Learning spatial representations for single-task navigation and multi-task policies

Pierre Marza

N° d'ordre NNT : 2024ISAL0105

Thèse de Doctorat de l'INSA LYON,
membre de l'Université de Lyon

École doctorale n° 512
Informatique et Mathématiques de Lyon (Infomaths)

Spécialité de doctorat :
Informatique

Soutenue publiquement le 25 Novembre 2024, par :
Pierre Marza

Learning spatial representations for single-task navigation and multi-task policies

---

Devant le jury composé de :

Laptev	Ivan	Directeur de Recherche INRIA Paris / MBZUAI	Rapporteur
Alahari	Karteek	Directeur de Recherche INRIA Grenoble	Rapporteur
Thome	Nicolas	Professeur des Universités Sorbonne Université	Examinateur
Chalvatzaki	Georgia	Full Professor TU Darmstadt	Examinatrice
Matignon	Laëtitia	Maître de Conférences Université Claude Bernard Lyon 1	Co-Directrice de thèse
Simonin	Olivier	Professeur des Universités, INSA Lyon	Co-Directeur de thèse
Wolf	Christian	Principal Scientist Naver Labs Europe	Co-Directeur de thèse

Département FEDORA – INSA Lyon - Ecoles Doctorales

SIGLE	ECOLE DOCTORALE	NOM ET COORDONNEES DU RESPONSABLE
ED 206 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
ED 341 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 e2m2.codir@listes.univ-lyon1.fr
ED 205 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** Laboratoire 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
ED 34 EDML	**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
ED 160 EEA	**ÉLECTRONIQUE, ÉLECTROTECHNIQUE, AUTOMATIQUE** https://edeea.universite-lyon.fr Sec. : Philomène TRECOURT 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
ED 512 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. Dragos IFTIMIE** Université Claude Bernard Lyon 1 Bâtiment Braconnier (ex-101) 21 Avenue Claude Bernard 69622 VILLEURBANNE Cedex Tél : 04.72.44.79.58 direction.infomaths@listes.univ-lyon1.fr
ED 162 MEGA	**MÉCANIQUE, ÉNERGÉTIQUE, GÉNIE CIVIL, ACOUSTIQUE** http://edmega.universite-lyon.fr Sec. : Philomène TRECOURT Tél : 04.72.43.71.70 Bâtiment Direction INSA Lyon mega@insa-lyon.fr	**M. Etienne PARIZET** INSA Lyon Laboratoire LVA Bâtiment St. Exupéry 25 bis av. Jean Capelle 69621 Villeurbanne CEDEX etienne.parizet@insa-lyon.fr
ED 483 ScSo	**ScSo**[1] https://edsciencessociales.universite-lyon.fr Sec. : Mélina FAVETON Tél : 04.78.69.77.79 melina.faveton@univ-lyon2.fr	**M. Bruno MILLY** (INSA : J.Y. TOUSSAINT) Univ. Lyon 2 Campus Berges du Rhône 18, quai Claude Bernard 69365 LYON CEDEX 07 Bureau BEL 319 bruno.milly@univ-lyon2.fr

[1] ScSo : Histoire, Géographie, Aménagement, Urbanisme, Archéologie, Science politique, Sociologie, Anthropologie

Abstract

Autonomously behaving in the 3D world requires a large set of skills, among which are perceiving the surrounding environment, representing it precisely and efficiently enough to keep track of the past, making decisions and acting to achieve specified goals. Animals, for instance humans, stand out by their robustness when it comes to acting in the world. In particular, they can efficiently generalize to new environments, but are also able to rapidly master many tasks of interest from a few examples. This manuscript will study how artificial neural networks can be trained to acquire a subset of these abilities.

We will first focus on training neural agents to perform semantic mapping, both from augmented supervision signal and with proposed neural-based scene representations. Neural agents are often trained with Reinforcement Learning (RL) from a sparse reward signal. Guiding the learning of scene mapping abilities by augmenting the vanilla RL supervision signal with auxiliary spatial reasoning tasks will help navigating efficiently. Instead of modifying the training signal of neural agents, we will also see how incorporating specific neural-based representations of semantics and geometry within the architecture of the agent can help improve performance in goal-driven navigation. Then, we will study how to best explore a 3D environment in order to build neural representations of space that are as satisfying as possible based on robotic-oriented metrics we will propose. Finally, we will move from navigation-only to multi-task agents, and see how important it is to tailor visual features from sensor observations to the task at hand to perform a wide variety of tasks, but also to adapt to new unknown tasks from a few demonstrations.

This manuscript will thus address different important questions such as: *How to represent a 3D scene and keep track of previous experience in an environment? – How to robustly adapt to new environments, scenarios, and potentially new tasks? – How to train agents on long-horizon sequential tasks? – How to jointly master all required sub-skills? – What is the importance of perception in robotics?*

Résumé

Agir de manière autonome dans notre monde 3D requiert un large éventail de compétences, parmi lesquelles se trouvent la perception du milieu environnant, sa représentation précise et suffisamment efficace pour garder une trace du passé, la prise de décisions et l'action en vue d'atteindre des objectifs. Les animaux, par exemple les humains, se distinguent par leur robustesse lorsqu'il s'agit d'agir dans le monde. En particulier, ils savent s'adapter efficacement à de nouveaux environnements, mais sont aussi capables de maîtriser rapidement de nombreuses tâches à partir de quelques exemples. Ce manuscrit étudiera comment les réseaux neuronaux artificiels peuvent être entrainés pour acquérir un sous-ensemble de ces capacités.

Nous nous concentrerons tout d'abord sur l'entrainement d'agents neuronaux à la cartographie sémantique, à la fois à partir d'un signal de supervision augmenté et avec des représentations neuronales de scènes. Les agents neuronaux sont souvent entrainés par apprentissage par renforcement (RL) à partir d'un signal de récompense peu dense. Guider l'apprentissage des capacités de cartographie d'une scène en ajoutant au signal de supervision des tâches auxiliaires favorisant le raisonnement spatial aidera à naviguer plus efficacement. Au lieu de travailler sur le signal d'entrainement des agents neuronaux, nous verrons également comment l'incorporation de représentations neuronales spécifiques de la sémantique et de la géométrie à l'architecture de l'agent peut contribuer à améliorer les performances de navigation sémantique. Ensuite, nous étudierons la meilleure façon d'explorer un environnement 3D afin de construire des représentations neuronales de l'espace qui soient aussi satisfaisantes que possible sur la base de métriques pensées pour la robotique que nous proposerons. Enfin, nous passerons d'agents de navigation à des agents multi-tâches et nous verrons à quel point il est important d'adapter les caractéristiques visuelles extraites des observations de capteurs à chaque tâche à accomplir pour réaliser une variété de tâches, mais aussi pour s'adapