasks focus on waypoint navigation (PointGoal), which essentially requires the estimation of free navigational space and the computation of shortest paths, large-scale training in simulation has opened the door to more complex problems, which require more advanced visual and spatial reasoning. Examples are ObjectNav (Batra et al. 2020), which requires finding and recognizing objects, whereas Multi-ON (Wani et al. 2020) and the K-item scenario (Beeching et al. 2020b; Beeching et al. 2020a) require mapping objects in some form of latent memory, allowing the agent to find them quickly when needed. This sub-field, Embodied Computer Vision, is heavily dominated by training *and evaluation* in simulated photo-realistic 3D environments like Habitat (Savva et al. 2019), AI-Thor (Kolve et al. 2017) etc. The transfer from simulation to real environments ("*sim2real*") is one of the main current challenges in robot learning and as such widely studied. However, thorough evaluations of navigation capabilities in real environments are rare, compared to other robotic tasks like grasping and object manipulations. Experiments with mobile robots are time-consuming, as they need to be monitored by human operators and the reproduction of identical or similar evaluation conditions is difficult.

In this work we describe an in-depth study evaluating the performance of agents trained in simulation on two different physical environments. We chose the PointGoal task with GPS pointer estimated from a SLAM algorithm, which allows to compare the trained agents to classical algorithms based on SLAM and optimal planning. In our scenario, absolutely no adaptation is done for sim2real transfer: the agents are trained in simulation, and evaluated on real robots strictly as they are, delegating the lower level continuous controls to preserve the discrete simulated action-space. Compared to existing reports, e.g. (Kadian et al. 2020), we

Figure 3.1 – We present a deep experimental study of navigation capabilities of mobile robots in two different real physical (top) indoor environments: “NLE” (= "Naver Labs Europe"), a French 19th century furnished castle (left) and “INSAL”(= "INSA de Lyon"), modern office spaces (right). The agents have been trained in different sets of simulated environments, which may contain, or not, 3D-scans of the evaluation environments (bottom: 2D observations from the simulator), targeting the evaluation of different generalization scenarios.

did not produce common laboratory conditions — our agents had to fight with challenging settings like sensor failures due to glass fronts and mirrors, wheels sliding on slippery floors or bumping into thick carpets.

Our main objectives were the study of internal visual reasoning learned and performed by the agent for this task, which requires to attain a target position given a (noisy) GPS signal and RGB-D observations. The agent can not blindly follow the direction indicated by the GPS signal and needs to use the RGB-D observations to estimate the geodesic path, which can significantly differ from the Euclidean path because of walls, narrow corridors and other obstacles.

Classical planning vs. ML while we also evaluate performance of classical planners on this task for comparison, the goal of our work is *not* to claim superiority of end-to-end training over the classics. We currently know that the PointGoal task can be addressed, "solved" with sufficient robustness by using symbolic planners, but they have a hard time to scale up to complex visual reasoning. The main question we explore here is whether end-to-end learning can work in general. In cases where end-to-end training is required, do we need to combine ML methods with classical low level planning when we target real physical environments, or can end-to-end training perform in its own right? In what follows we will show that low-level waypoint navigation in physical environments can indeed be addressed by a trained planner.

In our study, we try to answer the following questions:

- Is PointGoal navigation in real environments possible without any adaptation from simulated training environments?
- How are the different sensors used? When does the agent use visual observations and how? What are the regions attended to by the agent and in which situation?
- How do the trained agents generalize to unseen conditions? Is successful navigation conditioned on memorization on a trajectory level, or do the agents learn some form of spatial reasoning transferable from simulation to the real environment?

After a description of the related work (Section 3.2 on robot navigation and sim2real transfer), the following sections will describe the experimental setup (environments, agents, and evaluation protocol, Section 3.3), followed by a detailed analysis of the performance of the agents in real and simulated environments and visualizations of their sensor usage (Section 3.4).

3.2 Related work

Simulators and indoor navigation. Research on robot-inspired agents that perceive, navigate, and interact with their environment is currently carried out in simulation rather than in real physical environments, at least when based on machine learning (Savva et al. 2019; Xia et al. 2018; Kolve et al. 2017). Simulators can run experiments much faster than real-time due to high parallelization, and can deliver *decades* of simulated agent experience to in only *days* of wall-clock time (Gupta et al. 2017a). Moreover, evaluating agents in simulation is safer and cheaper and allows for easier automatic benchmarking of new techniques compared to handling physical robots in the real-world (Choi et al. 2020).

Simulators serve as a testbed for developing increasingly realistic indoor environments (Savva et al. 2019; Kolve et al. 2017; Shen et al. 2020) and navigational tasks (Gupta et al. 2017a; Anderson et al. 2018b), as well as running navigation challenges (*Habitat Challenge* 2020; *Embodied AI Workshop, CVPR 2021 Challenge* 2021). Massive synthetic data in simulators enables learning perception and control policies end-to-end (Gupta et al. 2017a).

Simulation-to-real gap. No simulation can perfectly replicate reality. Despite the tremendous progress in computer graphics and game engine technology, highly used in robot learning, the sim2real gap can compromise any strong performance achieved in simulation when agents are tested in the real-world.

Perception and control policies learned in simulation often do not generalize well to real robots due to inaccuracies in modelling, simplifications and biases. To close the gap, domain randomization methods (Tan et al. 2018; Peng et al. 2018) assume that the real world distribution is a randomized instance of the simulation environment; they treat the discrepancy between the domains as variability in the simulation parameters.

Alternatively, domain adaptation methods learn an invariant mapping function for matching distributions between the simulator and the robot environment. Related examples to the work presented here include sim2real transfer of visuo-motor policies for goal-directed reaching movement by adversarial learning (Zhang et al. 2019), adapting dynamics in reinforcement learning (Eysenbach et al. 2021), and adapting object recognition model in new domains (Zhu et al. 2019). Kadian *et al.* (Kadian et al. 2020) investigated the sim2real predictivity of Habitat-Sim (Savva et al. 2019) for PointGoal navigation and proposed a new metric to quantify predictivity, called Sim-vs-Real Correlation Coefficient (SRCC). Recently, Chattopadhyay *et al.* (Chattopadhyay et al. 2021) benchmarked the robustness of embodied agents to visual and dynamics corruptions, finding that agents trained in simulation severely under-perform when evaluated in corrupted target environments. They show that although standard data-augmentation techniques and self-supervised adaptation strategies offer some improvement, much remains to be done in terms of fully recovering lost performance. In (Truong et al. 2021b), a bi-directional method adapts the sensor gap from real2sim and the dynamics gap from sim2real.

Interpretability in RL and Computer Vision. Understanding and interpreting deep neural networks is an ongoing effort and research domain, and work has targeted different applications and tasks, including image classification (Zeiler et al. 2013; Selvaraju et al. 2020), but also gathering tasks in Video games simulators (Jaunet et al. 2020). The latter is close to robotics, sharing navigation components and even the general meta-architecture of the agent with a recurrent memory. The authors developed a data visualization interface allowing to inspect

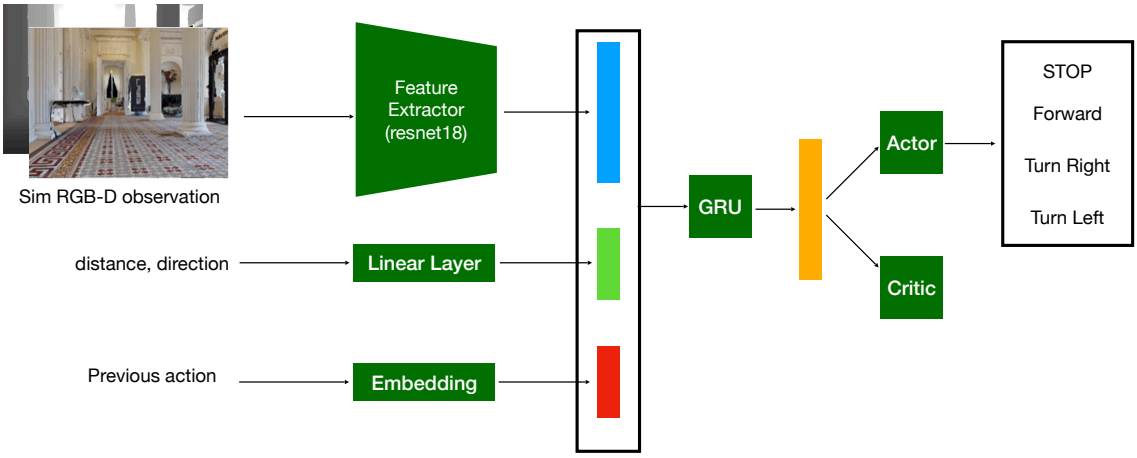


Figure 3.2 – The architecture of the baseline RL-agent trained with PPO, taking as input visual observations, a noisy GPS signal, and the previous action.

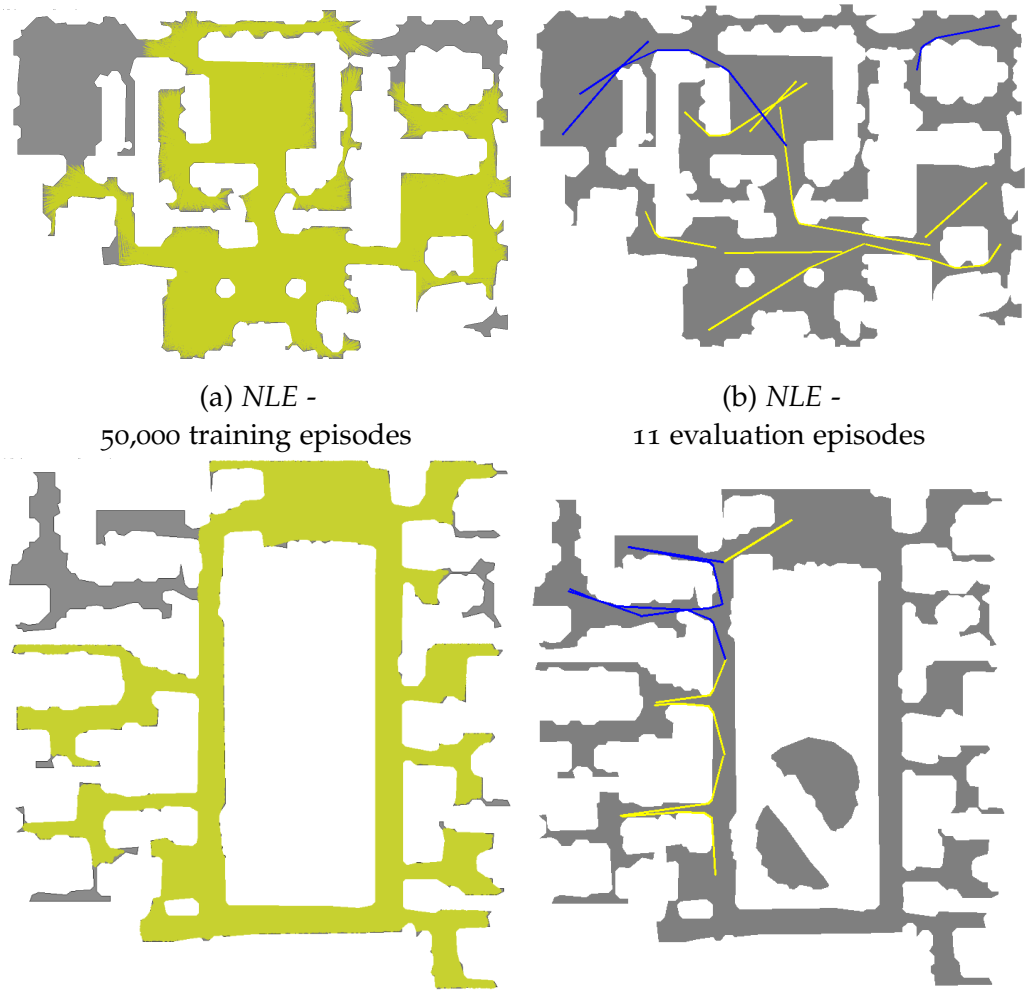
the usage of sensors and recurrent memory, helping to understand the agent's decision making process.

3.3 Experimental setup

In our experiments we train a neural policy, which is able to act in two different environments:

- a virtual environment through the *Habitat* simulator (Savva et al. 2019) with discrete actions (`MOVE_FORWARD`, `TURN_LEFT`, `TURN_RIGHT`) and `STOP`), and
- a real physical robot, a *LoCoBot* (*LoCoBot: An Open Source Low Cost Robot* 2017) [ ].

All training is done in the simulated environment only, we deploy the trained agent directly to the physical robot *without any adaptation in perception*. The translation of discrete actions to the continuous actions of the *LoCoBot* is performed by the standard `move_base` navigation stack of `ROS`, which also handles the navigation towards the starting position of each episode, and the evaluation of geodesic distances. To this end, it builds on a position estimation provided by the default `ROS` implementation of the Adaptive Monte-Carlo Localization algorithm (Thrun et al. 2005). Instead of creating a map dynamically with SLAM, we export a map from the *Habitat* simulator to share the coordinate system, which allows us to execute the same episode in the simulator and in the real world, thus ensuring a fair comparison.



(a) *NLE* - 50,000 training episodes

(b) *NLE* - 11 evaluation episodes

(c) *INSAL* - 50,000 training episodes

(d) *INSAL* - 10 evaluation episodes

Figure 3.3 – Overview of the 50,000 training episodes used to train the *Targeted* and *Finetuned* agents for both the *NLE* (3.3a) and the *INSAL* (3.3c) scenes. Parts of the scenes (appearing in gray) were kept out of the training set, in order to have "seen" (yellow) and "unseen" (blue) episodes in our evaluation sets (3.3b, 3.3d). Note that for the *Gibson* agent, **all** episodes are unseen.

The neural RL-agent policy architecture is based on a ResNet module for perception and recurrent GRU memory, as shown in Fig. 3.2. This agent, trained with the PPO algorithm, is the base implementation provided in the *Habitat* simulator (Savva et al. 2019). At each step, the agent receives a 160×120 pixels RGB-D observation, matching the extrinsic and intrinsic parameters of the *Intel RealSense* camera installed on the *LoCoBot*, in particular, its position, the field of view and aspect ratio. It also receives a GPS vector (Euclidean distance and

direction), describing the position of the goal relative to its current position, provided by the robot's position estimation system.

Generalization over … what? the PointGoal task requires an agent to navigation from a start position to a target position, receiving at each instant a GPS direction pointing towards the goal; in the real environment this direction is estimated by the positioning system and is often noisy. In general, naively following the GPS direction does not solve the task, as the presence of walls and obstacles requires the agent to avoid them and to find the shortest *geodesic* path, using the RGB-D input in addition to the GPS.

We evaluate two different generalization scenarios, *both* requiring to generalize from simulation to the real world:

① **Generalization to unseen environments** — an agent is trained on a set of environments and at test time needs to navigate in an unknown environment. This corresponds to a situation where a robot is operating "out of the box", with no possibility to retrain.

② **Generalization to unseen trajectories** — an agent is trained on a set of environments and at test time needs to navigate in a physical environment it has been trained on, i.e. in its simulated variant. It will not necessarily navigate the same trajectories, but eventually similar ones — we push this further by excluding regions from the environment when training ("unseen").

This use case is commercially feasible, but requires more effort at deployment, as the agent needs to be retrained on a virtual environment created from a 3D scan of the target building.

We performed experiments in two different physical environments: the *NLE* building, a French classical castle and *INSAL*, modern office space (Fig. 3.1). Both buildings were 3D scanned with a professional Matterport system (*Matterport* 2020) and loaded into the *Habitat* simulator, resulting in two novel simulated environments suitable for training navigation agents. We then generated 50,000 training episodes for both scenes, controlling for the distribution of episode difficulties by constraining the geodesic distances between start and goal and the ratio of Euclidean to geodesic distances. In both scenes, we isolate an "unseen" region, which is excluded from the training set and reserved for evaluation (cf. Fig. 3.3). This split between "seen" and "unseen" regions is meaningless in the case when the whole test environment is excluded from training (case ① above), since then all evaluation episodes are unseen.

Eleven evaluation episodes in *NLE* and ten episodes in *INSAL* cross both "seen" and "unseen" regions of the scenes. It is important to note that both *INSAL* and *NLE* scenes underwent changes between the digitization and the agents evaluation phase. Therefore, even the "seen" parts of the scenes contain some discrepancies

between simulation and real, such as moved pieces of furniture or absent/new objects. This contributes to the *sim2real* gap, alongside the rendering difference between real sensors and simulated ones.

Additionally, for certain agents we also use the standard *Gibson* dataset (Xia et al. 2018) for pre-training, which contains 3,600,000 episodes over 72 different scenes.

We deploy three variants of the ResNet+GRU agent:

- The *Gibson* agent, trained solely on *Gibson* scenes (generalization case ①).
- The *Targeted* agents (one per scene), trained on the seen region of their respective scenes (generalization case ②).
- The *Finetuned* agents (one per scene), first pre-trained on the *Gibson* dataset, then fine-tuned to their respective scenes (generalization case ②).

As a baseline, we also provide an evaluation of the standard `ROS` navigation stack using the same metrics.

Evaluation metrics We used three standard metrics from the field:

- *Success Rate* (SR) is the percentage of the episodes where the agent called `STOP` with the final distance d_k to the goal being lower than a predefined threshold d_{success}:

$$SR = \frac{1}{N}\sum_{k=1}^{N} \mathbb{1}_{d_k \leq d_{\text{success}}} \tag{3.1}$$

- *Success weighted by Path Length* (SPL) is the SR with each success weighted by the ratio between the optimal path length l_k^* and the distance travelled by the agent l_k:

$$SPL = \frac{1}{N}\sum_{k=1}^{N} \mathbb{1}_{d_k \leq d_{\text{success}}} \frac{l_k^*}{\max\{l_k, l_k^*\}} \tag{3.2}$$

- *Soft SPL* (sSPL) is a softer version of SPL where the Boolean success value is replaced by a continuous measure of the progress made towards the goal:

$$sSPL = \frac{1}{N}\sum_{k=1}^{N} \max\left\{0, 1 - \frac{d_k}{l_k^*}\right\} \frac{l_k^*}{\max\{l_k, l_k^*\}}. \tag{3.3}$$

Implementation We developed our own interface, publicly available online [1], which makes the real robot and its entire `ROS` stack appear as a "Simulator" in the `habitat-lab` framework. This allows to transparently switch from a simulation evaluation to real world one by changing a single line in the configuration file. This

1. https://github.com/wgw101/habitat_sim2real

Scene		Agent	Seen SR	Seen SPL	Seen sSPL	Unseen SR	Unseen SPL	Unseen sSPL	Overall SR	Overall SPL	Overall sSPL
NLE	sim	*Gibson*	*75.0*	*60.3*	*65.1*	66.7	53.2	52.5	72.7	58.3	61.7
		Targeted	**100**	**94.4**	**93.2**	**100**	72.9	72.2	**100**	88.6	87.5
		Finetuned	**100**	93.8	92.7	**100**	**83.7**	**82.8**	**100**	**91.1**	**90.0**
	real	*Gibson*	*100*	*73.0*	*71.1*	**100**	**85.0**	**84.3**	**100**	76.2	74.7
		Targeted	87.5	61.9	62.5	33.3	22.7	26.8	72.7	51.2	52.7
		Finetuned	**100**	**88.7**	**86.3**	66.7	55.6	54.6	90.9	**79.7**	**77.6**
INSAL	sim	*Gibson*	*100*	*89.4*	*87.9*	**100**	78.4	76.5	**100**	83.9	82.2
		Targeted	**100**	91.3	88.8	40.0	26.0	26.0	70.0	58.6	67.0
		Finetuned	**100**	**93.6**	**92.2**	**100**	**94.5**	**92.2**	**100**	**94.1**	**92.2**
	real	*Gibson*	*20.0*	*16.9*	*16.9*	80.0	46.4	45.4	50.0	31.7	31.2
		Targeted	80.0	43.6	42.5	80.0	50.6	49.4	80.0	47.1	45.9
		Finetuned	**100**	**92.8**	**90.6**	**100**	**91.6**	**89.1**	**100**	**92.2**	**89.8**

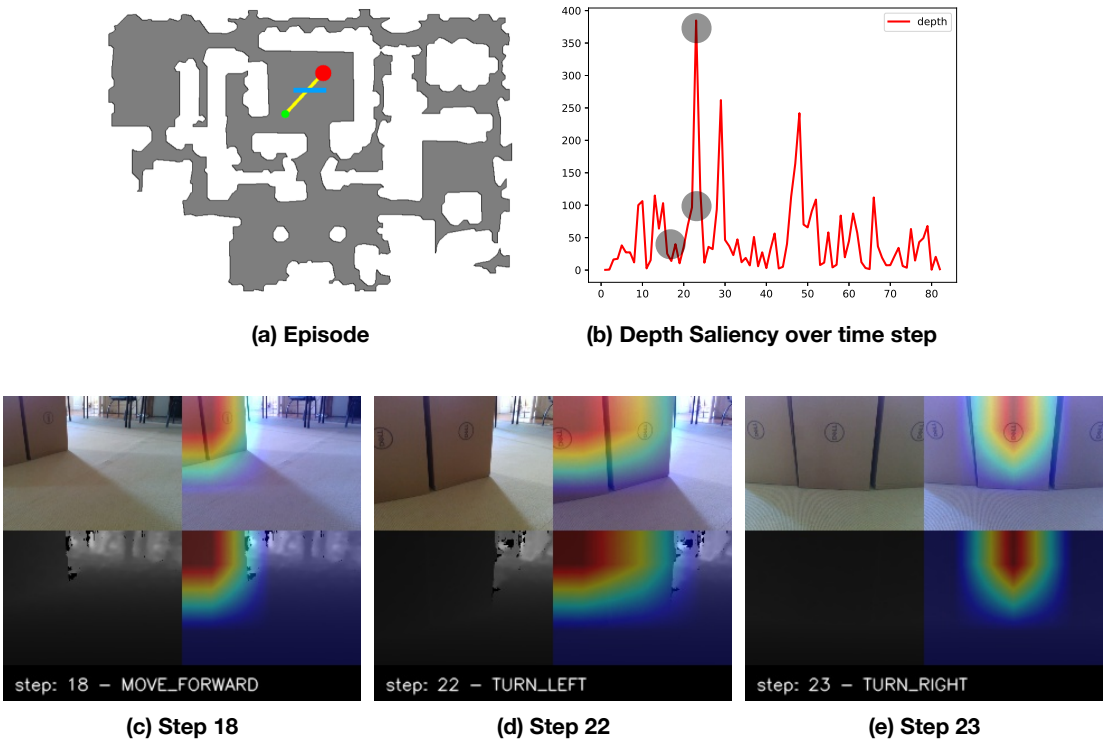
Table 3.1 – Quantitative results for experiments in simulation and real physical robots. We report the average SR, SPL and sSPL on seen, unseen and all episodes. **Bold** font highlights best values on the corresponding set of episodes. *Gray* font denotes that the *Gibson* agent has not really seen any part of the scenes during training.

allows us to re-use all the metrics and features already present in `habitat-lab` for the *PointNav* task, and to deploy the same agent in simulation and on the *LoCoBot*. We rely on `ROS` to provide the position estimation we need, which affects the GPS vector given to the agent, but more importantly the precision of our evaluation metrics.

3.4 Experimental Results

Quantitative performance analysis Table 3.1 shows the agent performance on real robot and in simulation, in both environments. We observe a performance drop between simulation and real, but also that this gap is quite low for the Finetuned agent. We conjecture that pre-training on a large number of additional simulated environments drastically reduces the sim2real gap.

Complete generalization to unseen environments (generalization case ①) is possible but not universal, as shows the poor performance of the Gibson agent in the *INSAL* environment. Deployment of a robot to real environment after pre-training on a 3D scanned variant (case ②), however, leads to surprisingly good



(a) Episode (b) Depth Saliency over time step

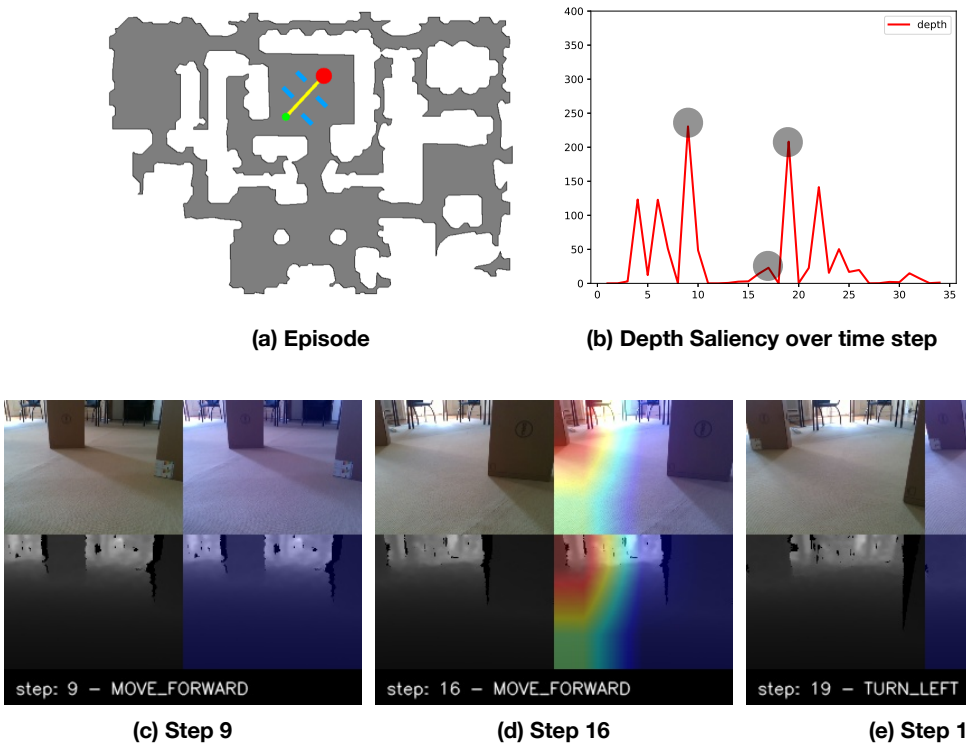
(c) Step 18 (d) Step 22 (e) Step 23

Figure 3.4 – Impact of obstacles on visual attention: obstacles (boxes) are close and on the path to target, the robot pays the main attention to them. Black dots on the saliency curve indicate the steps shown.

performance. For the Targeted and Finetuned agents (case ②), the performance gap between seen and unseen episodes is low — training on an environment benefits performance even on the unseen parts, which, unsurprisingly, suggests the existence of factors of variation common to the trajectories of a given environment (and thus particular to the environment).

Visualization and interpretation We have explored different ways to increase interpretability of the neural model and to visualize parts of its sensor usage. We ask the following questions: (a) what type of sensor is important at what time in an episode (RGB-D or GPS), (b) which region is attended to in an observed image, and ultimately, (c) why does an agent take a certain action at a certain moment?

We use the visualization procedure grad-CAM (Selvaraju et al. 2020), which computes a measure of attention or importance, by, written in simplified terms, calculating the gradients of the output layer of the policy corresponding to the taken action w.r.t. to the inputs or an intermediate neural unit. A high derivative for a pixel, value or unit value suggests a high impact on the decision. We chose



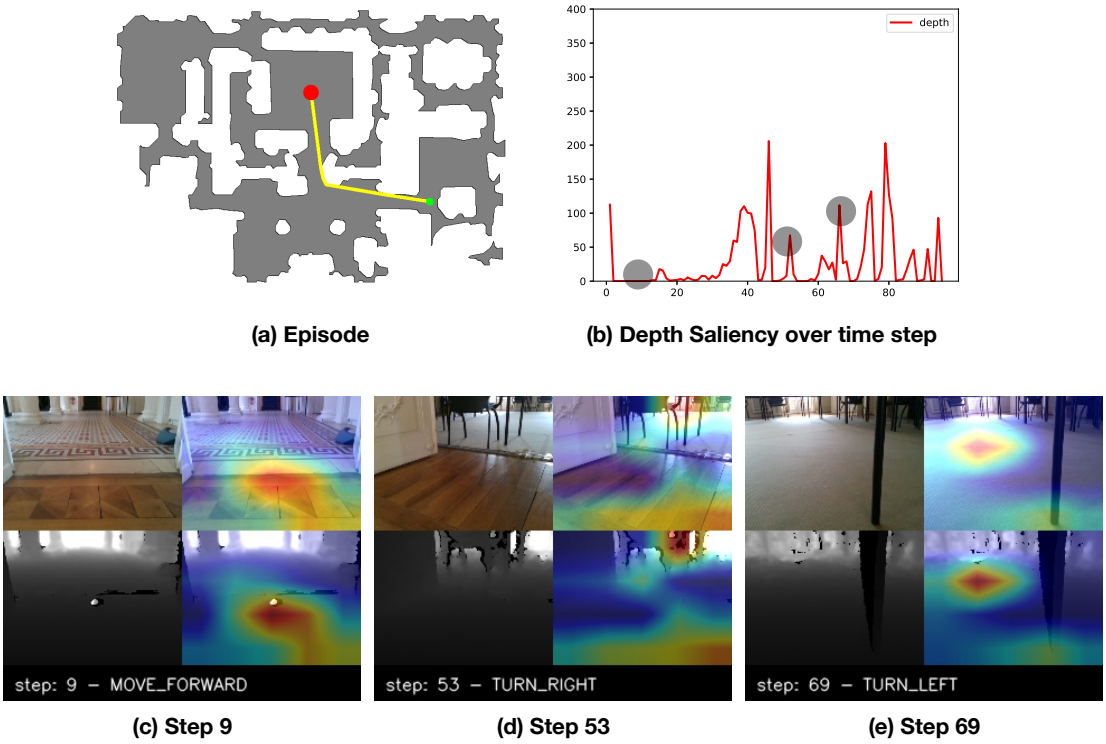
(a) Episode (b) Depth Saliency over time step

(c) Step 9 (d) Step 16 (e) Step 19

Figure 3.5 – The same boxes in figure 3.4 are still visible in the scene, but do not block the navigation towards target. Black dots on the saliency curve indicate the steps shown.

the last convolutional layer in the feature extractor of the RGB-D observation and overlay its gradient on the input image in pseudo colors, c.f. Figuress. 3.4, 3.5, 3.6 and 3.7. In these figures we also provide an indication of the importance of the visual input, which corresponds to the accumulated gradients. We saw that the importance of the GPS sensor stayed relatively flat.

In Figure 3.4, we visualize the impact of blocking obstacles (boxes) placed in front of the goal, on sensor usage. This obstacle should in principle require the agent to use the visual observation, as the GPS direction does not point towards a feasible path. The agent indeed focuses on the obstacles as they appear in front of the camera, verified in the Grad-CAM heatmap. In the selected three steps shown in the Figure we can see that, when the agent was fully blocked by the obstacles, the saliency value of the depth image peaks, indicating its influence on the chosen avoidance actions: `TURN_LEFT` in step 22 and `TURN_RIGHT` in step 23 as counter-action. After the peak is reached, the agent had to navigate a half-circle around the obstacle, during which the importance of the depth input started to decreases until the goal was reached. To check whether attention is



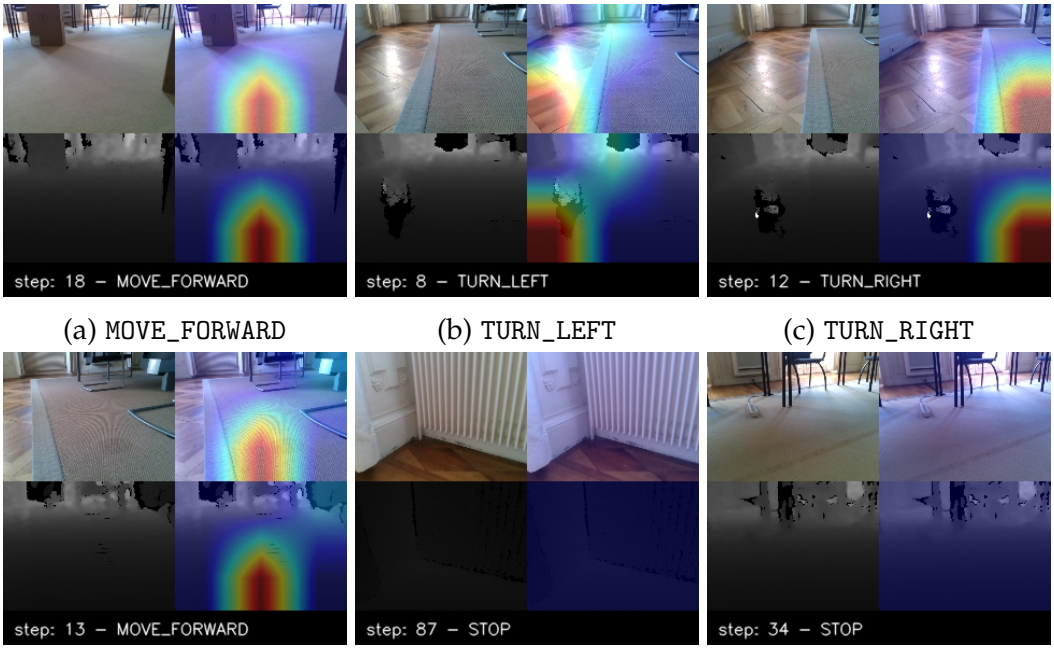
(a) Episode (b) Depth Saliency over time step

(c) Step 9 (d) Step 53 (e) Step 69

Figure 3.6 – Impact of navigation in a non-straight L-shape line on visual attention. The agent pays attention to visual input when it considers a direction change.

caused by visual saliency of the cardboard boxes, we created a "control episode" where these boxes are in a non-blocking placement (figure 3.5) — the agent had a significantly decreased (or no) focus on the depth, except when it needed to verify that blockage might happen, in step 16.

In Figure 3.6 we visualize an episode with an L-shaped trajectory, thus different from a straight path, which would have otherwise allowed to plainly follow the GPS directions. The depth input was ignored in the first part of the episode consisting mostly of forwards moves. However, once the agent needs to find the turning point, requiring to differentiate between `MOVE_FORWARD` and `TURN_RIGHT`, depth usage jumps up and the heatmap indicates that the action taken at each time step is strongly related to the regions where the agent looks at (left region for left turns etc.). We have cross checked several episodes to confirm this tendency, shown in Figure 3.7 — the agent has a strong tendency to look at the region towards which it will navigate with a turning action. Not surprisingly, the visual input is unused when the goal is reached and `STOP` is called, as this can be decided from the distance value.



(a) MOVE_FORWARD (b) TURN_LEFT (c) TURN_RIGHT

(d) MOVE_FORWARD (e) STOP (f) STOP

Figure 3.7 – Visual attention and motion: the agent has a strong tendency to attend to regions it will navigate to through turns.

		Seen			Unseen			Overall		
Scene	Agent	SR	SPL	sSPL	SR	SPL	sSPL	SR	SPL	sSPL
NLE real	*Finetuned*	**100**	**88.7**	**86.3**	66.7	55.6	54.6	90.9	79.7	77.6
	ROS	**100**	79.7	78.3	*100*	*96.5*	*95.4*	**100**	**84.3**	**83.0**
INSAL real	*Finetuned*	**100**	92.8	90.6	**100**	**91.6**	89.1	**100**	92.2	89.8
	ROS	**100**	**93.9**	**92.8**	*100*	*91.6*	*90.4*	**100**	**92.8**	**90.4**

Table 3.2 – Quantitative results for experiments on real physical robots comparing the classical planner (ROS) to the best trained agent. *Gray* font denotes that the ROS agent was given the full map of the scenes, not just the seen regions.

Classical vs. ML Table 3.2 compares the best trained agent to the classical baselines from the ROS navigation stack. As expected, the classical baselines slightly outperform the trained agent, but their performance is quite comparable. We insist again, that this comparison was not the goal, we report it for the sake of completeness.

Comparison Real vs. Noisy Sim The evaluation of a physical robot in the real world is a time-consuming exercise, we therefore provide a comparison to another

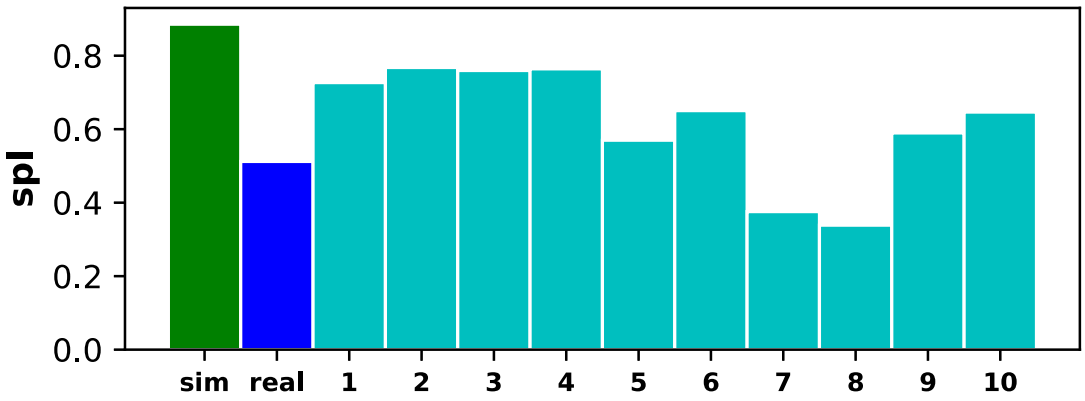


Figure 3.8 – Performance on noisy sim (Targeted agent), compared to sim and real (blue). Noise variants are (cyan):
(1-4): have Redwood Depth noise of intensity 6 and RGB noises based on different distributions: 1: Gaussian, 2: Speckle, 3: Salt & Pepper, 4: Poisson;
(5-6): applied Gaussian noises with different Redwood Depth Noises (intensity 3 and 9 respectively).
(7-9): Gaussian noises simulated on actuators based on three common controllers described by (Murali et al. 2019): 7: Proportional Controller (P), 8: Dynamic Window Approach Controller from Movebase (MB), 9: Linear Quadratic Regulator (ILQR).
(10): noise settings of CVPR Habitat challenge 2021 (Gaussian, Redwood with Intensity 1 and Proportional Controller noises).

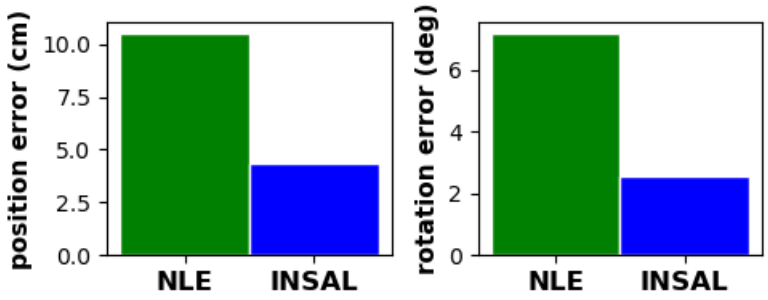


Figure 3.9 – Error in position (left) and angle (right) between the controller target and the actual motion done by the robot, on the two scenes: "*NLE*" (green) and "*INSAL*" (blue). Errors are w.r.t. to the estimate obtained by the ROS-NavStack.

common practice in embodied computer vision and robotics, evaluation in noisy simulated environments. The goal is to evaluate whether "noisy sim" can and should be chosen as a proxy for real experiments, and which types of noise should be chosen.

We evaluated the agent on the same evaluation episodes ("*NLE*" environment) on different environment variants, which differ in noise configurations, as shown in Figure 3.8. We can see that most of the noise techniques on the visual sensors are not representative and do not strongly impact performance. However, noise on the actuators does have a strong impact and can show similar or even worse performance than in the real environment.

We investigate actuator noise further by measuring the difference between the position difference planned by a single control step and the actual position difference performed by the robot, shown in Figure 3.9. We found that the physical agents need to perform far more actions for the same trajectory than the simulated one. The gap in error is also partly due to difficult floor conditions — the "*NLE*" scene was furnished with thick carpets, which created more challenges to the robot than the more modern "*INSAL*" scene.

3.5 Conclusion

We have evaluated three variants of agents on two different real physical environments to benchmark the generalization capabilities of physical robots agent in the real world. We showed that for the PointGoal task, an agent pre-trained on wide range of scenes and finetuned on a targeted scene in simulation can reach a high performance and reduce the sim2real gap without the need of any sim2real transfer technique. We also conducted in depth visualization for the sensor usage in the neural network to understand the visual reasoning of the agent, showing that the agent indeed puts a attention on the visual information when needed.

In this chapter, we have illustrated the agent reasoning and generalization capacity in real environment on the fundamental task of *PointNav*. In the next chapter, we want to move a step further by exploring and evaluating a more complex task in real environment that involves the semantics awareness of the agent and its memorization capacity. Therefore, we next focus on the *MultiON* task (Section 2.1.2). For such a task, we want to build a novel hybrid agent with a modular design that disentangles the multi-objects navigation task into sub-tasks (sub-skills). By this disentanglement, we believe that hybrid agent would perform better than other approaches who tackle the task in an end-to-end manner. Possibly, this can reduce the challenge of the sim2real gap that can encounter the hybrid agent without the need of any sim2real transfer technique.

CHAPTER 4

MULTI-OBJECT NAVIGATION IN REAL ENVIRONMENTS USING HYBRID POLICIES

Contents

Chapter abstract

Navigation has been classically solved in robotics through the combination of SLAM and planning. More recently, beyond waypoint planning, problems involving significant components of (visual) high-level reasoning have been explored in simulated environments, mostly addressed with large-scale machine learning, in particular RL, offline-RL or imitation learning. These methods require the agent to learn various skills like local planning, mapping objects and querying the learned spatial representations. In contrast to simpler tasks like waypoint planning (PointGoal), for these more complex tasks the current state-of-the-art models have been thoroughly evaluated in simulation but, to our best knowledge, not yet in real environments.

In this work we focus on sim2real transfer. We target the challenging Multi-Object Navigation (Multi-ON) task (Wani et al. 2020) and port it to a physical environment containing real replicas of the originally virtual Multi-ON objects. We introduce a hybrid navigation method, which decomposes the problem into two different skills: (1) waypoint navigation is addressed with classical SLAM combined with a symbolic planner, whereas (2) exploration, semantic mapping and goal retrieval are dealt with deep neural networks trained with a combination of supervised learning and RL. We show the advantages of this approach compared to end-to-end methods both in simulation and a real environment and outperform the SOTA for this task (Marza et al. 2022).

The work in this chapter has led to the publication of a conference paper:

- Assem Sadek, Guillaume Bono, Boris Chidlovskii, Atilla Baskurt, and Christian Wolf (2022a). "Multi-Object Navigation in real environments using hybrid policies". In: *ICRA 2023*;

4.1 Introduction

Robot navigation has progressed from simple waypoint navigation problems in richly prepared environments, for which robustly working systems are now used in production, to complex tasks involving high-level reasoning with visual and semantic concepts. This has been made possible through large-scale machine learning, mostly in photo-realistic simulators and reinforcement learning from billions of interactions. The resulting robotic agents, implemented as high-capacity neural networks, which are however subject to performance drop and lack of robustness when the policies are transferred from simulation to real environments with physical robots. This is mostly due to the gap in realism between simulation and reality ("sim2real gap") which we studied in Chapter 3, as well as the difficulty to explore a large amount of variation factors inherent in navigation problems, such as room layouts, furniture, textures and other room details, rare local scene geometries etc.

There is a recent trend towards modular approaches which decompose the problem into hierarchical parts (Chaplot et al. 2020b; Beeching et al. 2020c) and hybrid approaches, which combine a shortest path planner (symbolic or trained) with a trained policy. While these approaches have been shown to be more sample efficient (Ramakrishnan et al. 2022), state-of-the-art methods are still evaluated in simulation and lack thorough tests on real robots.

In this work we address the challenging problem of Multi-Object Navigation (Wani et al. 2020), which, similarly to the K-items scenario (Beeching et al. 2020b), requires an agent to sequentially navigate through a set of objects in an imposed order. This task definition favors agents capable of learning to map seen objects in an internal spatial representation, as navigating to them later in the episode can increase reward. This makes it stand out with respect to simpler tasks like ObjectNav, where the combined capacities of exploration and reactive local planning from the current observation are sufficient to solve the task [1].

As in the last chapter, we target sim2real transfer and to our best knowledge, are the first to perform a thorough performance evaluation of a method on Multi-Object Navigation in a real physical environment, see Figure4.1. While simpler tasks, such as PointGoal, have been evaluated on real robots (Kadian et al. 2020; Sadek et al. 2022b), evaluation of trained models on more complex tasks has

1. Regularities in spatial layouts may be exploited with an additional form of higher reasoning, for instance with potential fields (Ramakrishnan et al. 2022), but we do not focus on these aspects in this work.

Figure 4.1 – We perform Multi-Object Navigation (Wani et al. 2020), i.e. the sequential visual search of multiple object in a given order, and are the first do this in real physical environments (a) characterized by a large sim2real gap. This is illustrated by two first-person views (b), real and (c), simulation. We propose a hybrid method combining classical mapping and deep learning, and compare to the SOTA methods on this task using end-to-end RL training and auxiliary losses (Marza et al. 2022).

been sparse or nonexistent. We present a new method for navigation, whose design choices have been driven by the objective of optimizing performance in real environments. We propose a new hybrid method which decomposes the problem into two parts:

① **"Good Old Fashioned Robotics"(GOFR)**, that deals with classical navigation aspects not related to semantics, such as detection of navigable space and localization (geometric SLAM) combined with waypoint navigation on this map.

② **Semantics through Machine Learning**, i.e. mapping semantic concepts required for visual reasoning and exploiting them; exploration of the most promising areas of the environment exploiting layout regularities.

During navigation, a classical SLAM algorithm (Labbé et al. 2019) creates and maintains a 2D metric representation in the form of a tensor/map and localizes the robot on it using LIght Detection And Ranging (LIDAR) input. High-level features, extracted from visual RGB-D observations with a deep neural network, form a spatial and semantic point cloud, whose spatial coordinates are aligned with the metric representation, see Figure 4.2. The combined hybrid representation satisfies

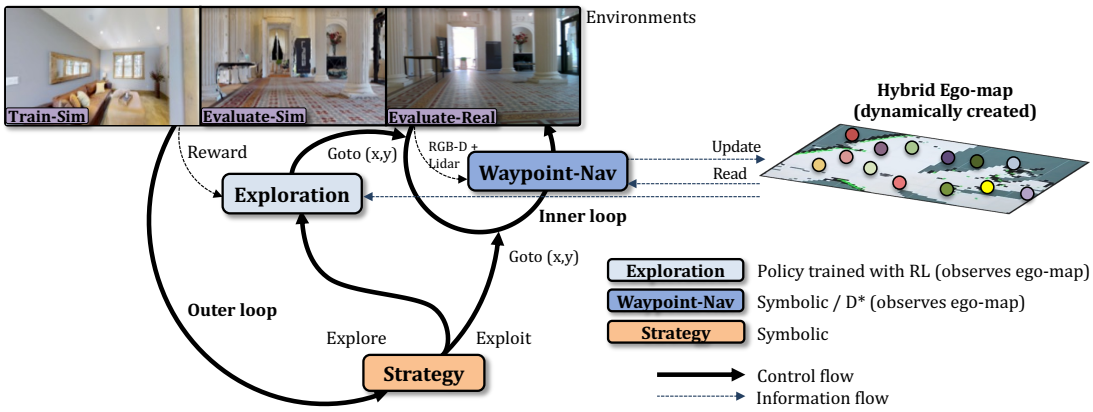


Figure 4.2 – An agent for multi-object navigation maintains a hybrid representation consisting of a metric bird's eye view map combined with a semantic point cloud. The agent switches between a trained exploration policy and symbolic waypoint selection, deferring low-level actions to a symbolic planner.

the needs of relevant sub-skills the agent requires: (i) to determine whether a target object has been observed in the past, (ii) to plan optimal trajectories between the agent and explored areas, and (iii) to determine the frontiers of unexplored areas in the environment and thus the next intermediate sub-goals in case the environment needs to explore to find the next goal. All these sub-skills are designed and trained separately, which allows to limit sample complexity of training.

The contributions of this work are the following: (i) we introduce a hybrid method for Multi-Object navigation combining classical metric SLAM and path planning with learned components trained with supervised learning and RL; (ii) we reproduce the Multi-ON benchmark (Wani et al. 2020) in a real environment, where we place manufactured reproductions of the goal objects, used in originally simulated target environment; (iii) we compare the proposed method to end-to-end trained methods in this real environment, in particular with the winning entry of the CVPR 2021 Multi-ON competition (Marza et al. 2022), which we outperform in both real and simulated environments.

4.2 Related work

Modular Embodied Navigation Casting a task as an end-to-end learning problem is widely used in CV and NLP. Navigation has been addressed through this lens early on, e.g. for exploration (Chen et al. 2019), where a neural policy processes raw sensory observations and directly predicts agent actions. However, jointly learning mapping, state-estimation and path-planning purely from data has

been shown to be expensive (Chaplot et al. 2020b). An alternative are hierarchical and hybrid architectures (Beeching et al. 2020c; Chaplot et al. 2020b; Chaplot et al. 2020a; Chaplot et al. 2020c) that compose a learned mapper with global and local policies, where all components interface via the map and an analytical path planner. Pushing modularity further, Ramakrishnan *et al.* (Ramakrishnan et al. 2022) propose to disentangle the skills of *'where to look?'* from navigation itself. The network predicts potential functions conditioned on a semantic map and uses them to decide where to look for an unseen object.

Sim2Real The sim2real gap can compromise strong performance achieved in simulation when agents are tested in the real-world (Höfer et al. 2020), as perception and control policies often do not generalize well to real robots due to inaccuracies in modelling, simplifications and biases. To close the gap, domain randomization methods (Peng et al. 2018; Tan et al. 2018) treat the discrepancy between the domains as variability in the simulation parameters. Alternatively, domain adaptation methods learn an invariant mapping for matching distributions between the simulator and the robot environment. Examples include transfer of visuo-motor policies by adversarial learning (Zhang et al. 2019), adapting dynamics in RL (Eysenbach et al. 2021), and adapting object recognition to new domains (Zhu et al. 2019). Bi-directional adaptation is proposed in (Truong et al. 2021b); Recently, Chattopadhyay *et al.* (Chattopadhyay et al. 2021) benchmarked the robustness of embodied agents to visual and dynamics corruptions. Kadian *et al.* (Kadian et al. 2020) investigated sim2real predictability of Habitat-Sim (Savva et al. 2019) for PointGoal navigation and proposed a new metric to quantify it, called Sim-vs-Real Correlation Coefficient (SRCC). The PointGoal task on real robots is also evaluated in (Sadek et al. 2022b). To reach competitive performance without modelling any sim2real transfer, the agent is pre-trained on a wide variety of environments and then fine-tuned on a simulated version of the target environment.

Memory and Maps (Inductive Bias) Memory is a crucial aspect of an intelligent agent's ability to reason about 3D space and geometry. Neural memories like NeuralMap (Haarnoja et al. 2018), MapNet (Henriques et al. 2018) and propose latent metric maps, which are updated incrementally from the camera observations and odometry and act as inductive bias for end-to-end training. EgoMap (Beeching et al. 2020c) augments these maps with multi-step objectives and attention reads, trained with RL. Alternative neural maps are topological maps (Savinov et al. 2018; Chaplot et al. 2020d; Beeching et al. 2020c), transformers (Pashevich et al. 2021; Fang et al. 2019; Chen et al. 2021; Janner et al. 2021; Chen et al. 2022a; Reed et al. 2022), which break the Markovian assumption and attend to a large temporal horizon and implicit representations (Li et al. 2022c; Adamkiewicz et al. 2022).

Exploration is at the core of all navigation tasks (Chaplot et al. 2020b) and is in itself studied and evaluated (Anderson et al. 2018a). Efficiently visiting the environment is useful for solving tasks in known environments and pre-mapping in unknown ones. Chen *et al.* (Chen et al. 2019) explored policies with spatial memory that are bootstrapped with imitation learning and finetuned with coverage reward. In (Chaplot et al. 2020c), an exploration policy is trained by introducing semantic curiosity based on observation consistency. SEAL (Chaplot et al. 2021) trains perception models on internet images to learn an active exploration policy. They build 3D semantic maps to learn both action and perception models, and integrate intrinsic motivation. Episodic semantic maps are proposed in (Chaplot et al. 2020a).

4.3 Hybrid planning and navigation

We target the task of Multi Object Navigation (Multi-ON) introduced by Wani *et al.* (Wani et al. 2020), in particular the 3 object variant: during each episode, the agent has to find 3 cylindrical objects G_n, $n = 1, 2, 3$, in a pre-defined order, where G_n is the n^{th} object to find, and is required to call the *Found* action at each goal. The episode duration is limited to 2,500 environment steps. At each step t, the agent receives an egocentric RGB-D observation $\mathbf{O}_t \in \mathbb{R}^{h \times w \times 4}$, a Lidar frame, and the class label of the current target object taken from 8 classes. All training was performed in simulation only with the Habitat simulator (Savva et al. 2019), but the system was evaluated, both, in simulation and on a real Locobot robot in a real environment, more details are given in Section4.4.

With operations on robots in real environment and conditions in mind, we follow a modular approach, outlined in Figure4.2. The method is hybrid; it leverages both trained neural modules for perception and exploration, and classical algorithms for occupancy mapping, localization and waypoint navigation. The main motivation behind this approach is a maximum reduction of the sim2real gap, avoiding the main pitfalls of end-to-end training of navigation in simulation followed by a transfer of neural models to the real environment. We explore an approach that prefers classical methods based on sensor models and optimization, motivated by their robustness, and employ machine learning in a targeted way for parts of the system where its use is both necessary and beneficial. We also limit input to trained models to representations with a potentially low sim2real gap. For this reason, during navigation the agent builds a metric bird's eye view occupancy map from the Lidar input and localizes itself on it using metric SLAM (Thrun et al. 2005). This binary map is combined with an overlaid semantic point cloud, which contains the positions of key objects and their semantic classes, which are detected from the RGB input with an object detector. Detection and mapping are aligned through the SLAM algorithm's localization module.

Navigation is performed hierarchically on two different levels. On a higher level (outer loop in Figure4.2), 2D waypoint coordinates $\mathbf{p}_t=(x,y)$ are produced and provided to the lower level controller (inner loop), whose task is to navigate to the waypoint using the maintained occupancy map. The high-level controller switches between two different strategies:

① **Exploration** — when the target object has not yet been observed, i.e. the robot explores the environment, maximizing coverage. This is done with a learned policy trained with RL, see below.

② **Exploitation** — when the target object has been observed and thus is part of the semantic point cloud, its location is taken as a new waypoint and given to the local planner.

Metric EgoMap To gather navigability information along its path and more efficiently revisit previously seen areas, the agent builds what is called an EgoMap, an occupancy grid of fixed spatial resolution centered on its current position and aligned with its heading direction.

On the real robot, this map is obtained using the RTABMap (Labbé et al. 2019) library. It uses a graph-based SLAM algorithm with loop closure, a flexible design taking advantage of RGB-D, Lidar and odometry sensor data. Lidar and/or depth are used to create a 2D/3D local occupancy grid, associated to a node whose initial position relies on odometry integration. Descriptors are then created from keypoints extracted from RGB frames in order to facilitate node comparison and loop closure detection. RTABMap also includes short- and long-term memory management, global map compression and multi-session mapping.

In simulation, we take advantage of privileged information to retrieve a complete top-down view of the scene navigability, through a projection of the NavMesh generated by the Recast&Detour (*Recast & Detour library* 2016) library in Habitat-Sim. A fog-of-war mask is then built by ray-tracing in the agent's field of view directly on this top-down view using perfect localization.

Both real and sim approaches generate a global map on which we apply a simple affine transformation parameterized by the agent's current pose to get the EgoMap.

Exploration is the main module based on machine learning. In contrast to most recent work in embodied AI (Beeching et al. 2020c; Chaplot et al. 2020b), the policy does not take the first person RGB input, but the EgoMap $\mathbf{M}_t$ produced by the metric SLAM algorithm. This leads to a significant simplification of the task and increased sample efficiency, and it minimizes the sim2real gap, as changes in lightning, color and texture are avoided. The policy is a part of the outer loop and predicts 2D waypoint coordinates $\mathbf{p}_t$. The problem is partially observable for multiple reasons: (i) not all areas of the scene have been observed at any point

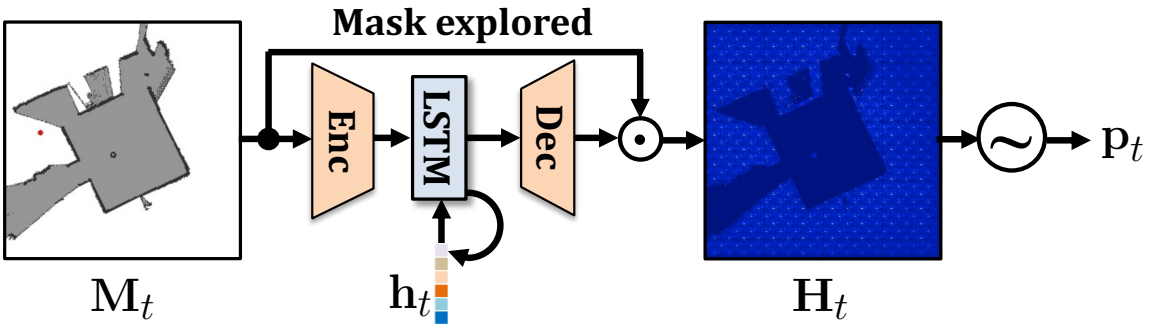


Figure 4.3 – The exploration policy takes as input EgoMaps $\mathbf{M}_t$ and predicts a heatmap, which is limited/masked ($\odot$) to unexplored areas. The next waypoint $\mathbf{p}_t$ is sampled ($\sim$) from the resulting map $\mathbf{H}_t$.

in time; (ii) for efficiency reasons, the EgoMap $\mathbf{M}_t$ does not cover the full scene, observed areas can therefore be forgotten when the agent navigates sufficiently far away from them; (iii) even theoretically fully observable problems (MDPs) can be transformed into POMDPs ("*Epistemic POMDPs*") in the presence of uncertainty in the environment, which is a standard case in robotics, as has recently been shown in (Ghosh et al. 2021). We therefore imbued the policy with hidden memory $\mathbf{h}_t$ and made it recurrent.

The policy π needs to be able to predict multi-modal distributions, as there are multiple valid trajectories exploring an environment efficiently. We baked this into the policy through an inductive bias, which forces prediction to pass through a spatial heatmap $\mathbf{H}_t$, from which the chosen waypoint location is sampled. Before sampling, we restrict the heatmap to unexplored areas through masking. This choice also leads to a more interpretable model, as the distribution of targeted exploration points can be visualized (see Section4.4). This can be formalized as follows (see also Figure 4.3):

$$\mathbf{h}_t = \phi(\mathbf{M}_t, \mathbf{h}_{t-1}; \theta_\phi) \tag{4.1}$$

$$\mathbf{H}'_t = \pi(\mathbf{h}_t; \theta_\pi),\ \mathbf{H}_t = \mathbf{H}'_t \odot [M_t == \text{"Unexplored"}]) \tag{4.2}$$

$$\mathbf{p}_t = \sim (\mathbf{H}_t), \tag{4.3}$$

where θ_π and θ_ϕ are trainable parameters and ϕ is the update recurrent function of an LSTM with hidden state $\mathbf{h}_t$; gates have been omitted in the notation for simplicity. Here, $\mathbf{p}_t{=}(x_t, y_t)$ is 2D coordinates of the point sampled in the spatial heatmap $\mathbf{H}_t$, '$\sim$' is the sampling operator.

We train the exploration policy with RL to maximize coverage and use the following reward function r_t:

$$r_t = \alpha r_{de}, \quad r_{de} = e_t - e_{t-1}, \tag{4.4}$$

where e_t denotes the explored area at step t, l is the number of inner environment steps necessary to navigate to the coordinates (x, y) predicted by the policy, α is a scaling hyper-parameter set to 0.01.

Local navigation to the waypoint $\mathbf{p}_t$ is performed by an analytical planner that computes the shortest path on the current occupancy EgoMap $\mathbf{M}_t$. This is not necessarily the optimal path, as the map is not equal to the (unobserved) GT map and the intermediate regions to be traversed (and even the waypoint $\mathbf{p}_t$) might be unexplored. We employ a dynamic planner D* which calculates the shortest path under classical assumptions and replans when new information is available. Since we optimize our method to be robust and efficient in real environments, unlike recent work (Ramakrishnan et al. 2022; Chaplot et al. 2020a) we choose a D* planner over the commonly used Fast Marching Method (Sethian 1996b). The path feasibility in real and the speed of planning are the two main reasons behind this design choice.

Stabilizing training The potential failures and sub-optimal trajectories produced by local planning in uncertain conditions using D*, as described above, also negatively impact the training process of the exploration policy. This policy is a part of the outer loop and predicts waypoints $\mathbf{p}_t$, receiving a reward only upon completion of the full local navigation process. Noise in local planning impacts the stability of the RL training process and leads to lack of convergence.

We solved this by training the exploration policy interfacing a local policy on which we imposed a length limit. The full trajectory from the current position to the next waypoint $\mathbf{p}_t$ predicted by the exploration policy is split into a sequence of small sub goals distanced by 0.3m, and the local policy is limited to 5 of these sub goals. Control is given back to the outer loop if the waypoint $\mathbf{p}_t$ has been reached, or the limit of 5 subgoals is reached. This choice lead to stable training and the trained policy transferred well to the targeted exploration task, without changes. The same limitation on the length of local planning is also applied at deployment, which led to improved robustness in real conditions and makes complex recovery behavior obsolete.

Object Detection and Mapping is framed as a semantic segmentation task from the current RGB-D frame o_t, which we supervise from GT masks calculated from privileged information in the simulator. The predictor is a DeepLab v3 network (Chen et al. 2017), detected objects in the mask are inversely projected and aligned with the EgoMap using depth information and the episodic odometry. Note that both, depth and odometry, are noisy in the real robot / real environment evaluation settings.

High-level decisions are fully handcrafted, as this leads to a robust and transferable decision process where learning is arguably not required. Given our decision choices, only one type of decision is required, whether to perform exploration or exploitation (i.e. navigation towards the goal). This is taken on the basis whether the current target object has been observed at mapped, or not. If multiple

Method	— LIDAR Usage — Map	Low-lev cntrl	Aux losses(Marza et al. 2022)	Obj. Segm.
ProjNMap+AUX (Marza et al. 2022)	–	✓	✓	–
Ours	✓	✓	–	✓

Table 4.1 – Comparability of the different methods in terms of sensor and information availability. Both methods use LIDAR.

objects of the same class have been detected, the location with the most probably detection (in terms of segmented object pixels) is chosen. A minimum number of pixels is required for an object to be mapped.

4.4 Experimental Results

Simulation for training in simulation and for additional evaluation (complementary to experiments on the real platform) we used the photo-realistic Habitat simulator (Savva et al. 2019) and two datasets with 3D scanned environments, each with the standard train/validation/test split: (i) the Gibson dataset (120 scenes) and the Matterport 3D dataset (90 scenes).

Real environment To evaluate the method in a real environment, we used the same platform also employed during the work described in the last chapter: a LoCoBot robot (*LoCoBot: An Open Source Low Cost Robot* 2017) [] equipped with an Intel *RealSense* RGB-D camera and a single-ray Lidar of type *RPLIDAR A2M8* (see Figure 4.1), which we restrict to Field-Of-View equivalent to the RGB camera. We used the publicly available *habitat_sim2real* library (*Habitat Sim2real* 2021), which allows to connect a real robot under `ROS` to the Habitat simulator as an agent. We perform tests in a building with classical architecture covering one large conference room and several adjoining rooms, cf. Figure 4.1, and for a map, Figure 4.5. The environment features difficult conditions including windows and glass panels, thick carpets, textureless walls, etc.

Baselines We compared with two end-to-end methods:

- **ProjNMap** (Projective Neural Map) is based on projective neural memory as inductive bias for neural networks (Henriques et al. 2018) and has been explored for the specific task of Multi-ON in (Wani et al. 2020), providing the best results in the original experiments when the task was introduced.

- **ProjNMap+AUX** combines the original ProjNMap model with auxiliary losses on an additional head, which predicts the direction and distance to the current

		M-ON Setup			
Dataset	Agent	**Progress**	**PPL**	**Success**	**SPL**
MP3D	ProjNMap	41.63	21.81	23.10	14.41
	ProjNMap+AUX	**62.47**	**35.21**	**48.20**	**62.47**
	Hybrid (Ours)	55.23	12.41	41.80	11.72
10 episodes	ProjNMap	23.33	14.39	10.00	9.21
	ProjNMap+AUX	43.33	**28.19**	30.00	**24.88**
	Hybrid (Ours)	**50.00**	9.20	**50.00**	9.20

"M-ON": CVPR 2021 Multi-ON challenge sensor settings.

(a)

		LoCo Setup			
Dataset	Agent	**Progress**	**PPL**	**Success**	**SPL**
MP3D	ProjNMap	35.47	**23.97**	19.10	15.49
	ProjNMap+AUX	59.97	23.29	33.40	**19.75**
	Hybrid (Ours)	**63.37**	19.72	**50.80**	18.44
10 episodes	ProjNMap	16.67	**13.63**	10.00	9.52
	ProjNMap+AUX	40.00	12.77	30.00	10.65
	Hybrid (Ours)	**56.67**	12.09	**50.00**	**11.99**

*"LoCo": Sensor settings equivalent to the physical robot: FoV=56°, RGB size=*160×120.

(b)

Table 4.2 – Performance in Simulation (Habitat) for two different environments: the Matterport 3D validation set (comparable with the CVPR 2021 Multi-ON challenge), and 10 test episodes of the simulated version of our real environment. The agent sensor configurations used are compatible with CVPR21 challenge (Table a) and the physical robot (Table c) used in real environment evaluation (Table 4.3)

target during training (Marza et al. 2022). This method achieved the winning performance in the CVPR 2021 M-ON Challenge (*The CVPR 2021 Multi-Object Navigation Challenge* 2021) and is the current state-of-the-art on this task.

Table 4.1 summarizes sensor usage and information availability of the main baseline (Marza et al. 2022) compared to our method. All methods use Lidar: ours - to maintain a metric map, the baseline (Marza et al. 2022) - for a localization step necessary to perform closed-loop low-level control, mapping the discrete action space of the neural agent to the continuous motor space of the robot. All methods use information on object positions *during training*: ours - in the pre-training step of the visual encoder, the baseline (Marza et al. 2022) - through auxiliary losses.

Agent	Progress	PPL	Success	SPL
ProjNMap	3.00	0.85	0.00	0.00
ProjNMap+AUX	0.00	0.00	0.00	0.00
Hybrid (Ours)	43.10	6.02	20.00	4.99

Table 4.3 – Performance in the **real environment** by the physical Locobot on 10 test episodes. We compare with the CVPR 2021 Multi-ON Challenge winner (Marza et al. 2022) (current SOTA).

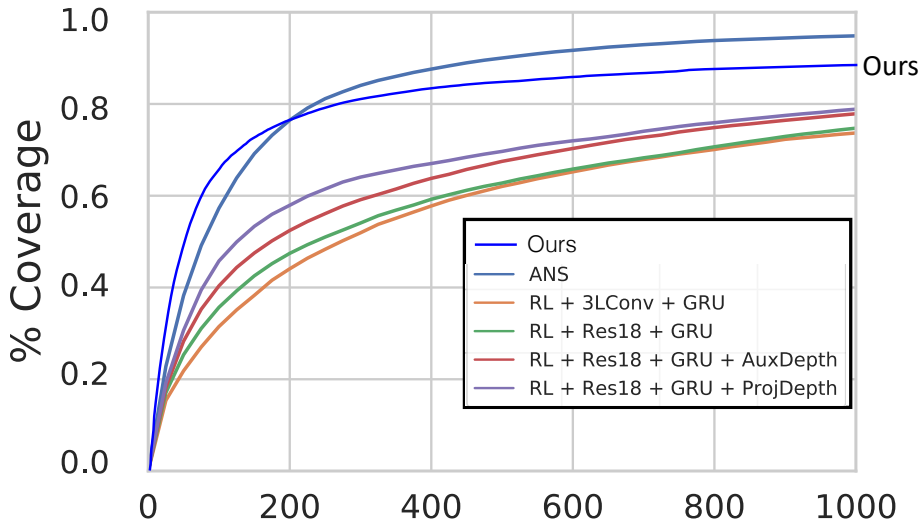


Figure 4.4 – Coverage (%) obtained by the exploration policy as a function of episode length (the number of simulation steps), compared to ANS (Chaplot et al. 2020b) and end-to-end RL baselines using egocentric input taken from (Chaplot et al. 2020b) on Gibson/Val.

Configurations With different evaluation goals in mind, we created two configurations of the agent:

- **Locobot**: this configuration corresponds to the physical robot (Locobot) and its sensors. We also created a corresponding Habitat simulator configuration, which is equal to these settings: FOV of 56° (camera+Lidar), frame size of 160×120 and a compatible camera position. This configuration is of double use, i.e. can be used for evaluation in both simulation and the real environment.

- **Multi-ON-Compat**: we also reused the settings of the Multi-ON benchmark, making this configuration compatible with prior work like (Marza et al. 2022; Wani et al. 2020). This includes a FOV of 79° and camera frames of size 256×256. This configuration can be used in simulation only.

Agent	Coverage (%)	
	Gibson	MP3D (domain generaliz.)
RL+3LConv+GRU	73.7	33.2
RL+Res18+GRU	74.7	34.1
RL+Res18+GRU+AuxDepth	77.9	35.6
RL+Res18+GRU+ProjDepth	78.9	37.8
ANS (Chaplot et al. 2020b)	**94.8**	52.1
Ours	88.4	**67.14**

Table 4.4 – Coverage obtained by different exploration policies on Gibson and MP3D. All agents were trained on the Gibson train split (Results on competing methods taken from (Chaplot et al. 2020b)).

Setup and hyper-parameters decision thresholds are set as follows: at least 0.7% of detected pixels is required for an object to be placed on a map; 5% or more of detected pixels is required for an object to trigger the *Found* action.

Results in Simulation are shown in Table4.2. They have been obtained on two different datasets: (i) on the validation split of the Matterport 3D dataset, making these runs comparable to the validation entries of the CVPR 2021 Multi-ON competition, and (ii) on 10 episodes in a 3D scanned version of our real environment, shown in Figure4.2. These 10 episodes correspond to simulated versions of the episodes tested in the real environment, see further below. This simulated environment has *not* been used for training.

While our main design choices are biased toward a robust performance in real environments, we can see that it is also highly performant in simulation. In the LoCo Setup, where the (virtual) sensor configuration mirrors real sensors, the method outperforms the state of the art on the *Progress* and *Success* metrics and is competitive in the others.

Results in the real environment are shown in Table4.3, on the same 10 episodes that we tested in simulation. Our hybrid agent was able to collect 43% of the targets successfully and even finished 2 out of 10 episodes retrieving all 3 required items. We conjecture that this is due to the strategy to disentangle perception, exploration, and waypoint navigation, which allows for keeping the sim2real gap lower than what can be done for the end-to-end (E2E) trained methods.

On the other hand, the performance of the baselines can be considered a failure. While it has been reported, that end-to-end training of the simpler PointGoal task in similar conditions can be successful (Sadek et al. 2022b), this did not apply to our experiences on the much more complex Multi-ON task. While (Marza et al. 2022) obtained the state of the art in the official benchmark, i.e. in simulation, not a single episode was successful in the real environment. Most failure cases were related to the poor exploration of the scene and high uncertainty in detecting

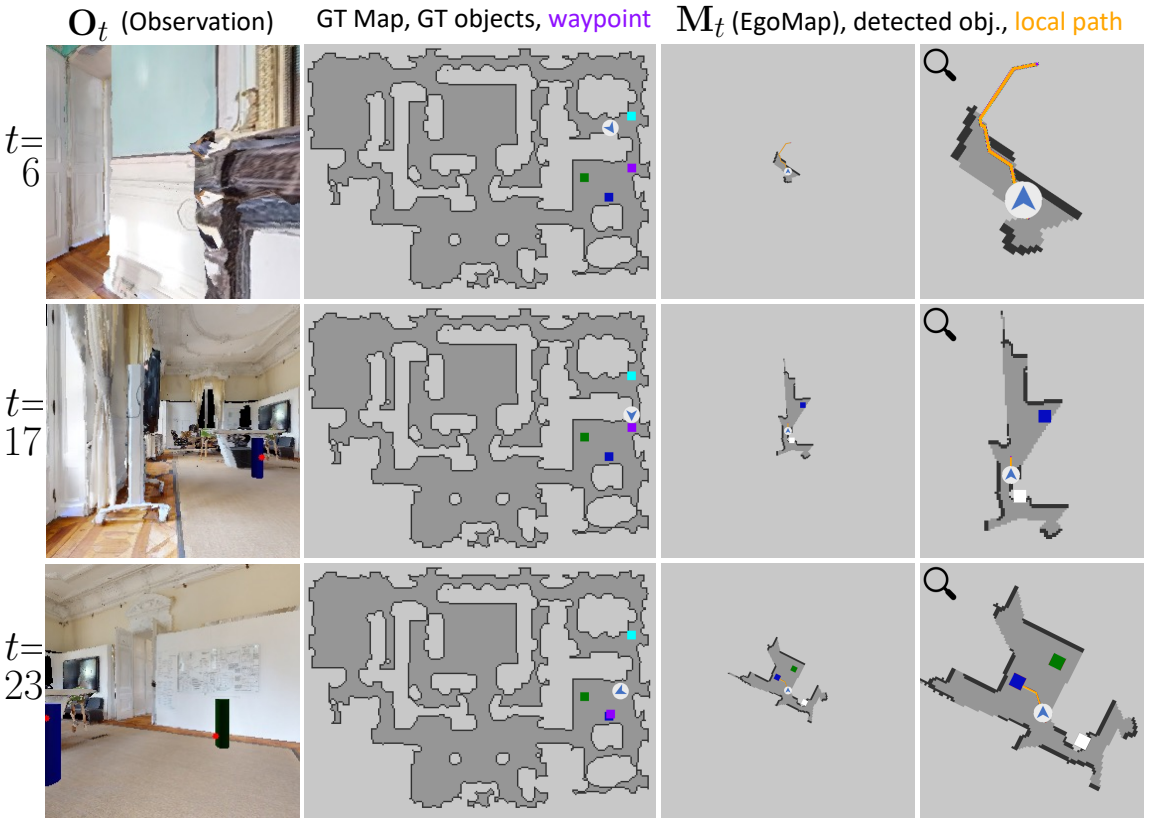


Figure 4.5 – A rollout of an episode with the hybrid model. From left to right: (1) RGB observation; (2) GT map with the GT goal positions, the current agent position, the current waypoint p_t; (3) EgoMap M_t with the planned local path and (4) a zoomed version. The initial goal is blue. At t=6, an exploration goal is predicted. The agent enters a new room, and at t=17 it detects the **blue goal** and switches to exploitation mode advancing towards it. At t=23, it observes the very dark **green goal** and maps it for future use. A false positive example (white cylinder) was also detected.

the targets. The agent had a hard time changing rooms and repeatedly failed to apply 'Found' when the object is closely upfront. We conjecture that the high impact of the sim2real gap on raw sensor data(cf Figure 4.1) requires to adapt the E2E methods on real data for such complex tasks. Although training with real data is time-consuming, we believe that finetuning the E2E agents on offline real data with behavioral cloning is possible and would enhance the performance in upcoming real experiments.

Exploration performance is provided as complementary information in Figure4.4. On this task, we compare with *Active Neural SLAM* by Chaplot *et al.* (Chaplot et al. 2020b). We outperform it on the first 200 steps, making our method more

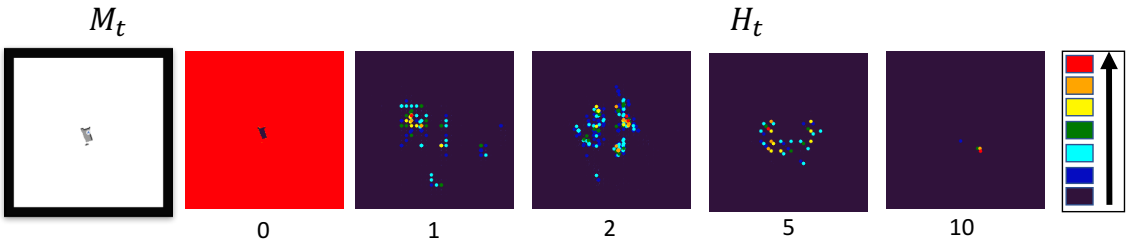


Figure 4.6 – For a given time step, we plot the predicted spatial heatmaps $\mathbf{H}_t$ for different training checkpoints, after 0,1,2,5 and 10 million updates. The lowest and highest probabilities are in dark blue and red, respectively.

robust for limited time-budget exploration. More importantly, given our main objective of optimizing sim2real performance, we outperform the state-of-the-art on domain generalization by +15% margin, as shown in Table4.4.

Qualitative results are given in Figure4.5, which shows a rollout for a single episode. We see that the agent first explores the scene, observes the goal quickly, and switches to exploitation mode. While navigating to the first goal, it observes a potential future goal and correctly maps it. Some drawbacks of the hybrid agent are presented in the detection of false positives such as the white cylinder.

In Figure4.6, we visualize, for a chosen episode in a given environment step, the evolution of the predicted heatmaps $\mathbf{H}_t$ as training evolves (hence, different checkpoints). Heatmaps correspond to high-entropy distributions at the beginning, with high uncertainty on exploration targets. As training goes on, the distribution gets narrower and peaky, making the predictions more and more certain.

4.5 Conclusion

We have extended the Multi-ON task to real environments and up to our knowledge we present the first experimental evaluation of this task in these settings. We have introduced a hybrid model, which disentangles waypoint planning and semantics, significantly decreases the sim2real gap, and outperforms E2E trained models which were the current SOTA in simulation. Future work can focus on the enhancement of the handcrafted high-level strategy, which needs to be robust to the false positive detections, a common challenge also in other navigation tasks (e.g. ObjectNav).

Another enhancement can be done on the reasoning of the high-level planner (Global Policy) by not limiting to only choosing the next way-point. We can extend its decision capacity to choose the navigation behavior that suits the

current situation. Moving toward this direction, in the next chapter, we will design another hybrid agent that leverages two local planners: a neural End-to-End planner (Reinforcement Learning (RL) recurrent agent as in chapter 3) and a classical planner that is commonly used as local analytical planner in the literature (Sethian 1996a). The planners will be ruled by a high-level policy that dynamically switches between the two planners depending on the actual situation in the real environments in order to reach the assigned target.

CHAPTER 5

LEARNING WHOM TO TRUST IN NAVIGATION: ALTERNATING BETWEEN PLANNERS

Contents

Chapter abstract

Navigation of terrestrial robots is typically addressed either with localization and mapping (SLAM) followed by classical planning on the dynamically created maps, or by machine learning (ML), often through end-to-end training with reinforcement learning (RL) or imitation learning (IL). Recently, modular designs have achieved promising results, and hybrid algorithms that combine ML with classical planning have been proposed. Existing methods implement these combinations with hand-crafted functions, which cannot fully exploit the complementary nature of the policies and the complex regularities between scene structure and planning performance.

Our work builds on the hypothesis that the strengths and weaknesses of neural planners and classical planners follow some regularities, which can be learned from training data, in particular from interactions. This is grounded on the assumption that, both, trained planners and the mapping algorithms underlying classical planning are subject to failure cases depending on the semantics of the scene and that this dependence is learnable: for instance, certain areas, objects or scene structures can be reconstructed easier than

others. We propose a hierarchical method composed of a high-level planner dynamically switching between a classical and a neural planner. We fully train all neural policies in simulation and evaluate the method in both simulation and real experiments with a LoCoBot robot, showing significant gains in performance, in particular in the real environment. We also qualitatively conjecture on the nature of data regularities exploited by the high-level planner.

The work of this chapter is the result of the internship done by Sombit Dey at Naver Labs Europe and co-supervised by Assem Sadek. It's an extension (in idea and codebase) of the work described in the last chapter. Apart from close supervision and providing conceptual input, Assem Sadek also executed the experiments with the real robot. This work has led to the following publication:

- Sombit Dey, Assem Sadek, Gianluca Monaci, Boris Chidlovskii, and Christian Wolf (2022). "Learning whom to trust in navigation: dynamically switching between classical and neural planning". In: *IROS 2023*;

5.1 Introduction

Large-scale machine learning has had a significant impact on robotics, and in particular on navigation of mobile robots, where end-to-end training in simulated 3D environments like Habitat Savva et al. 2019 and AI-Thor Kolve et al. 2017 has been proposed as an alternative to classical map and plan baselines. The potential advantages of learning to plan with high-capacity deep neural networks are the promise of complex decision functions, able to cope with large amounts of noise, sensor failure and unmodeled disturbances, and complex dependencies on scene semantics, which are difficult to design with handcrafted algorithms. This complexity comes with a price, the dependency on massive amounts of training data in the form of 3D scene models loaded into simulators. While the amount of data seen during training can be almost unlimited (modern models are trained on typically 100M — up to 7B environment steps (Partsey et al. 2022)), the main factors of variation are the number of scenes, which are limited due to the required effort of scanning physical buildings. Current datasets contain dozens or hundreds of scenes (Xia et al. 2018; Chang et al. 2018), with up to 1000 scenes for the latest HM3D dataset (Ramakrishnan et al. 2021). Lack of sufficient diversity in scenes and the sim2real gap — the difference between simulation and real environment — limit the transfer of navigation performance to real environments.

For these reasons, classical map and plan baselines (Marder-Eppstein et al. 2010; Macenski et al. 2020) are still competitive in many situations where the navigation task itself does not depend on complex high-level visual reasoning, and where maps can be estimated with sufficient reliability. In this work we ask two scientific questions: (1) are trained and classical planning strategies complementary and

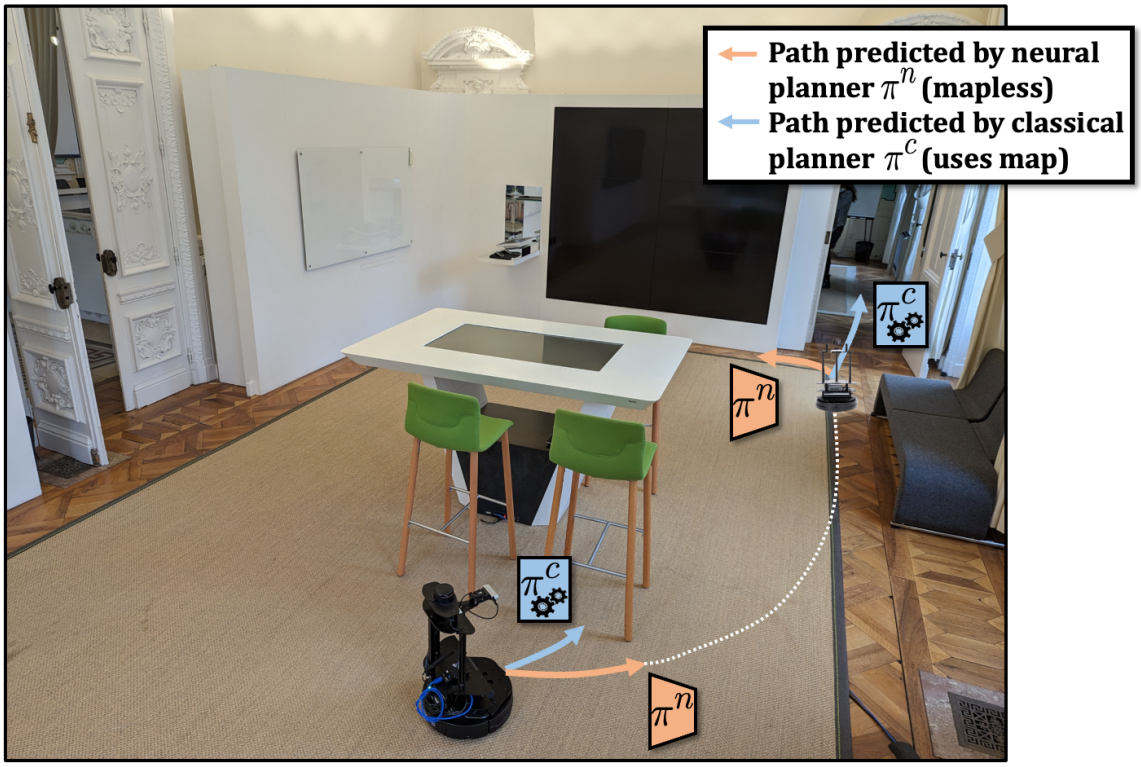


Figure 5.1 – In indoor navigation problems, we present an agent which can resort to two different strategies, a trained neural planner and a classical planner based on occupancy maps. An additional high-level governor is trained to switch between the two strategies based on learned regularities between planning performance and scene semantics, for instance that high chairs are not well reconstructed and lead to bad performance of a classical map-and-plan solution. We train in simulation and evaluate in, both, simulation and an office building using a real robot.

excel in different situations, and (2) can these different types of situations be clearly distinguished from visual observations, making it possible to exploit these regularities?

We explore these questions in a series of experiments and propose a new hybrid method combining classical and neural planning. Compared to existing hybrid solutions in the literature (Dashora et al. 2021; K. Weerakoon and A.J. Sathyamoorthy and D. Manocha 2022; A. Faust and O. Ramirez and M. Fiser and K. Oslund and A. Francis and J. Davidson and L. Tapia 2018), our method is based on a trained combination. A high-level planner, trained with RL, dynamically switches between the two alternative planning methods and learns to adapts to the situation at hand, as shown in Fig. 5.2. To this end, it receives as input features extracted from first-person images, which may be useful to exploit correlations

between scene semantics and planning performance. We also experiment with a variant which takes the high-level decision on, both, the first person input and occupancy map. The exact regularities picked up by the high-level planner may be complex, and we attempt to answer this question in the experimental part of this paper. To further motivate this approach beforehand, we mention possible scenarios: 3D scene structures difficult to reconstruct and to project into an occupancy map might be recognizable from their first person depth input, or linked to their semantic class and recognizable from the first person RGB input; 2D structures in the occupancy map harmful to classical or neural planning could be detectable directly; the trained low-level planner might be subject to biases picked up in simulation from spurious correlations, and these biases might be learnable by the high-level planner, switching over to classical planner when needed.

We claim the following contributions:

- a hybrid method switching between complementary navigation strategies based on a high-level planner trained with reinforcement learning on dense reward (geodesic distance to the goal).
- Large-scale training in 3D photorealistic simulation using complex first person RGB-D input.
- Transfer from simulation to a real environment and extensive experiments with a LoCoBot mobile robot.

5.2 Related work

Navigation with mapping and planing is the core capability of service robots since their introduction (Burgard et al. 1998). Classic navigation stacks often assume access to a pre-scanned map of the environment (Burgard et al. 1998; Marder-Eppstein et al. 2010; Macenski et al. 2020) and are composed of three main modules: mapping and localization using visual or Lidar SLAM (Thrun et al. 2005; Labbé et al. 2019), global planning with, for example, A* (Konolige 2000) or Fast Marching Method (FMM) (Sethian 1996a), and low-level local path planning to reach intermediate waypoints (Fox et al. 1997; Rösmann et al. 2015). The classical planner used in this work does not have access to the environment map where it is deployed. It uses depth images and odometry to incrementally build a 2D egocentric occupancy map and localize the agent on it, while planning is done using FMM.

End-to-End navigation directly trains an agent to predict actions from observed input, either with reinforcement learning (RL) or imitation learning (IL). Given the partial observable nature of the problem, the agent keeps latent memory, typically

through a recurrent neural network. Additional structured neural memory has been proposed, e.g. neural metric maps (Henriques et al. 2018; Beeching et al. 2020b), semantic maps (Chaplot et al. 2020a), neural topological maps (Chaplot et al. 2020d; Savinov et al. 2018; Shah et al. 2022; Beeching et al. 2020c) or implicit representations (Li et al. 2022c; Marza et al. 2023b). Recently, it has also been proposed to replace recurrence by Transformers (Vaswani et al. 2017) with self-attention over the history of observations (Fang et al. 2019; Du et al. 2021; Chen et al. 2022a; Reed et al. 2022).

Modular and hybrid navigation Modular approaches decompose planning hierarchically. While the option framework R.S. Sutton 1999 provides a generic solution in the context of planning with RL, specific solutions have been proposed for navigation. Typically, waypoints are proposed by a high-level (HL) planner, and then followed by a low-level (LL) planner. In one line of work, the HL planner is a trained model, which triggers actions by the LL planner, which is either also trained (Chaplot et al. 2020b) or classically based on shortest path calculations on a map (Chaplot et al. 2020a; Sadek et al. 2022a) or optimal control (Bansal et al. 2019). In the complementary line of work, the HL planner is based on classical optimization based algorithms, e.g. Probabilistic Roadmaps (A. Faust and O. Ramirez and M. Fiser and K. Oslund and A. Francis and J. Davidson and L. Tapia 2018) or shortest-path calculations in a high-level graph (Beeching et al. 2022). Both of these solutions defer point-to-point navigation to a LL planner trained with RL.

Hybrid methods combine classical planning with learned planning. Some of the modular approaches mentioned above can be considered to be hybrid, but there exist hybrid approaches in the literature which combine different planners more tightly and in a less modular way. In K. Weerakoon and A.J. Sathyamoorthy and D. Manocha 2022 and similarly in Dashora et al. 2021, a planner trained with RL generates trajectories, which are used to generate a cost-map used by a classical planner. In Zeng et al. 2020, a neural planner generates UAV trajectories which are then used by a model-predictive control as support for optimization. *Neural-A** learns a model predicting a cost-map for planning with a differentiable version of *A**, backpropagating a supervised loss through it (Yonetani et al. 2021). Similarly, *Cognitive Mapping and Planning* (Gupta et al. 2017a) learns a mapping function by backpropagating through a differentiable planner, in the form of *Value Iteration Networks* (Tamar et al. 2017). In Beeching et al. 2020c, a graph-network is imbued with inductive bias for planning with the *Bellman-Ford* algorithm.

All these existing solutions combine planners with different but handcrafted designs. In contrast, our method dynamically switches between types of planners with a trained model. Similar to our approach, in Kastner et al. 2022, a HL planner is trained on a schematic simulation to switch between a classic model-based planner and a learned planner for dynamic obstacle avoidance. However, this

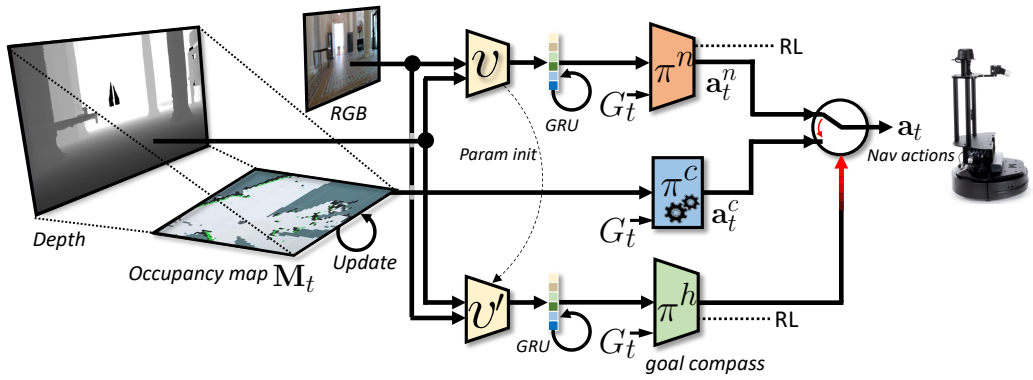


Figure 5.2 – We distribute navigation decisions over two different planners: a trained low-level planner π^n takes RGB-D first-person input, and a classical planner π^c takes a metric occupancy map $\mathbf{M}_t$ as input. A high-level planner π^h exploits regularities between scene elements and planning performance and learns to take a binary decision between these two planners, based only on first person inputs. The hidden state of the recurrent policy π^n is updated even when the classical planner is used.

work considers a simple 2D set-up where all planners have access to the full map of the environment, perfect 360° Lidar scans and exact obstacle positions, while our methods only access noisy first-person images in a realistic 3D simulator and a real robot. Also, the HL planner in (Kastner et al. 2022) tackles the considerably simpler task of selecting one of two options tailored to two different situations: efficiently navigate to a goal or avoid dynamic obstacles. In contrast, our HL planner has to learn to exploit subtle correlations between scene structure and semantics and planning performance to combine LL algorithms that are designed for the same task with comparable performance.

5.3 Learning to choose planners

We address the *PointGoal* task where an agent receives a visual observation $\mathbf{o}_t \in \mathbb{R}^{4\times H\times W}$ (an RGB-D image) and a Euclidean goal compass vector G_t at each time step t and must take actions $\mathbf{a}_t$ in a discrete action space $\Lambda = \{$`MOVE_FORWARD 25cm`, `TURN_LEFT` 10°, `TURN_RIGHT` 10° and `STOP`$\}$. The `STOP` action terminates the episode successfully if the agent is within 0.2m of the goal, or unsuccessfully if not.

As shown in Fig. 5.2, our method takes decisions at each time t on whether to choose an action predicted by a neural planner π^n or a classical planner π^c. We will first introduce each low-level planner and then the high-level governor π^h. In what follows, superscripts $.^n$, $.^c$ and $.^h$ do not take numerical values but rather

denote choices between the neural, classical, or high-level planner, respectively. Network architectures of all trainable functions will be provided in section 5.3.4.

5.3.1 The neural planner

The neural planner π^n is trained in simulation to directly predict navigation actions $\mathbf{a}_t^n \in \Lambda$ from visual input $\mathbf{o}_t$. It sequentially builds a representation $\mathbf{h}_t$ from the sequence $\{\mathbf{o}_{t'}\}_{t'<t}$ of visual first-person observations, and then predicts a distribution over actions,

$$\mathbf{h}_t = f^n(\mathbf{h}_{t-1}, v(\mathbf{o}_t), \mathbf{a}_{t-1}^n), \tag{5.1}$$

$$p(\mathbf{a}_t^n) = \pi^n(\mathbf{h}_t, G_t), \tag{5.2}$$

where f^n is the update function of a recurrent GRU network, with gates omitted from the notation for convenience; v is a visual encoder, i.e. a trained ResNet extracting features from observations.

We train this planner end-to-end with PPO (Schulman et al. 2017b) with the reward definition from (Chattopadhyay et al. 2021),

$$r_t = \mathrm{K} \cdot \mathbb{I}_{\text{success}} - \Delta_t^{\text{Geo}} - \lambda, \tag{5.3}$$

where K=2.5, Δ_t^{Geo} is the gain in geodesic distance to the goal, and slack cost λ=0.01 encourages efficiency.

5.3.2 The classical planner

Numerous algorithms and implementations exist for planning based on dynamically estimated maps. We use the map and plan baseline approach proposed in Gupta et al. 2017a, which maintains an egocentric metric occupancy map $\mathbf{M}_t \in [0,1]^{N \times M}$, called "*Egomap*", over time by first inversely projecting the depth channel of the visual observation $\mathbf{o}_t$ (using intrinsics of the calibrated camera) and then pooling the resulting point cloud to the ground, resulting in a local bird's-eye-view map for this observation. Consecutive maps are aligned with odometry and integrated with max pooling, as in Chaplot et al. 2020a. Planning is performed on this map using FMM (Sethian 1996a).

The action space of a planner based on shortest path calculations is inherently tied to the underlying representation it uses for planning, which in our case is the resolution of the metric map $\mathbf{M}_t$: a navigation action is a part of a path in the graph structure of the map $\mathbf{M}_t$, i.e. the choice of an edge between two nodes. However, to align the action spaces of the two complementary navigation strategies, we chose to translate these predictions into actions taken from the discrete alphabet Λ of the downstream navigation task. This not only facilitates

the design of the high-level planner, but also allows to run both low-level planners simultaneously and maintain their respective states, as will be discussed in the next section. This translation is done with a well-known, publicly available map and plan baseline [1].

5.3.3 The high-level planner

The high-level planner π^h takes a binary decision $d_t \in \{0, 1\}$ on the choice of planners, such that the final navigation action $\mathbf{a}_t$ is given as

$$\mathbf{a}_t = d_t \mathbf{a}_t^n + (1 - d_t)\mathbf{a}_t^c. \tag{5.4}$$

The planner is implemented as a recurrent policy, which maintains a hidden state $\mathbf{r}_t$ with a GRU, denoted as f^h, and which takes as input features extracted from the first person input $\mathbf{o}_t$,

$$\mathbf{r}_t = f^h(\mathbf{r}_{t-1}, v'(\mathbf{o}_t), d_{t-1}), \tag{5.5}$$

$$p(d_t) = \pi^h(\mathbf{r}_t, G_t). \tag{5.6}$$

The feature extractor v' has the same architecture as v in Eq. (5.1), see Section 5.3.4, and it is fine-tuned from the trained version of v.

The high-level planner is trained with PPO end-to-end, jointly with the encoder v', with a reward used for the neural low-level planner in Eq. (5.3). We train with vectorized environments and maintain 12 agents per batch. The neural planner π^n is operated in parallel to the classical one π^c, and its hidden state $\mathbf{h}_t$ is updated with Eq. (5.1), even if it has *not* been chosen by the high-level planner, by providing it with the action taken by π^c. This allows the neural planner to maintain a spatial internal representation during navigation consistent with what it experienced during training, regardless of its actual use in the hybrid setting. Two key design choices were necessary to make this possible: the alignment of the action spaces of the two planners (see Section 5.3.2), and the possibility of updating the internal state $\mathbf{h}$ with an action different from the one predicted by the agent π^n itself. The latter is enabled through sampling actions stochastically from the predicted discrete distribution $p(\mathbf{a}_t^n)$ during training; this leads the agent to update its internal (spatial) representation of the scene not based on its previously predicted action, but on the effectively performed previous action $\mathbf{a}_{t-1}$ input to the policy in Eq. (5.1).

We add two remarks here. First, during training, we sample from the predicted distribution $p(\mathbf{a}_t^n)$, which is different from the distribution of frequent action choices by the competing classical planner π^c — we chose to ignore this difference. Second, as in large part of the literature, we train without actuation noise, i.e. the

1. https://github.com/s-gupta/map-plan-baseline

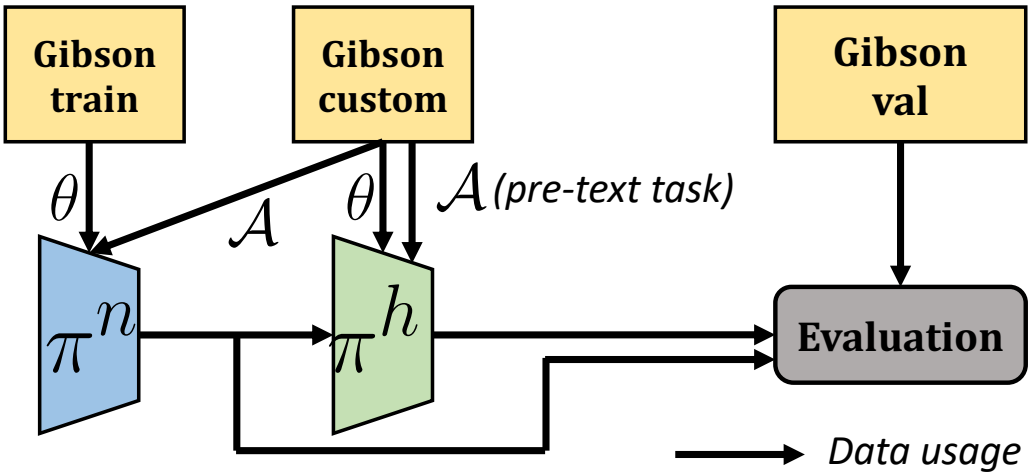


Figure 5.3 – **Training pipeline and data splits:** training the hybrid planner π^h requires a custom data split, as training needs to be performed on data which have not been seen during training of the low-level neural planner π^n. $\rightarrow\theta$ indicates training network parameters with SGD training; $\rightarrow\mathcal{A}$ indicates architecture optimization (manual, through "grad student descent"). We accepted some overlap in optimizing hyper-parameters, see the text. However, *evaluation was performed only on scenes unseen during training and hyper-parameter optimization*.

previous action $\mathbf{a}_{t-1}$ provides the exact odometry information during training. Previous work (Kadian et al. 2020) shows that doing so improves performance in real-world experiments. At testing the learned policy is directly transferred to the noisy setting.

5.3.4 Network architectures

The visual encoders v and v' are ResNet18 (He et al. 2016) architectures. The recurrent policies are composed of GRUs f^n and f^h with 2 layers and hidden states of size 512. Previous actions $\mathbf{a}^n_{t-1}$ and goal compass vector G_t are encoded with learned embeddings of size 32.

5.4 Experimental Results

Experimental setup all training was performed in simulation only with the Habitat simulator (Savva et al. 2019) and scenes from the *Gibson* dataset (Xia et al. 2018), which contains 3.6M episodes over 72 different scenes for training, and 994 navigation episodes over 14 scenes for validation. We test the system in both simulation, with additional noise, and a real physical robot, in particular a

LoCoBot robot (*LoCoBot: An Open Source Low Cost Robot* 2017) [ ] equipped with an Intel *RealSense* D435i RGB-D camera and a single-ray Lidar of type *RPLIDAR A2M8*.

The agent receives an RGB-D observation of size 160×120 pixels at each step, which in simulation matches the extrinsic and intrinsic parameters (position, the field of view and aspect ratio) of the onboard camera on the LoCoBot. It also gets a goal compass vector in the form of the Euclidean distance and direction, provided as privileged information in the simulator, and by the robot's position estimation system in experiments with the real physical robot. In the robot experiments this is done using the default `ROS` implementation of the Adaptive Monte-Carlo Localization algorithm (Thrun et al. 2005), which is based on RTAB-Map, a 2D metric representation (Labbé et al. 2019) generated from Lidar input. The Lidar is only used to localize the robot, while sensing, mapping and planning are based solely on RGB-D camera input.

Simulator settings we removed the possibility of the robot to slide across the walls (*sliding OFF*). This makes the PointGoal navigation task more challenging for both low-level planners, and previous work (Kadian et al. 2020) finds this setting crucial for real-world deployment. We configure the Habitat simulator and adjusted it to the properties of the physical robot (LoCoBot) and its sensors: FOV of 56° camera, frame size of 160×120 and a compatible camera position. For the experiments which involved evaluation in simulation, i.e. Tables 5.1, 5.2 and 5.3, we used a second simulator configuration which is compatible with the prior work (Wani et al. 2020; Marza et al. 2022). It includes a FOV of 79° and camera frames of size 256×256.

Data splits and training pipeline as usually done in the relevant literature, we report results on the validation set of the Gibson dataset, as the test set is not available. However, to obtain optimal performance, this requires additional splits for validating the different models (hyperparameter optimization). In our case, differently from the classical settings, we require additional splits due to the fact that the high-level planner is trained on output of the neural low-level planner. Therefore, the high-level planner π^h needs to be trained on data different from training π^n, in order to avoid a potential bias of π^h trained on an overconfident π^n overfitting on its training environment and leading to skewed decisions. We therefore introduced an additional dataset split called *Gibson-custom* which consists of 1036 episodes over 14 unused scenes selected from the full Gibson dataset.

Figure 5.3 illustrates the training pipeline. The neural low-level planner π^n is trained on the Gibson training set. The high-level planner is trained on the custom split, and the Gibson validation set is used to report results. The hyper-parameters (network architectures $\mathcal{A}$) of the low-level planner π^n have been optimized on *Gibson-custom*. In other words, we accepted a small possibility of training π^h on

Agent	Input	Train-$\mathcal{N}$	Test-$\mathcal{N}$	Succ.	SPL
Neural	RGB-D	✗	✗	90.94	77.14
Neural	RGB-D	✗	Redwood+	87.87	74.21
Neural	RGB-D	Redwood+	Redwood+	89.24	75.92
Classical	Egomap	N/A	✗	87.93	79.69
Classical	Egomap	N/A	Redwood	78.67	72.17

Table 5.1 – Performance of different low-level planners in simulation (Gibson-val), where $\mathcal{N}$ is the noise model. The table shows how the difference between Redwood and Redwood+ impacts the neural planner.

Agent	Success	SPL
Classical only (π^c)	78.67	72.17
Neural only (π^n)	89.24	75.92
Random HL-decisions	88.88±1.4	73.78±1.1
Hybrid (Ours)	90.64	75.62

Table 5.2 – **Performance of the hybrid method in simulation**, tested with Redwood+ Noise.

— Input to π^h —		Success	SPL
1^{st} person	Egomap*		
RGB-D	✗	90.64	75.62
RGB-D	✓	90.85	75.78
✗	✓	89.03	74.87

Table 5.3 – **Impact of the privileged map information** on the high-level planner: simulation with Noise on Gibson-val.

overconfident decisions based on validation overfit, but we judged this risk to be small. To work around the requirement of one more data split to optimize the hyper-parameters of the high-level planner, we optimized them using a proxy task, namely exploration. More precisely, we use the network architecture of the high-level planner introduced in Chapter 4 (Sadek et al. 2022a). This planner provides high-level decisions of different nature, waypoint coordinates followed by a low-level planner, and we adapted its later layers to take binary decisions instead. These decisions did not interfere with the soundness of the evaluation protocol: *all evaluation was performed only on scenes unseen during training or hyper-parameter optimization*.

Agent	Success	SPL	SPL^{Succ}
Classical only (π^c)	33.33	27.19	81.57
Neural only (π^n)	100.00	58.55	58.55
Hybrid (Ours)	100.00	72.50	72.50

Table 5.4 – **Performance of the hybrid method in the real environment**: A LoCoBot in a real classical European office building (the "Chateau" of Naver Labs Europe), on 12 test episodes. SPL^{Succ} indicates the SPL metric only for the episodes which were succeeded.

5.4.1 Quantitative Results

Performance of the low-level planners we evaluated the two low-level planners in simulation and report results in Table 5.1. We explored different noise types on the depth observation, which is used, both, as input to the neural planner and to generate the Egomap for the classical planner. Redwood noise is classically used in evaluation of navigation (Anderson et al. 2018a), and we also explored an additional variant which we call "*Redwood+*" in Table 5.1. It is motivated by the observation that in the standard Habitat implementation of the depth noise model, a depth `D` above a given threshold `T` was set to zero [2], i.e. `if(D>T)D=0`, which is the inverse behavior of the noiseless setting, which truncates depth, ie. `if(D>T)D=T`. We argue that this extremely strong discrepancy does not fall into the category of noise but rather to a change in the nature of the sensor (it corresponds to the behavior of certain depth sensors like Kinect), degrades transfer and does not allow a sound evaluation; we therefore replaced this zeroing version with the standard truncating variant. This difference mostly has an impact on the neural planner, not the classical one.

As we can see in Table 5.1, the planners perform similarly in the noiseless environment. However, the classical planner's performance drops significantly in the noisy environment, due to a degraded quality of the Egomap on which planning is performed. The impact of noise on the neural planner is less pronounced.

Hybrid planning in simulation Table 5.2 compares performances of the low-level planners with the proposed hybrid planner. The hybrid planner outperforms both low-level variants in the Success rate, and also outperforms the baseline random high-level decisions. This version of the HL-planner takes as input the first-person RGB-D observation and thus exploits regularities between the currently observed scene structure and low-level planning performance.

We also explored whether there exist correlations between the 2D structure of the occupancy map and performance of the two low-level planners and a

2. https://github.com/facebookresearch/habitat-sim/blob/d3d150c62f7d47c4350dd64d798017b2f47e66a9/habitat_sim/sensors/noise_models/redwood_depth_noise_model.py#L73

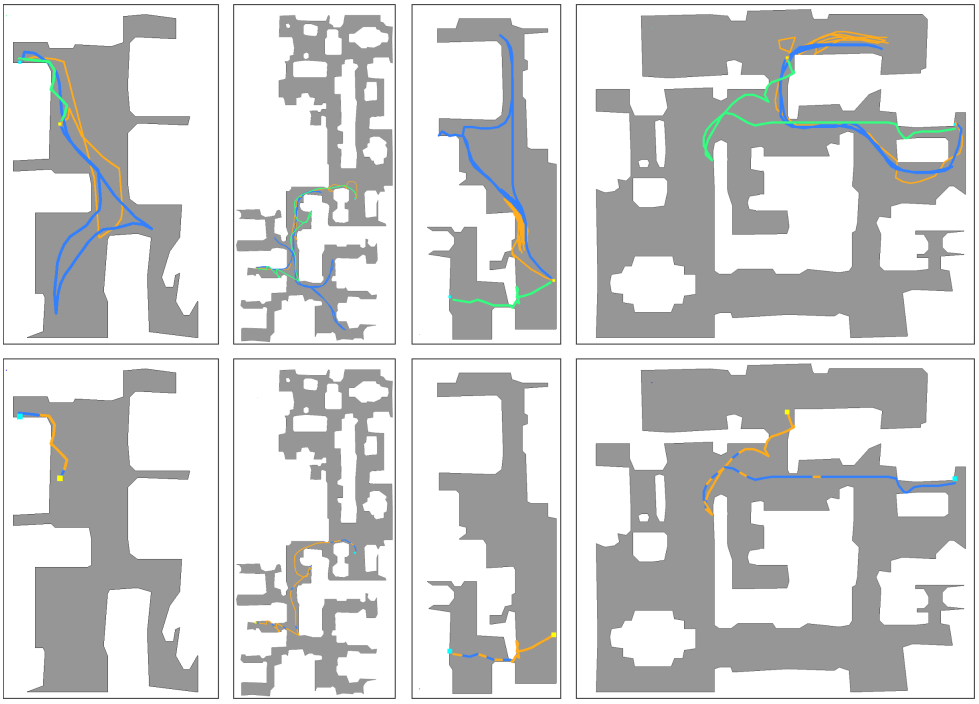


Figure 5.4 – **Rollouts of four episodes in different environments**: the robot starts at ■ and has to reach the goal position ■. **Top row**: comparing trajectories taken by the trained neural planner, classical planner and our hybrid planner. **Bottom row**: each step of the hybrid planner path in the top row is colored with the chosen low-level planner, neural or classical.

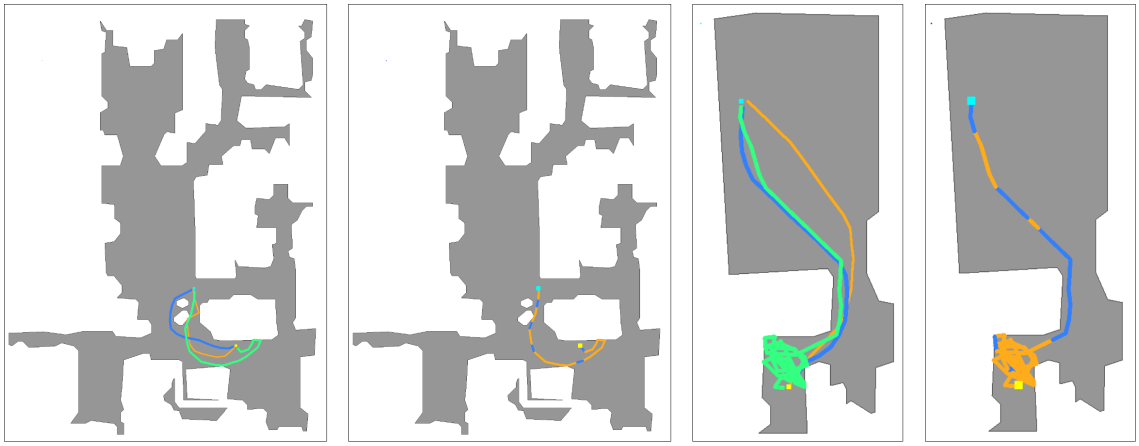


Figure 5.5 – **Failure cases**: Two examples where the hybrid planner perform worse than the neural and classical planners.

high-level planner on this input, additional to first-person input. As a proof of concept, and to minimize the impact of noise and purely focus on scene structure, we performed this experiment with a noiseless Egomap* generated through

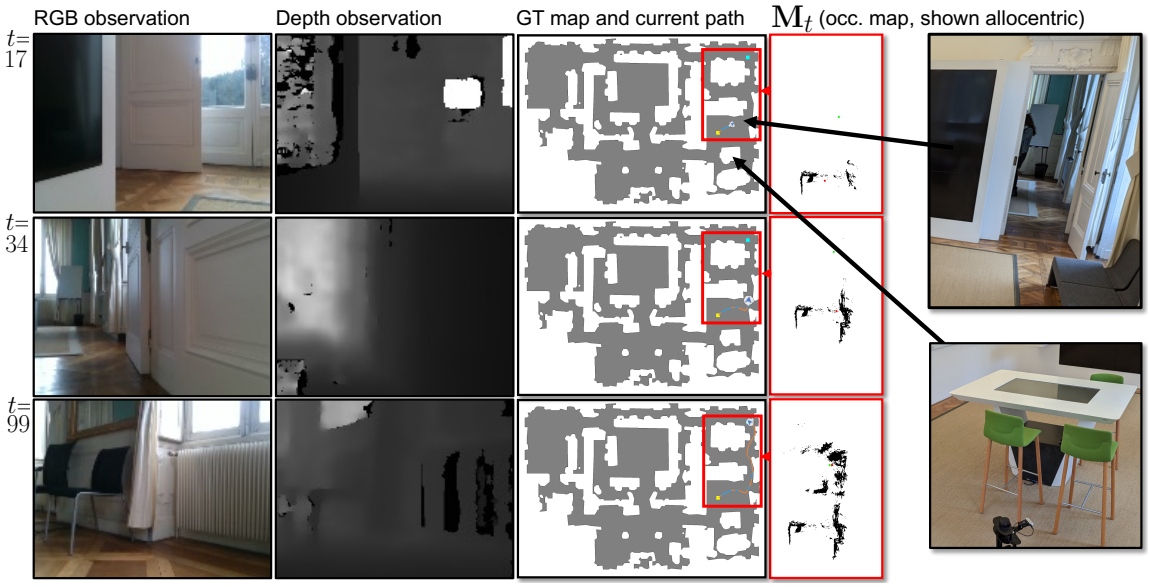


Figure 5.6 – A rollout of an episode, showing inputs and representations. We overlay the current path over the ground-truth (GT) map, color coding neural steps and classical steps. The robot starts at ■ and has to reach the goal position ■. For better comparability, the Egomap M_t is shown here not as an Egomap but in an allocentric frame. The big black arrows indicate parts of the map corresponding to the scene shown in Figure 5.1.

privileged information in the simulator. Results in Table 5.3 show that the impact of the scene structure is minimal.

Experiments with a real robot we carried out 12 episodes with the LoCoBot in Naver Labs Europe office building, the "Chateau" (see Figure 5.1) with multiple rooms and challenging situations, like thick carpets and multiple big windows that pose problems to the onboard depth sensors. We report results in Table 5.4 shows that the hybrid solution outperforms both individual standalone low-level planners significantly.

The low performance of the classical planner is explained by the fact that the maps it produces are noisy. One particular aspect we can single out is the choice of map integration over time as in Chaplot et al. 2020a, which uses max pooling to combine the bird's-eye-view estimate of the latest observation with the current global bird's-eye-view map. This choice is simple to implement but not as robust as state-of-the art Lidar based solutions like RTAB-Map, that feature a sophisticate probabilistic model and loop closure. Our choice is motivated by the objective to minimize the algorithmic sim2real gap of the two representations: the current state-of-the-art mapping solutions are difficult to integrate into a simulator like Habitat. The goal of these experiences is not to achieve state-of-the-

art performance in planning, but to study the possibility of learning regularities in planning performance.

Another reason of the low classical planner's performance is the lack of high-level reasoning in case of missing information. The planning algorithm assumes that any unobserved area in the map is navigable, it corrects these estimates when a new observation becomes available and re-plans. This leads to backtracking and long trajectories. In contrast, the neural planner takes decisions not based on a 2D occupancy map but on first person input, which provides better cues on dead ends. It can also learn higher-level visual reasoning from a large amount of environment interactions and can avoid situations where backtracking would be needed otherwise. To quantify this behavior, in Table 5.4 we also provide an additional metric, SPL^{Succ}, which corresponds to the SPL metric only for the successful episodes by the respective planner. This metric is high for the classical planner, which is efficient in cases where it does not get lost in local minima and requires extensive backtracking, leading to exceeding the maximum number of steps the task allows (=500).

The hybrid planner achieves the same 100% success rate as the standalone neural planner, but with a better SPL metric (72.50 instead of 58.55), which indicates that it is more efficient. The neural planner indeed spends more time exploring, which makes it more robust than the classical planner in certain situations but can also be harmful in others. The hybrid planner manages to combine both advantages by dynamically switching between them.

5.4.2 Qualitative Results

Sample trajectories in simulation Figure 5.4 shows four episodes in different environments. The top row of pictures compares the behaviour of the neural, classical and hybrid planners. The bottom row shows the decisions taken by the hybrid planner in each episode. Our hybrid solution combines the low-level planners to solve long-horizon navigation tasks by exploring complex unknown environments, maneuvering in narrow spaces and efficiently reaching the goal. The hybrid planner starts episodes by using mainly the neural planner, which has better exploration capabilities. The neural planner is the preferred choice when the robot has to pass through a narrow corridor, as in the 3rd example. The classical planner is frequently employed towards the end of the episodes, when the path to the goal is clearer, as in the 4th example.

While the proposed hybrid approach has on average better navigation performance, this strategy can occasionally perform worse than the individual low-level planners. Figure 5.5 shows two typical failure cases: on the left, the hybrid planner selects the neural planner to start the episode, but it explores the wrong side of the scene, so the hybrid planner has to take a long detour to reach the goal. On the right, a more rare but dramatic failure case occurs when the hybrid planner,

driven mainly by the neural planner, gets lost and starts to frenetically explore the environment. We conjecture that this might be due to few actions executed by the classical planner that put the neural planner in an unstable state.

Example robot rollout Figure 5.6 shows an example episode rollout for three time instants $t = 17, 34, 99$, including the first person input $\mathbf{o}_t$ (RGB and depth), the GT map with the overlaid path and color coded high-level decisions, as well as the occupancy Egomap $\mathbf{M}_t$ — which we display in an allocentric way (and not as an egocentric map) for better comparability with the GT map. During the episode, we can notice that the HL planner relies more on the neural planner, which is more capable of navigating through narrow spaces encountered in this episode, except when the robot deviates from the most promising direction (towards the door) and the classical planner is chosen. Indeed, until $t = 17$, the classical planner dominate the HL decisions, and guides the robot towards the goal. After a segment where the neural planner is chosen, at $t = 34$, the classical planner takes over again to readjust the direction of the robot; then, until the end of the episode, the HL planner switches again to the neural one to traverse the final narrow passage.

5.5 Conclusion

We have presented a hybrid method for navigation in real environments, which combines advantages of classical planning methods based on occupancy maps and shortest path computations with the power of neural methods trained in large-scale 3D photo-realistic simulations. We used RL to train a neural HL planner to dynamically switch between the two different LL planners and showed that they are complementary. Our experiments provide evidence for correlations between the observed scene structure and the difference in planning performance between the two LL planners, which are exploited by the hybrid solution. We have evaluated the proposed method in, both, simulation and a robot in a real office building, showing that the learned regularities transfer well. Future work will focus on learning the high-level decision on real data in the form of offline trajectories captured with a physical robot.

CHAPTER 6

CONCLUSION

Contents

6.1 Summary of Contributions

In this manuscript, we described different contributions that leverage classical robotics and deep learning-based agents for robot navigation in real environments. Our contributions can be summarized as follows:

Sim2Real generalization capacity of end-to-end neural agents In two real physical environments, we conducted an evaluation of three variants of end-to-end neural agents. The purpose is to benchmark the generalization capacities of end-to-end agent models, deployed on physical robots. We showed that for the *PointNav* task, fine-tuning an agent on the simulation twin of the targeted real scene reduces the sim2real gap applying any sim2real adaptation technique. The agent has already been trained on various publicly available scenes. Currently, scanning buildings can be done in a very short time with handheld devices (e.g. matterport devices - *Matterport* (2020) or smartphones equipped with LIDAR), and produces high-quality photorealistic scenes. We encourage the idea of fine-tuning end-to-end agents, deployed on commercialized robots, on scanned buildings. This can enhance the current performance of the robot to realize its assigned task in the targeted real environment.

Interpreting neural agents in real settings We realized an in-depth analysis to visualize and interpret the sensor usage of the neural agents during real scene navigation. The analysis helped in understanding the visual reasoning of the agent and how the agent understands the usefulness of each input sensory signal. We visualized the backpropagated gradients with respect to the input. We found that the neural network attends to regions in visual observation where the agent is going to act next, in another word, to where it is looking at (e.g., looking right to move right). More interestingly, in the case of an encountered obstacle, the average

gradients of depth sensor are higher than in the case of no obstacle. Moreover, the neural network attends to the region of the blocking obstacle. On the other side, the visual sensors had negligible attention (possibly, unused) when the agent reached the goal. We conjecture that the agent currently "understands" (in some sense) the usefulness of the odometery and goal sensors over visual sensors. The current interpretation study indicates that the agent indeed puts an attention on the visual information when needed.

Hybrid agent for multi objects reasoning tasks We have introduced a novel hybrid agent for the *MultiON* task, designed to be deployed in real scenarios. The design choice of the hybrid agent disentangles semantics reasoning, exploration reasoning and waypoint planning. The disentanglement decreases the sim2real gap for trained agents and increases the interpretability of the agent. We did a comparison between the hybrid agent and the end-to-end neural agents trained, whether with and without privileged information. Despite the state-of-the-art performance of end-to-end agents in simulation, the hybrid agent outperforms them in real environments. In Simulation, the hybrid agent has the best performance with real robot configurations.

Novel EgoMap exploration policy We introduced a novel exploration module for the hybrid agent trained with Deep Reinforcement Learning (RL). The policy doesn't process any visual observations, only ego-centric maps (EgoMap), by a metric Simultaneous Localization and Mapping (SLAM) is running in parallel. The usage of a metric map only solution for exploration minimized the sim2real gap. The exploration policy outperforms the state-of-the-art methods on new unseen environments.

Hybrid agent for efficient planning We leverage the availability of two different decision-making approaches: symbolic and neural planners. We introduced another high-level planner that dynamically switches between the two planners. The switching mechanism is based on the high-level planner prediction on "whom to trust" to do planning at the current time step. We showed that adding two different planners simultaneously didn't produce any "self-destructive" behavior that can elongate the path to reach the end goal. On the contrary, when we compared with the two baselines: standalone symbolic and neural planners, the new agent outperforms both. We tested the hybrid approach on *PointNav* task in both setups: simulation and real.

6.2 Perspectives for Future Work

Semantics' aware agent, robust to false detections The high-level policy we introduced in Chapter 4 (Ego-Map Exploration) is a semantic-agnostic planner that proposes a way-point for navigation, it considers neither the semantic layout of the environment nor the current semantic target. Although, our high-level policy has an exploration performance in scene coverage, in addition to its contribution to our hybrid agent performance in the *MultiON* task, we consider the lack of semantic awareness to be a limitation. Some of the existing hybrid agent works introduced semantic inductive biases (Chaplot et al. 2020a; Ramakrishnan et al. 2022; Zhai et al. 2022) in their high-level policies and offer promising results on single object navigation (*ObjectNav*) in simulation and in real (Gervet et al. 2023). We believe that these methods, if adapted to the multi-objects variants, can maintain good results.

But a challenge that we share with these works are the false positive detections in the environment, which are accumulated on the map once they are projected. These detections act as false guidance for the high-level planner, hence the whole navigation episode is prone to fail drastically. Most of the semantic projection techniques used in hybrid agent rely on off-the-shelves 2D semantic segmentation models. These models don't consider any 3D structure of the model in the environment and treat different observations as independent samples of the environment (refer to Passive AI in introduction - Chapter 1). Future directions in semantic mapping should consider these previous detections. An agent should be able to "rethink and readjust" its previous detections with the current one and only map the object once it has high certainty. One possible direction is to use transformers to correlate and attend between the history of detection and mapped positions. Transformers have already been used in similar contexts. They have been used as an alternative for hand-designed "non-maximum suppression" in object detections (Carion et al. 2020) and as a data association between frames in object tracking (Meinhardt et al. 2022).

Implicit neural representations for hybrid agents Although, in this thesis, we favor explicit representations of the environment, implicit neural representations have recently shown some advancements in novel view synthesis. In 2020, Neural Radiance Fields (NeRF) (Mildenhall et al. 2020) have been introduced as a neural-based solution for the novel view synthesis problem. A scene is represented in a fully connected neural network, whose input is the spatial location and viewing direction and outputs the RGB view with the appearance (emitted radiance). Additionally, Yen-Chen et al. (2021) build a framework, called iNeRF that performs pose estimation by "inverting" NeRF. Implicit neural representation opens the door for new advancements in robotics. Adamkiewicz et al. (2022) propose an algorithm for navigating a drone through an indoor 3D environment represented

as a NeRF (trained offline) using RGB-only for localization. The drone has to avoid obstacles to reach a specific goal using the NeRF representations. Li et al. (2022c) uses implicit representations to encode the distance between any position in the scene and a navigable goal. Li et al. (2021) uses implicit representations to tackle robot manipulation tasks that involve rigid bodies and fluids. They show that a dynamics model, constructed from learned implicit representations space, enables visuomotor control for manipulations.

In mobile navigation, Marza et al. (2023b) proposes two implicit representations as inductive biases in autonomous agents navigating to find multiple objects (*MultiON*). A semantic finder function the predicts the position of a previously seen objects and a structural representation function that predicts occupancy and exploration. They succeeded to learn these functions dynamically during the navigation episodes. Moreover, Sucar et al. (2021) bring out an implicit SLAM approach: a dense real-time SLAM that uses implicit neural scene representation. It's potentially capable of jointly optimizing a full 3D map alongside the camera poses. An extension of this work has been introduced in semanticNeRF (Zhi et al. 2021) that jointly encodes semantic with appearance and geometry. Therefore, a 2D semantic labels can be attained accurately and completely using few annotations. We suggest that SemanticNeRF can cooperate inside hybrid modular agents with explicit mapped objects to add a "verification layer" to minimize the possibility of false objects mapping. Marza et al. (2023a) has already moved into this direction by using trained SemanticNeRFs with exploration embodied agent. The agent plans with a symbolic planner on the Bird Eye-View map, estimated from the NeRFs.

BIBLIOGRAPHY

A. Faust and O. Ramirez and M. Fiser and K. Oslund and A. Francis and J. Davidson and L. Tapia (2018). "PRM-RL: Long-range Robotic Navigation Tasks by Combining Reinforcement Learning and Sampling-based Planning". In: *ICRA* (cit. on pp. 93, 95).

Adamkiewicz, Michal, Timothy Chen, Adam Caccavale, Rachel Gardner, Preston Culbertson, Jeannette Bohg, and Mac Schwager (2022). "Vision-Only Robot Navigation in a Neural Radiance World". In: *IEEE Robotics and Automation Letters* (cit. on pp. 79, 109).

Anderson, Peter, Angel Chang, Devendra Singh Chaplot, Alexey Dosovitskiy, Saurabh Gupta, Vladlen Koltun, Jana Kosecka, Jitendra Malik, Roozbeh Mottaghi, Manolis Savva, and Amir R. Zamir (2018a). *On Evaluation of Embodied Navigation Agents*. arXiv: 1807.06757 [cs.AI] (cit. on pp. 15, 16, 80, 102).

Anderson, Peter, Ayush Shrivastava, Joanne Truong, Arjun Majumdar, Devi Parikh, Dhruv Batra, and Stefan Lee (2020). *Sim-to-Real Transfer for Vision-and-Language Navigation*. arXiv: 2011.03807 [cs.CV] (cit. on p. 5).

Anderson, Peter, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel (2018b). "Vision-and-Language Navigation: Interpreting Visually-Grounded Navigation Instructions in Real Environments". In: *IEEE Conference on Computer Vision and Pattern Recognition,CVPR*, pp. 3674–3683 (cit. on p. 63).

Angeli, Adrien, Stephane Doncieux, Jean-Arcady Meyer, and David Filliat (2009). "Visual topological SLAM and global localization". In: pp. 4300–4305 (cit. on p. 32).

Antol, Stanislaw, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C. Lawrence Zitnick, and Devi Parikh (2015). "VQA: Visual Question Answering". In: *International Conference on Computer Vision (ICCV)* (cit. on p. 3).

Armeni, Iro, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese (2016). "3D Semantic Parsing of Large-Scale Indoor Spaces". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1534–1543 (cit. on p. 40).

Bansal, Somil, Varun Tolani, Saurabh Gupta, Jitendra Malik, and Claire Tomlin (2019). *Combining Optimal Control and Learning for Visual Navigation in Novel Environments*. arXiv: 1903.02531 [cs.RO] (cit. on pp. x, 57, 58, 95).

Batra, Dhruv, Aaron Gokaslan, Aniruddha Kembhavi, Oleksandr Maksymets, Roozbeh Mottaghi, Manolis Savva, Alexander Toshev, and Erik Wijmans (2020). *ObjectNav Revisited: On Evaluation of Embodied Agents Navigating to Objects*. arXiv: 2006.13171 [cs.CV] (cit. on pp. 17, 60).

Beeching, E., J. Dibangoye, O. Simonin, and C. Wolf (2020a). *EgoMap: Projective mapping and structured egocentric memory for Deep RL* (cit. on pp. 18, 32, 44, 60).

Beeching, E., M. Peter, P. Marcotte, J. Dibangoye, O. Simonin, J. Romoff, and C. Wolf (2022). "Graph augmented Deep Reinforcement Learning in the GameRLand3D environment". In: *AAAI Workshop on Reinforcement Learning in Games* (cit. on p. 95).

Beeching, E., C. Wolf, J. Dibangoye, and O. Simonin (2020b). "Deep Reinforcement Learning on a Budget: 3D Control and Reasoning Without a Supercomputer". In: *ICPR* (cit. on pp. 60, 76, 95).

Beeching, Edward, Jilles Dibangoye, Olivier Simonin, and Christian Wolf (2020c). "Learning to plan with uncertain topological maps." In: *European Conference on Computer Vision* (cit. on pp. 41, 44, 51, 76, 79, 81, 95).

Bellman, Richard (1957). "A Markovian Decision Process". In: *Indiana Univ. Math. J.* 6 (4), pp. 679–684 (cit. on pp. 4, 23).

Bender, Douglas J. and Alan J. Laub (1987). "The linear-quadratic optimal regulator for descriptor systems: Discrete-time case". In: *Automatica* 23.1, pp. 71–85. URL: https://www.sciencedirect.com/science/article/pii/0005109887901191 (cit. on p. 58).

Blackwell, David (1947). "Conditional expectation and unbiased sequential estimation". In: *The Annals of Mathematical Statistics*, pp. 105–110 (cit. on p. 38).

Bojarski, Mariusz, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, and Urs Muller (2017). *Explaining How a Deep Neural Network Trained with End-to-End Learning Steers a Car*. arXiv: 1704.07911 [cs.CV] (cit. on p. 42).

Burgard, Wolfram, Armin B Cremers, Dieter Fox, Dirk Hähnel, Gerhard Lakemeyer, Dirk Schulz, Walter Steiner, and Sebastian Thrun (1998). "The interactive museum tour-guide robot". In: *Aaai/iaai*, pp. 11–18 (cit. on p. 94).

Calonder, Michael, Vincent Lepetit, Christoph Strecha, and Pascal Fua (Sept. 2010). "BRIEF: Binary Robust Independent Elementary Features". In: vol. 6314, pp. 778–792 (cit. on p. 34).

Carion, Nicolas, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko (2020). *End-to-End Object Detection with Transformers*. arXiv: 2005.12872 [cs.CV] (cit. on p. 109).

Chang, Angel, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niebner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang (2018). "Matterport3D: Learning from RGB-D data in indoor environments". In: (cit. on pp. 40, 92).

Chang, Matthew, Arjun Gupta, and Saurabh Gupta (2020). "Semantic Visual Navigation by Watching Youtube Videos". In: *NeurIPS* (cit. on pp. ix, 4, 5, 39).

Chaplot, Devendra Singh, Murtaza Dalal, Saurabh Gupta, Jitendra Malik, and Ruslan Salakhutdinov (2021). "SEAL: Self-supervised Embodied Active Learning

using Exploration and 3D Consistency". In: *NeurIPS*, pp. 13086–13098 (cit. on p. 80).

Chaplot, Devendra Singh, Dhiraj Gandhi, Abhinav Gupta, and Ruslan Salakhutdinov (2020a). "Object Goal Navigation using Goal-Oriented Semantic Exploration". In: *In Neural Information Processing Systems* (cit. on pp. 7, 44, 53–56, 79, 80, 83, 95, 97, 104, 109).

Chaplot, Devendra Singh, Dhiraj Gandhi, Saurabh Gupta, Abhinav Gupta, and Ruslan Salakhutdinov (2020b). "Learning To Explore Using Active Neural SLAM". In: *International Conference on Learning Representations (ICLR)* (cit. on pp. xi, xvi, 41, 44, 51–53, 55, 76, 79–81, 86–88, 95).

Chaplot, Devendra Singh, Helen Jiang, Saurabh Gupta, and Abhinav Gupta (2020c). "Semantic Curiosity for Active Visual Learning". In: *ECCV*. Vol. 12351. Springer, pp. 309–326 (cit. on pp. 79, 80).

Chaplot, Devendra Singh, Ruslan Salakhutdinov, Abhinav Gupta, and Saurabh Gupta (2020d). "Neural Topological SLAM for Visual Navigation". In: *CVPR* (cit. on pp. 7, 44, 51, 79, 95).

Chattopadhyay, Prithvijit, Judy Hoffman, Roozbeh Mottaghi, and Aniruddha Kembhavi (2021). "RobustNav: Towards Benchmarking Robustness in Embodied Navigation". In: *CoRR* 2106.04531 (cit. on pp. 63, 79, 97).

Chen, L.-C., G. Papandreou, F. Schroff, and A. Hartwig (2017). "Rethinking Atrous Convolution for Semantic Image Segmentation". In: *CVPR* (cit. on pp. 4, 83).

Chen, Lili, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch (2021). "Decision Transformer: Reinforcement Learning via Sequence Modeling". In: *NeurIPS* (cit. on p. 79).

Chen, Shizhe, Pierre-Louis Guhur, Makarand Tapaswi, Cordelia Schmid, and Ivan Laptev (2022a). "Think Global, Act Local: Dual-scale Graph Transformer for Vision-and-Language Navigation". In: (cit. on pp. 79, 95).

Chen, Tao, Saurabh Gupta, and Abhinav Gupta (2019). "Learning Exploration Policies for Navigation". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SyMWn05F7 (cit. on pp. 4, 21, 41, 78, 80).

Chen, Tianyou, Guofeng Zhang, Xiaoguang Hu, and Jin Xiao (2018). "Unmanned aerial vehicle route planning method based on a star algorithm". In: *2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 1510–1514 (cit. on p. 23).

Chen, Ting, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton (2020a). "A Simple Framework for Contrastive Learning of Visual Representations". In: *CoRR* abs/2002.05709. arXiv: 2002.05709. URL: https://arxiv.org/abs/2002.05709 (cit. on p. 49).

Chen, Xiangning, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and

Quoc V. Le (2023). "Symbolic Discovery of Optimization Algorithms". In: *ArXiv* abs/2302.06675. URL: https://api.semanticscholar.org/CorpusID:256846990 (cit. on p. 1).

Chen, Xinlei, Haoqi Fan, Ross B. Girshick, and Kaiming He (2020b). "Improved Baselines with Momentum Contrastive Learning". In: *CoRR* abs/2003.04297. arXiv: 2003.04297. URL: https://arxiv.org/abs/2003.04297 (cit. on p. 50).

Chen, Yujing, Fenghua Zhao, and Yunjiang Lou (2022b). "Interactive Model Predictive Control for Robot Navigation in Dense Crowds". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.4, pp. 2289–2301 (cit. on p. 4).

Cheng, Li Ping, Chuanxi Liu, and Bo Yan (2014). "Improved hierarchical A-star algorithm for optimal parking path planning of the large parking lot". In: *2014 IEEE International Conference on Information and Automation (ICIA)*, pp. 695–698 (cit. on p. 23).

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). *Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation*. arXiv: 1406.1078 [cs.CL] (cit. on pp. 4, 25).

Choi, Heesun, Cindy Crump, Christian Duriez, Asher Elmquist, Gregory D. Hager, David Han, Frank Hearl, Jessica K. Hodgins, Abhinandan Jain, Frederick A. Leve, Chen Li, Franziska Meier, Dan Negrut, Ludovic Righetti, Alberto Rodriguez, Jie Tan, and Jeffrey C. Trinkle (2020). "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward". In: *Proceedings of the National Academy of Sciences of the United States of America* 118 (cit. on pp. 40, 62).

Codevilla, Felipe, Eder Santana, Antonio M. López, and Adrien Gaidon (2019). *Exploring the Limitations of Behavior Cloning for Autonomous Driving*. arXiv: 1904.08980 [cs.CV] (cit. on p. 42).

Coumans, Erwin and Yunfei Bai (2016). *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. URL: http://pybullet.org (cit. on pp. 5, 39).

Das, Abhishek, Samyak Datta, Georgia Gkioxari, Stefan Lee, Devi Parikh, and Dhruv Batra (2018). "Embodied Question Answering". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 3).

Dashora, N., D. Shin, D. Shah, H. Leopold, D. Fan, A. Agha-Mohammadi, N. Rhinehart, and Sergey Levine (2021). "Hybrid Imitative Planning with Geometric and Predictive Costs in Off-road Environments". In: *NeurIPS Workshop on Deep-RL* (cit. on pp. 93, 95).

Deitke, Matt, Dhruv Batra, Yonatan Bisk, Tommaso Campari, Angel X. Chang, Devendra Singh Chaplot, Changan Chen, Claudia Pérez D'Arpino, Kiana Ehsani, Ali Farhadi, Li Fei-Fei, Anthony Francis, Chuang Gan, Kristen Grauman, David Hall, Winson Han, Unnat Jain, Aniruddha Kembhavi, Jacob Krantz, Stefan Lee, Chengshu Li, Sagnik Majumder, Oleksandr Maksymets, Roberto Martín-

Martín, Roozbeh Mottaghi, Sonia Raychaudhuri, Mike Roberts, Silvio Savarese, Manolis Savva, Mohit Shridhar, Niko Sünderhauf, Andrew Szot, Ben Talbot, Joshua B. Tenenbaum, Jesse Thomason, Alexander Toshev, Joanne Truong, Luca Weihs, and Jiajun Wu (2022). *Retrospectives on the Embodied AI Workshop*. arXiv: 2210.06849 [cs.CV] (cit. on pp. 2, 12, 16, 18, 19).

Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09* (cit. on p. 4).

Dey, Sombit, Assem Sadek, Gianluca Monaci, Boris Chidlovskii, and Christian Wolf (2022). "Learning whom to trust in navigation: dynamically switching between classical and neural planning". In: *IROS 2023* (cit. on pp. 10, 92).

Dijkstra, Edsger W. (1959). "A note on two problems in connexion with graphs". In: *Numerische Mathematik* 1, pp. 269–271 (cit. on pp. 3, 4, 22).

Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby (2021). *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv: 2010.11929 [cs.CV] (cit. on p. 4).

Doucet, A., Nando de Freitas, Kevin P. Murphy, and Stuart J. Russell (2000). "Rao-Blackwellised Particle Filtering for Dynamic Bayesian Networks". In: *ArXiv* abs/1301.3853 (cit. on p. 37).

Drews, Paul, Grady Williams, Brian Goldfain, Evangelos A. Theodorou, and James M. Rehg (2017). *Aggressive Deep Driving: Model Predictive Control with a CNN Cost Model*. arXiv: 1707.05303 [cs.RO] (cit. on p. 57).

Drews, Paul, Grady Williams, Brian Goldfain, Evangelos A. Theodorou, and James M. Rehg (2018). *Vision-Based High Speed Driving with a Deep Dynamic Observer*. arXiv: 1812.02071 [cs.RO] (cit. on p. 57).

Du, Heming, Xin Yu, and Liang Zheng (2021). "VTNet: Visual Transformer Network for Object Goal Navigation". In: *ICLR* (cit. on p. 95).

Durrant-Whyte, Hugh F. and Tim Bailey (2006). "Simultaneous Localisation and Mapping (SLAM) : Part I The Essential Algorithms". In: URL: https://api.semanticscholar.org/CorpusID:17915751 (cit. on p. 33).

Embodied AI Workshop, CVPR 2021 Challenge (2021). https://embodied-ai.org/ (cit. on p. 63).

Espeholt, Lasse, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. (2018). "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *Proceedings of the International Conference on Machine Learning (ICML)* (cit. on p. 5).

Eysenbach, Benjamin, Shreyas Chaudhari, Swapnil Asawa, Sergey Levine, and Ruslan Salakhutdinov (2021). "Off-Dynamics Reinforcement Learning: Training for Transfer with Domain Classifiers". In: *Intern. Conf. Learning Representations, ICLR* (cit. on pp. 63, 79).

Fadzli, Syed Abdullah, Sani Iyal Abdulkadir, Mokhairi Makhtar, and Azrul Amri Jamal (2015). "Robotic Indoor Path Planning Using Dijkstra's Algorithm with Multi-Layer Dictionaries". In: *2015 2nd International Conference on Information Science and Security (ICISS)*, pp. 1–4 (cit. on p. 22).

Faessler, Matthias, Flavio Fontana, Christian Forster, and Davide Scaramuzza (2015). "Automatic re-initialization and failure recovery for aggressive flight with a monocular vision-based quadrotor". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1722–1729 (cit. on p. 57).

Fang, Kuan, Alexander Toshev, Li Fei-Fei, and Silvio Savarese (2019). *Scene Memory Transformer for Embodied Agents in Long-Horizon Tasks*. arXiv: 1903.03878 [cs.LG] (cit. on pp. 18, 79, 95).

Ferguson, Dave, Maxim Likhachev, and Anthony Stentz (2005). "A Guide to Heuristic-based Path Planning". In: (cit. on pp. 4, 22).

Fox, Dieter, Wolfram Burgard, and Sebastian Thrun (1997). "The dynamic window approach to collision avoidance". In: *IEEE Robotics & Automation Magazine* 4.1, pp. 23–33 (cit. on p. 94).

Gervet, Theophile, Soumith Chintala, Dhruv Batra, Jitendra Malik, and Devendra Singh Chaplot (2023). *Navigating to Objects in the Real World*. eprint: 2212.00922 (cs.RO) (cit. on p. 109).

Ghosh, D., J. Rahme, A. Kumar, A. Zhang, R.P. Adams, and S. Levine (2021). "Why Generalization in RL is Difficult: Epistemic POMDPs and Implicit Partial Observability". In: *NeurIPS* (cit. on p. 82).

Girshick, Ross, Jeff Donahue, Trevor Darrell, and Jitendra Malik (2014). *Rich feature hierarchies for accurate object detection and semantic segmentation*. arXiv: 1311.2524 [cs.CV] (cit. on p. 1).

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press (cit. on p. 1).

Guo, Zhaohan Daniel, Mohammad Gheshlaghi Azar, Bilal Piot, Bernardo A. Pires, Toby Pohlen, and Rémi Munos (2018). "Neural Predictive Belief Representations". In: *CoRR* abs/1811.06407. arXiv: 1811.06407. URL: http://arxiv.org/abs/1811.06407 (cit. on p. 46).

Gupta, Saurabh, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik (2017a). *Cognitive Mapping and Planning for Visual Navigation* (cit. on pp. 44, 51, 62, 63, 95, 97).

Gupta, Saurabh, David Fouhey, Sergey Levine, and Jitendra Malik (2017b). *Unifying Map and Landmark Based Representations for Visual Navigation*. arXiv: 1712.08125 [cs.CV] (cit. on p. 44).

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). *Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor*. arXiv: 1801.01290 [cs.LG] (cit. on pp. 44, 79).

Habitat Challenge (2020). https://aihabitat.org/challenge/2020/ (cit. on p. 63).

Habitat Sim2real (2021). https://github.com/wgw101/habitat_sim2real (cit. on p. 84).

Hahn, Meera, Devendra Chaplot, Shubham Tulsiani, Mustafa Mukadam, James M. Rehg, and Abhinav Gupta (2021). *No RL, No Simulation: Learning to Navigate without Navigating*. arXiv: 2110.09470 [cs.CV] (cit. on p. 51).

Hansen, Nicklas, Yu Sun, Pieter Abbeel, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang (2020a). "Self-Supervised Policy Adaptation during Deployment". In: *CoRR* abs/2007.04309. arXiv: 2007.04309. URL: https://arxiv.org/abs/2007.04309 (cit. on p. 49).

Hansen, Nicklas and Xiaolong Wang (2020b). "Generalization in Reinforcement Learning by Soft Data Augmentation". In: *CoRR* abs/2011.13389. arXiv: 2011.13389. URL: https://arxiv.org/abs/2011.13389 (cit. on p. 49).

Hartley, R. I. and A. Zisserman (2004). *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518 (cit. on p. 14).

He, Kaiming, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick (2019). "Momentum Contrast for Unsupervised Visual Representation Learning". In: *CoRR* abs/1911.05722. arXiv: 1911.05722. URL: http://arxiv.org/abs/1911.05722 (cit. on p. 49).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *CVPR* (cit. on pp. 1, 4, 99).

Henderson, Peter, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger (2019). *Deep Reinforcement Learning that Matters*. arXiv: 1709.06560 [cs.LG] (cit. on p. 5).

Henriques, João F. and Andrea Vedaldi (2018). "MapNet: An Allocentric Spatial Memory for Mapping Environments". In: *CVPR*, pp. 8476–8484 (cit. on pp. 79, 84, 95).

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (2015). *Distilling the Knowledge in a Neural Network*. arXiv: 1503.02531 [stat.ML] (cit. on p. 50).

Hochreiter, Sepp and Jürgen Schmidhuber (Dec. 1997). "Long Short-term Memory". In: *Neural computation* 9, pp. 1735–80 (cit. on pp. 4, 25).

Höfer, Sebastian, Kostas E. Bekris, Ankur Handa, Juan Camilo Gamboa Higuera, Florian Golemo, Melissa Mozifian, Christopher G. Atkeson, Dieter Fox, Ken Goldberg, John Leonard, C. Karen Liu, Jan Peters, Shuran Song, Peter Welinder, and Martha White (2020). "Perspectives on Sim2Real Transfer for Robotics: A Summary of the R: SS 2020 Workshop". In: *CoRR* abs/2012.03806. URL: https://arxiv.org/abs/2012.03806 (cit. on p. 79).

Jaderberg, Max, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z. Leibo, David Silver, and Koray Kavukcuoglu (2016a). "Reinforcement Learning with Unsupervised Auxiliary Tasks". In: *CoRR* abs/1611.05397. arXiv: 1611.05397. URL: http://arxiv.org/abs/1611.05397 (cit. on p. 46).

Jaderberg, Max, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu (2016b). *Spatial Transformer Networks*. arXiv: 1506.02025 [cs.CV] (cit. on p. 52).

Jain, Jitesh, Jiachen Li, MangTik Chiu, Ali Hassani, Nikita Orlov, and Humphrey Shi (2023). "OneFormer: One Transformer to Rule Universal Image Segmentation". In: (cit. on p. 1).

Janner, Michael, Qiyang Li, and Sergey Levine (2021). "Offline Reinforcement Learning as One Big Sequence Modeling Problem". In: *NeurIPS* (cit. on p. 79).

Jaunet, Theo, Guillaume Bono, Romain Vuillemot, and Christian Wolf (2021). "Sim2RealViz: Visualizing the Sim2Real Gap in Robot Ego-Pose Estimation". In: *arXiv preprint arXiv:2109.11801*. (cit. on p. 5).

Jaunet, Theo, Romain Vuillemot, and Christian Wolf (2020). *DRLViz: Understanding Decisions and Memory in Deep Reinforcement Learning*. arXiv: 1909.02982 [cs.LG] (cit. on pp. 5, 63).

Jazwinski, A. H. (1970). *Stochastic Processes and Filtering Theory*. Academic Press (cit. on p. 37).

K. Weerakoon and A.J. Sathyamoorthy and D. Manocha (2022). "Sim-to-Real Strategy for Spatially Aware Robot Navigation in Uneven Outdoor Environments". In: *ICRA 2022 Workshop on Releasing Robots into the Wild:* (cit. on pp. 93, 95).

Kadian, Abhishek, Joanne Truong, Aaron Gokaslan, Alexander Clegg, Erik Wijmans, Stefan Lee, Manolis Savva, Sonia Chernova, and Dhruv Batra (2020). "Sim2Real Predictivity: Does Evaluation in Simulation Predict Real-World Performance?" In: *IEEE Robotics Autom. Lett.* 5.4, pp. 6670–6677 (cit. on pp. 60, 63, 76, 79, 99, 100).

Kálmán, Rudolf E. (1960). "A new approach to linear filtering and prediction problems" transaction of the asme journal of basic". In: (cit. on p. 37).

Kanervisto, Anssi, Janne Karttunen, and Ville Hautamäki (2020a). *Playing Minecraft with Behavioural Cloning*. arXiv: 2005.03374 [cs.AI] (cit. on p. 42).

Kanervisto, Anssi, Joonas Pussinen, and Ville Hautamäki (2020b). *Benchmarking End-to-End Behavioural Cloning on Video Games*. arXiv: 2004.00981 [cs.AI] (cit. on p. 42).

Kang, Hwan Il, Byung hee Lee, and Kab il Kim (2008). "Path Planning Algorithm Using the Particle Swarm Optimization and the Improved Dijkstra Algorithm". In: *2008 IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application* 2, pp. 1002–1004 (cit. on p. 22).

Kastner, Linh, Johannes Cox, Teham Buiyan, and Jens Lambrecht (2022). "All-in-one: A DRL-based control switch combining state-of-the-art navigation planners". In: *ICRA* (cit. on pp. 95, 96).

Kaufmann, Elia, Mathias Gehrig, Philipp Foehn, René Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza (2019). *Beauty and the Beast: Optimal Methods Meet Learning for Drone Racing*. arXiv: 1810.06224 [cs.RO] (cit. on p. 57).

Kaufmann, Elia, Antonio Loquercio, Rene Ranftl, Alexey Dosovitskiy, Vladlen Koltun, and Davide Scaramuzza (2018). *Deep Drone Racing: Learning Agile Flight in Dynamic Environments*. arXiv: 1806.08548 [cs.RO] (cit. on p. 57).

Kolve, Eric, Roozbeh Mottaghi, Daniel Gordon, Yuke Zhu, Abhinav Gupta, and Ali Farhadi (2017). "AI2-THOR: An Interactive 3D Environment for Visual AI". In: *CoRR* 1712.05474 (cit. on pp. 39, 60, 62, 63, 92).

Konolige, Kurt (2000). "A gradient method for realtime robot control". In: *IROS* (cit. on p. 94).

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf (cit. on p. 1).

Kwon, Obin, Nuri Kim, Yunho Choi, Hwiyeon Yoo, Jeongho Park, and Songhwai Oh (2021). "Visual Graph Memory with Unsupervised Representation for Visual Navigation". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15870–15879 (cit. on p. 44).

Labbé, Mathieu and François Michaud (2019). "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation". In: *Journal of Field Robotics* 36.2, pp. 416–446 (cit. on pp. 77, 81, 94, 100).

Laskin, Michael, Aravind Srinivas, and Pieter Abbeel (2020). "CURL: Contrastive Unsupervised Representations for Reinforcement Learning". In: *Proceedings of the 37th International Conference on Machine Learning*. Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 5639–5650. URL: https://proceedings.mlr.press/v119/laskin20a.html (cit. on p. 49).

Lecun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on p. 4).

Leonard, John J. and Hans Jacob S. Feder (2000). "A Computationally Efficient Method for Large-Scale Concurrent Mapping and Localization". In: (cit. on p. 37).

Levine, Sergey, Chelsea Finn, Trevor Darrell, and Pieter Abbeel (2015). "End-to-End Training of Deep Visuomotor Policies". In: *Journal of Machine Learning Research* abs/1504.00702. URL: http://arxiv.org/abs/1504.00702 (cit. on p. 5).

Li, Chenliang, Haiyang Xu, Junfeng Tian, Wei Wang, Ming Yan, Bin Bi, Jiabo Ye, Hehong Chen, Guohai Xu, Zheng Cao, et al. (2022a). "mPLUG: Effective and Efficient Vision-Language Learning by Cross-modal Skip-connections". In: *arXiv preprint arXiv:2205.12005* (cit. on p. 1).

Li, Feng, Hao Zhang, Huaizhe xu, Shilong Liu, Lei Zhang, Lionel M. Ni, and Heung-Yeung Shum (2022b). *Mask DINO: Towards A Unified Transformer-based Framework for Object Detection and Segmentation*. arXiv: 2206.02777 [cs.CV] (cit. on p. 1).

Li, Shangda, Devendra Singh Chaplot, Yao-Hung Hubert Tsai, Yue Wu, Louis-Philippe Morency, and Ruslan Salakhutdinov (2020). *Unsupervised Domain Adaptation for Visual Navigation*. arXiv: 2010.14543 [cs.LG] (cit. on p. 5).

Li, Xueting, Shalini De Mello, Xiaolong Wang, Ming-Hsuan Yang, Jan Kautz, and Sifei Liu (2022c). "Learning Continuous Environment Fields via Implicit Functions". In: *ICLR* (cit. on pp. 79, 95, 110).

Li, Yunzhu, Shuang Li, Vincent Sitzmann, Pulkit Agrawal, and Antonio Torralba (2021). *3D Neural Scene Representations for Visuomotor Control*. arXiv: 2107.04004 [cs.RO] (cit. on p. 110).

Lin, Tsung-Yi, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár (2015). *Microsoft COCO: Common Objects in Context*. arXiv: 1405.0312 [cs.CV] (cit. on p. 4).

Lipton, Zachary (Oct. 2016). "The Mythos of Model Interpretability". In: *Communications of the ACM* 61 (cit. on p. 5).

Liu, Liyuan, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han (2020). "Understanding the Difficulty of Training Transformers". In: *Conference on Empirical Methods in Natural Language Processing*. URL: https://api.semanticscholar.org/CorpusID:215814515 (cit. on p. 1).

LoCoBot: An Open Source Low Cost Robot (2017). http://www.locobot.org (cit. on pp. 64, 84, 100).

Lowe, David G (2004). "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2, pp. 91–110 (cit. on p. 32).

Luo, Haokuan, Albert Yue, Zhang-Wei Hong, and Pulkit Agrawal (2022). *Stubborn: A Strong Baseline for Indoor Object Navigation*. arXiv: 2203.07359 [cs.RO] (cit. on pp. xv, 54, 55).

Macenski, Steve, Francisco Martín, Ruffin White, and Jonatan Ginés Clavero (2020). "The marathon 2: A navigation system". In: *IROS* (cit. on pp. 92, 94).

Majumdar, Arjun, Gunnar A Sigurdsson, Robinson Piramuthu, Jesse Thomason, Dhruv Batra, and Gaurav S Sukhatme (2022). "SSL Enables Learning from Sparse Rewards in Image-Goal Navigation". In: *International Conference on Machine Learning*. PMLR, pp. 14774–14785 (cit. on pp. 5, 50).

Maksymets, Oleksandr, Vincent Cartillier, Aaron Gokaslan, Erik Wijmans, Wojciech Galuba, Stefan Lee, and Dhruv Batra (2021). "THDA: Treasure Hunt Data Augmentation for Semantic Navigation". In: *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*. IEEE, pp. 15354–15363. URL: https://doi.org/10.1109/ICCV48922.2021.01509 (cit. on p. 50).

Marder-Eppstein, Eitan, Eric Berger, Tully Foote, Brian Gerkey, and Kurt Konolige (2010). "The office marathon: Robust navigation in an indoor office environment". In: *ICRA* (cit. on pp. 92, 94).

Martin, Jesus Bujalance and Fabien Moutarde (2022). "Reward Relabelling for combined Reinforcement and Imitation Learning on sparse-reward tasks". In: *CoRR* abs/2201.03834. arXiv: 2201.03834. URL: https://arxiv.org/abs/2201.03834 (cit. on p. 30).

Marza, Pierre, L. Matignon, O. Simonin, and C. Wolf (2022). "Teaching Agents how to Map: Spatial Reasoning for Multi-Object Navigation". In: (cit. on pp. xi, xv, 5, 47, 49, 75, 77, 78, 84–87, 100).

Marza, Pierre, Laetitia Matignon, Olivier Simonin, Dhruv Batra, Christian Wolf, and Devendra Singh Chaplot (2023a). *AutoNeRF: Training Implicit Scene Representations with Autonomous Agents*. arXiv: 2304.11241 [cs.CV] (cit. on p. 110).

Marza, Pierre, Laetitia Matignon, Olivier Simonin, and Christian Wolf (2023b). "Multi-Object Navigation with dynamically learned neural implicit representations". In: *International Conference on Computer Vision (ICCV)* (cit. on pp. 95, 110).

Matl, Mattew (2020). *PyRender*. URL: http://github.com/mmatl/pyrender (cit. on p. 39).

Matterport (2020). https://matterport.com (cit. on pp. 66, 107).

Meinhardt, Tim, Alexander Kirillov, Laura Leal-Taixe, and Christoph Feichtenhofer (2022). "TrackFormer: Multi-Object Tracking with Transformers". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 109).

Metropolis, N. and S. Ulam (July 1949). "The Monte Carlo Method". In: *Journal of the American Statistical Association* 44, pp. 335–341 (cit. on p. 37).

Mezghani, Lina, Sainbayar Sukhbaatar, Thibaut Lavril, Oleksandr Maksymets, Dhruv Batra, Piotr Bojanowski, and Karteek Alahari (2021). "Memory-Augmented Reinforcement Learning for Image-Goal Navigation". In: *CoRR* abs/2101.05181. arXiv: 2101.05181. URL: https://arxiv.org/abs/2101.05181 (cit. on pp. 50, 51).

Mildenhall, Ben, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng (2020). "NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis". In: *ECCV* (cit. on p. 109).

Minsky, Marvin (1961). "Steps toward Artificial Intelligence". In: *Proceedings of the IRE* 49.1, pp. 8–30 (cit. on p. 25).

Mirowski, Piotr, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell (2016). "Learning to Navigate in Complex Environments". In: *CoRR* abs/1611.03673. arXiv: 1611.03673. URL: http://arxiv.org/abs/1611.03673 (cit. on p. 46).

Misra, Ishan and Laurens van der Maaten (2019). "Self-Supervised Learning of Pretext-Invariant Representations". In: *CoRR* abs/1912.01991. arXiv: 1912.01991. URL: http://arxiv.org/abs/1912.01991 (cit. on p. 49).

Mnih, Volodymyr, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu (2016). *Asyn-*

chronous Methods for Deep Reinforcement Learning. arXiv: 1602.01783 [cs.LG] (cit. on p. 27).

Montemerlo, Michael, Sebastian Thrun, Daphne Koller, and Ben Wegbreit (2002). "FastSLAM: a factored solution to the simultaneous localization and mapping problem". In: *AAAI/IAAI* (cit. on p. 37).

Moutarlier, Philippe and R. Chatila (1989). "Stochastic multisensory data fusion for mobile robot location and environment modeling". In: (cit. on pp. 3, 36).

Mueller, Mark W., Markus Hehn, and Raffaello D'Andrea (2013). "A computationally efficient algorithm for state-to-state quadrocopter trajectory generation and feasibility verification". In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3480–3486 (cit. on p. 57).

Mur-Artal, Raul, J. M. M. Montiel, and Juan D. Tardos (2015). "ORB-SLAM: A Versatile and Accurate Monocular SLAM System". In: 31.5. URL: https://doi.org/10.1109/TRO.2015.2463671 (cit. on p. 33).

Mur-Artal, Raúl and Juan D. Tardós (2017). "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras". In: *IEEE Transactions on Robotics* 33.5, pp. 1255–1262 (cit. on p. 33).

Murali, Adithyavairavan, Tao Chen, Kalyan Vasudev Alwala, Dhiraj Gandhi, Lerrel Pinto, Saurabh Gupta, and Abhinav Gupta (2019). "PyRobot: An Open-source Robotics Framework for Research and Benchmarking". In: *CoRR* abs/1906.08236. arXiv: 1906.08236. URL: http://arxiv.org/abs/1906.08236 (cit. on pp. xi, 73).

Müller, Matthias, Alexey Dosovitskiy, Bernard Ghanem, and Vladlen Koltun (2018). *Driving Policy Transfer via Modularity and Abstraction*. arXiv: 1804.09364 [cs.RO] (cit. on p. 58).

Newcombe, Richard A., Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon (2011a). "KinectFusion: Real-time dense surface mapping and tracking". In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136 (cit. on p. 35).

Newcombe, Richard A., Steven J. Lovegrove, and Andrew J. Davison (2011b). "DTAM: Dense tracking and mapping in real-time". In: *2011 International Conference on Computer Vision*, pp. 2320–2327 (cit. on p. 35).

Oord, Aäron van den, Yazhe Li, and Oriol Vinyals (2018). "Representation Learning with Contrastive Predictive Coding". In: *CoRR* abs/1807.03748. arXiv: 1807.03748. URL: http://arxiv.org/abs/1807.03748 (cit. on p. 50).

OpenAI (2018). "OpenAI Five". In: URL: https://openai.com/research/openai-five (cit. on p. 5).

OpenAI (2023). *GPT-4 Technical Report*. arXiv: 2303.08774 [cs.CL] (cit. on p. 1).

Partsey, R., E. Wijmans, N. Yokoyama, O. Dobosevych, D. Batra, and O. Maksymets (2022). "Is Mapping Necessary for Realistic PointGoal Navigation?" In: *CVPR* (cit. on p. 92).

Pashevich, A., C. Schmid, and C. Sun (2021). "Episodic Transformer for Vision-and-Language Navigation". In: *ICCV* (cit. on pp. 41, 79).

Pathak, Deepak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell (2017). "Curiosity-driven Exploration by Self-supervised Prediction". In: *CoRR* abs/1705.05363. arXiv: 1705.05363. URL: http://arxiv.org/abs/1705.05363 (cit. on p. 46).

Peng, Xue Bin, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel (2018). "Sim-to-Real Transfer of Robotic Control with Dynamics Randomization". In: *IEEE International Conference on Robotics and Automation, ICRA*, pp. 1–8 (cit. on pp. 63, 79).

Puig, Xavier, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang, Sanja Fidler, and Antonio Torralba (2018). "VirtualHome: Simulating Household Activities via Programs". In: *CoRR* abs/1806.07011. arXiv: 1806.07011. URL: http://arxiv.org/abs/1806.07011 (cit. on p. 40).

Ramakrishnan, Santhosh K., Devendra Singh Chaplot, Ziad Al-Halah, Jitendra Malik, and Kristen Grauman (2022). "PONI: Potential Functions for ObjectGoal Navigation with Interaction-free Learning". In: *Computer Vision and Pattern Recognition (CVPR), 2022 IEEE Conference on*. IEEE (cit. on pp. 55, 56, 76, 79, 83, 109).

Ramakrishnan, Santhosh Kumar, Aaron Gokaslan, Erik Wijmans, Oleksandr Maksymets, Alexander Clegg, John M Turner, Eric Undersander, Wojciech Galuba, Andrew Westbury, Angel X Chang, Manolis Savva, Yili Zhao, and Dhruv Batra (2021). "Habitat-Matterport 3D Dataset (HM3D): 1000 Large-scale 3D Environments for Embodied AI". In: *NeurIPS — Datasets and Benchmarks Track* (cit. on p. 92).

Ramesh, Aditya, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen (2022). "Hierarchical Text-Conditional Image Generation with CLIP Latents". In: *ArXiv* abs/2204.06125. URL: https://api.semanticscholar.org/CorpusID:248097655 (cit. on p. 1).

Ramrakhya, Ram, Dhruv Batra, Erik Wijmans, and Abhishek Das (2023). *PIRLNav: Pretraining with Imitation and RL Finetuning for ObjectNav*. arXiv: 2301.07302 [cs.LG] (cit. on pp. 30, 42).

Ramrakhya, Ram, Eric Undersander, Dhruv Batra, and Abhishek Das (2022). "Habitat-Web: Learning Embodied Object-Search Strategies from Human Demonstrations at Scale". In: *CVPR* (cit. on pp. 5, 42).

Recast & Detour library (2016). https://github.com/recastnavigation/recastnavigation (cit. on p. 81).

Reed, Scott, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, Tom Eccles, Jake Bruce, Ali Razavi, Ashley Edwards, Nicolas Heess, Yutian Chen, Raia Hadsell, Oriol Vinyals, Mahyar Bordbar, and Nando de Freitas (2022). "A Generalist Agent". In: *arXiv:2205.06175* (cit. on pp. 79, 95).

Rösmann, Christoph, Frank Hoffmann, and Torsten Bertram (2015). "Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control". In: *European Control Conference (ECC)* (cit. on p. 94).

Ross, Stéphane, Geoffrey J. Gordon, and J. Andrew Bagnell (2010). "No-Regret Reductions for Imitation Learning and Structured Prediction". In: *CoRR* abs/1011.0686. arXiv: 1011.0686. URL: http://arxiv.org/abs/1011.0686 (cit. on p. 30).

Rosten, E. and T. Drummond (2005). "Fusing points and lines for high performance tracking". In: *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*. Vol. 2, 1508–1515 Vol. 2 (cit. on p. 34).

R.S. Sutton D. Precup, S. Singh (1999). "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial Intelligence* 112, pp. 181–211 (cit. on p. 95).

Sadek, Assem, Guillaume Bono, Boris Chidlovskii, Atilla Baskurt, and Christian Wolf (2022a). "Multi-Object Navigation in real environments using hybrid policies". In: *ICRA 2023* (cit. on pp. 10, 76, 95, 101).

Sadek, Assem, Guillaume Bono, Boris Chidlovskii, and Christian Wolf (2022b). "An in-depth experimental study of sensor usage and visual reasoning of robots navigating in real environments". In: *ICRA 2022* (cit. on pp. 10, 60, 76, 79, 87).

Samak, Tanmay Vilas, Chinmay Vilas Samak, and Sivanathan Kandhasamy (2021). "Robust Behavioral Cloning for Autonomous Vehicles Using End-to-End Imitation Learning". In: *SAE International Journal of Connected and Automated Vehicles* 4.3, pp. 279–295. URL: https://doi.org/10.4271/12-04-03-0023 (cit. on p. 42).

Savinov, Nikolay, Alexey Dosovitskiy, and Vladlen Koltun (2018). "Semi-parametric topological memory for navigation". In: *International Conference on Learning Representations (ICLR)* (cit. on pp. 44, 79, 95).

Savva, Manolis, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, Julian Straub, Jia Liu, Vladlen Koltun, Jitendra Malik, Devi Parikh, and Dhruv Batra (2019). "Habitat: A Platform for Embodied AI Research". In: (cit. on pp. 4, 5, 39, 40, 46, 55, 60, 62–65, 79, 80, 84, 92, 99).

Schöps, Thomas, Jakob J. Engel, and Daniel Cremers (2014). "Semi-dense visual odometry for AR on a smartphone". In: *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 145–150. URL: https://api.semanticscholar.org/CorpusID:1453808 (cit. on p. 34).

Schulman, John, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel (2017a). *Trust Region Policy Optimization*. arXiv: 1502.05477 [cs.LG] (cit. on p. 28).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017b). *Proximal Policy Optimization Algorithms*. arXiv: 1707.06347 [cs.LG] (cit. on pp. 28, 97).

Selvaraju, Ramprasaath R., Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra (2020). "Grad-CAM: Visual Explanations

from Deep Networks via Gradient-Based Localization". In: *Int. J. Comput. Vis.* 128.2, pp. 336–359 (cit. on pp. 63, 69).

Sethian, James A (1996a). "A fast marching level set method for monotonically advancing fronts." In: *PNAS* 93.4, pp. 1591–1595 (cit. on pp. 3, 90, 94, 97).

Sethian, James A. (1996b). "A fast marching level set method for monotonically advancing fronts." In: *Proceedings of the National Academy of Sciences of the United States of America* 93 4, pp. 1591–5 (cit. on pp. 55, 83).

Seyde, Tim, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin Riedmiller, Markus Wulfmeier, and Daniela Rus (2021). *Is Bang-Bang Control All You Need? Solving Continuous Control with Bernoulli Policies*. arXiv: 2111.02552 [cs.LG] (cit. on p. 15).

Shah, Dhruv and Sergey Levine (2022). "ViKiNG: Vision-Based Kilometer-Scale Navigation with Geographic Hints". In: *RSS* (cit. on p. 95).

Shen, Bokui, Fei Xia, Chengshu Li, Roberto Martín-Martín, Linxi Fan, Guanzhi Wang, Shyamal Buch, Claudia D'Arpino, Sanjana Srivastava, Lyne P. Tchapmi, Micael E. Tchapmi, Kent Vainio, Li Fei-Fei, and Silvio Savarese (2020). "iGibson, a Simulation Environment for Interactive Tasks in Large Realistic Scenes". In: *CoRR* abs/2012.02924. arXiv: 2012.02924. URL: https://arxiv.org/abs/2012.02924 (cit. on pp. 39, 63).

Siciliano, Bruno and Oussama Khatib (2007). *Springer Handbook of Robotics*. Berlin, Heidelberg: Springer-Verlag (cit. on p. 35).

Silva, Marcelo, Ana Cristina Garcia, and Aura Conci (Nov. 2010). "A Multi-agent System for Dynamic Path Planning". In: pp. 47 –51 (cit. on p. 22).

Smith, Randall, Matthew Self, and Peter Cheeseman (Jan. 1987). "A stochastic map for uncertain spatial relationships". In: pp. 467–474 (cit. on pp. 3, 36).

Stooke, Adam, Kimin Lee, Pieter Abbeel, and Michael Laskin (2020). "Decoupling Representation Learning from Reinforcement Learning". In: *CoRR* abs/2009.08319. arXiv: 2009.08319. URL: https://arxiv.org/abs/2009.08319 (cit. on p. 49).

Sucar, Edgar, Shikun Liu, Joseph Ortiz, and Andrew J. Davison (2021). *iMAP: Implicit Mapping and Positioning in Real-Time*. arXiv: 2103.12352 [cs.CV] (cit. on p. 110).

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (2014). *Sequence to Sequence Learning with Neural Networks*. arXiv: 1409.3215 [cs.CL] (cit. on p. 1).

Szot, Andrew, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, Aaron Gokaslan, Vladimir Vondrus, Sameer Dharur, Franziska Meier, Wojciech Galuba, Angel X. Chang, Zsolt Kira, Vladlen Koltun, Jitendra Malik, Manolis Savva, and Dhruv Batra (2021). "Habitat 2.0: Training Home Assistants to Rearrange their Habitat". In: *CoRR* abs/2106.14405. arXiv: 2106.14405. URL: https://arxiv.org/abs/2106.14405 (cit. on p. 40).

Tamar, Aviv, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel (2017). *Value Iteration Networks*. arXiv: 1602.02867 [cs.AI] (cit. on pp. 52, 95).

Tan, Jie, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke (2018). "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots". In: *Robotics: Science and Systems* (cit. on pp. 63, 79).

Tang, Yunhao and Shipra Agrawal (2020). *Discretizing Continuous Action Space for On-Policy Optimization*. arXiv: 1901.10500 [cs.LG] (cit. on p. 15).

The CVPR 2021 Multi-Object Navigation Challenge (2021). http://multion-challenge.cs.sfu.ca/2021.html (cit. on p. 85).

Thrun, Sebastian, Wolfram Burgard, and Dieter Fox (2005). *Probabilistic Robotics*. MIT Press (cit. on pp. 30, 64, 80, 94, 100).

Thrun, Sebastian, Yufeng Liu, Daphne Koller, Andrew Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte (July 2004). "Simultaneous Localization and Mapping with Sparse Extended Information Filters". In: *I. J. Robotic Res.* 23, pp. 693–716 (cit. on p. 37).

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "MuJoCo: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033 (cit. on p. 39).

Toromanoff, Marin, Emilie Wirbel, and Fabien Moutarde (2020). "End-to-End Model-Free Reinforcement Learning for Urban Driving Using Implicit Affordances". In: pp. 7151–7160 (cit. on p. 5).

Triggs, Bill, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon (1999). "Bundle Adjustment - A Modern Synthesis". In: *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*. ICCV '99. Berlin, Heidelberg: Springer-Verlag, 298–372 (cit. on p. 34).

Truong, Joanne, Sonia Chernova, and Dhruv Batra (2021a). "Bi-Directional Domain Adaptation for Sim2Real Transfer of Embodied Navigation Agents". In: *IEEE Robotics and Automation Letters* 6.2, pp. 2634–2641. URL: https://doi.org/10.1109%2Flra.2021.3062303 (cit. on p. 6).

Truong, Joanne, Sonia Chernova, and Dhruv Batra (2021b). "Bi-directional Domain Adaptation for Sim2Real Transfer of Embodied Navigation Agents". In: *IEEE Robotics and Automation Letters* 6.2 (cit. on pp. 63, 79).

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is All you Need". In: *NeurIPS* (cit. on pp. 4, 95).

Vondrus, Valdimir and Contributions (2020). *Magnum*. URL: http://magnum.graphics (cit. on p. 39).

Walter, Matthew R., Ryan M. Eustice, and John J. Leonard (2007). "Exactly Sparse Extended Information Filters for Feature-based SLAM". In: *The International Journal of Robotics Research* 26, pp. 335 –359 (cit. on p. 37).

Wani, Saim, Shivansh Patel, Unnat Jain, Angel X. Chang, and Manolis Savva (2020). "MultiON: Benchmarking Semantic Map Memory using Multi-Object

Navigation". In: *NeurIPS* (cit. on pp. xi, 9, 18, 32, 43, 45, 60, 75–78, 80, 84, 86, 100).

Wijmans, Erik, Abhishek Kadian, Ari Morcos, Stefan Lee, Irfan Essa, Devi Parikh, Manolis Savva, and Dhruv Batra (2019). "Decentralized Distributed PPO: Solving PointGoal Navigation". In: *CoRR* abs/1911.00357. arXiv: 1911.00357. URL: http://arxiv.org/abs/1911.00357 (cit. on pp. 5, 28, 46).

Williams, Ronald J. (2004). "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning". In: *Machine Learning* 8, pp. 229–256 (cit. on p. 27).

Williams, Stefan, Gamini Dissanayake, and Hugh Durrant-Whyte (Feb. 2002). "Towards multi-vehicle simultaneous localisation and mapping". In: vol. 3, pp. 2743 –2748 (cit. on p. 37).

Xia, Fei, Amir R. Zamir, Zhiyang He, Alexander Sax, Jitendra Malik, and Silvio Savarese (2018). "Gibson Env: real-world perception for embodied agents". In: (cit. on pp. 5, 39, 40, 62, 67, 92, 99).

Xia, Fei, William B Shen, Chengshu Li, Priya Kasimbeg, Micael Edmond Tchapmi, Alexander Toshev, Roberto Martín-Martín, and Silvio Savarese (2020). "Interactive Gibson Benchmark: A Benchmark for Interactive Navigation in Cluttered Environments". In: *IEEE Robotics and Automation Letters* 5.2, pp. 713–720 (cit. on p. 39).

Xiang, Fanbo, Yuzhe Qin, Kaichun Mo, Yikuan Xia, Hao Zhu, Fangchen Liu, Minghua Liu, Hanxiao Jiang, Yifu Yuan, He Wang, Li Yi, Angel X. Chang, Leonidas J. Guibas, and Hao Su (2020). "SAPIEN: A SimulAted Part-based Interactive ENvironment". In: *CoRR* abs/2003.08515. arXiv: 2003.08515. URL: https://arxiv.org/abs/2003.08515 (cit. on p. 39).

Yadav, Karmesh, Ram Ramrakhya, Arjun Majumdar, Vincent-Pierre Berges, Sachit Kuhar, Dhruv Batra, Alexei Baevski, and Oleksandr Maksymets (2022). *Offline Visual Representation Learning for Embodied Navigation*. arXiv: 2204.13226 [cs.CV] (cit. on p. 42).

Yamauchi, B. (1997). "A frontier-based approach for autonomous exploration". In: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pp. 146–151 (cit. on p. 55).

Yan, Claudia, Dipendra Kumar Misra, Andrew Bennett, Aaron Walsman, Yonatan Bisk, and Yoav Artzi (2018). "CHALET: Cornell House Agent Learning Environment". In: *CoRR* abs/1801.07357. arXiv: 1801.07357. URL: http://arxiv.org/abs/1801.07357 (cit. on pp. 39, 40).

Yao, Junfeng, Chao Lin, Xiaobiao Xie, Andy Ju An Wang, and Chih-Cheng Hung (2010). "Path Planning for Virtual Human Motion Using Improved A* Star Algorithm". In: *2010 Seventh International Conference on Information Technology: New Generations*, pp. 1154–1158 (cit. on p. 23).

Ye, Joel, Dhruv Batra, Abhishek Das, and Erik Wijmans (2021). "Auxiliary Tasks and Exploration Enable ObjectNav". In: *CoRR* abs/2104.04112. arXiv: 2104.04112. URL: https://arxiv.org/abs/2104.04112 (cit. on p. 46).

Ye, Joel, Dhruv Batra, Erik Wijmans, and Abhishek Das (2020). "Auxiliary Tasks Speed Up Learning PointGoal Navigation". In: *CoRR* abs/2007.04561. arXiv: 2007.04561. URL: https://arxiv.org/abs/2007.04561 (cit. on p. 46).

Yen-Chen, Lin, Pete Florence, Jonathan T. Barron, Alberto Rodriguez, Phillip Isola, and Tsung-Yi Lin (2021). "iNeRF: Inverting Neural Radiance Fields for Pose Estimation". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (cit. on p. 109).

Yijing, Wang, Liu Zhengxuan, Zuo Zhiqiang, and Ll Zheng (2018). "Local Path Planning of Autonomous Vehicles Based on A* Algorithm with Equal-Step Sampling". In: *2018 37th Chinese Control Conference (CCC)*, pp. 7828–7833 (cit. on p. 23).

Yonetani, Ryo, Tatsunori Taniai, Mohammadamin Barekatain, Mai Nishimura, and Asako Kanezaki (2021). "Path Planning using Neural A* Search". In: *ICML* (cit. on p. 95).

Zeiler, Matthew D and Rob Fergus (2013). *Visualizing and Understanding Convolutional Networks*. arXiv: 1311.2901 [cs.CV] (cit. on p. 63).

Zeng, Kuo-Hao, Roozbeh Mottaghi, Luca Weihs, and Ali Farhadi (2020). *Visual Reaction: Learning to Play Catch with Your Drone*. arXiv: 1912.02155 [cs.CV] (cit. on p. 95).

Zhai, Albert J. and Shenlong Wang (2022). *PEANUT: Predicting and Navigating to Unseen Targets*. arXiv: 2212.02497 [cs.CV] (cit. on pp. 55–57, 109).

Zhang, Fangyi, Jürgen Leitner, Zongyuan Ge, Michael Milford, and Peter Corke (2019). "Adversarial discriminative sim-to-real transfer of visuo-motor policies". In: *Int. J. Robotics Res.* 38.10-11 (cit. on pp. 63, 79).

Zhang, Jingwei, Lei Tai, Ming Liu, Joschka Boedecker, and Wolfram Burgard (2020). *Neural SLAM: Learning to Explore with External Memory*. arXiv: 1706.09520 [cs.LG] (cit. on p. 44).

Zhang, Zhanying and Zi xiang Zhao (2014). "A Multiple Mobile Robots Path planning Algorithm Based on A-star and Dijkstra Algorithm". In: *International Journal of Smart Home* 8, pp. 75–86 (cit. on p. 23).

Zhao, Jiang, Jiaming Sun, Zhihao Cai, Longhong Wang, and Yingxun Wang (2021). "End-to-End Deep Reinforcement Learning for Image-Based UAV Autonomous Control". In: *Applied Sciences* 11.18 (cit. on p. 5).

Zhi, Shuaifeng, Tristan Laidlow, Stefan Leutenegger, and Andrew J. Davison (2021). *In-Place Scene Labelling and Understanding with Implicit Scene Representation*. arXiv: 2103.15875 [cs.CV] (cit. on p. 110).

Zhu, Alex Zihao, Liangzhe Yuan, Kenneth Chaney, and Kostas Daniilidis (2017). *Unsupervised Event-based Learning of Optical Flow, Depth, and Egomotion*. arXiv: 1812.08156 [cs.CV] (cit. on p. 52).

Zhu, Xinge, Jiangmiao Pang, Ceyuan Yang, Jianping Shi, and Dahua Lin (2019). "Adapting Object Detectors via Selective Cross-Domain Alignment". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 687–696 (cit. on pp. 63, 79).

Zhu, Yuke, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi (2016). *Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning*. arXiv: 1609.05143 [cs.CV] (cit. on pp. 5, 17, 41).

FOLIO ADMINISTRATIF

THESE DE L'INSA LYON, MEMBRE DE L'UNIVERSITE DE LYON

NOM : SADEK
Prénoms : Assem

DATE de SOUTENANCE : 31/01/2024

TITRE : M.

NATURE : Doctorat

Numéro d'ordre : 2024ISAL0013

Ecole doctorale : INFOMATH

Spécialité : Informatique

RESUME :
Les progrès récents de l'IA, et plus particulièrement de l'apprentissage automatique, permettent aux robots de s'intégrer de manière plus transparente dans nos habitudes quotidiennes. L'objectif de cette thèse est de faire un pas de plus vers le développement d'agents autonomes intelligents qui peuvent être intégrés dans notre environnement quotidien, comme les maisons, les hôpitaux, les centres commerciaux, etc. Ces agents devraient posséder la capacité de naviguer efficacement dans leur environnement pour atteindre un certain objectif, comme atteindre une certaine zone de l'environnement ou trouver un certain objet. C'est pourquoi nous examinons le large éventail de techniques existantes pour la construction d'un agent de navigation incarné. Ces techniques peuvent entièrement être apprises par des réseaux neuronaux (techniques basées sur l'apprentissage) ou elles peuvent être des techniques fondées sur la géométrie qui reposent sur une modélisation explicite de l'agent et de son environnement. Dans cette thèse, nous construisons des approches hybrides qui utilisent les deux techniques afin de pouvoir fonctionner, non seulement dans une simulation, mais également dans un environnement physique réel. Il s'agit d'un objectif commun dans toutes les contributions de cette thèse.

MOTS-CLÉS : Robotics, Deep Learning

Laboratoire (s) de recherche : LIRIS

Directeur de thèse: M. Atilla BASKURT

Président de jury : MME. Marie BABEL

Composition du jury : M. David Filliat (Rapporteur), M. Fabien MOUTARDE (Rapporteur), M. Boris CHIDLOVSKII (Co-Directeur de thèse), Christian WOLF (Co-Directeur de thèse)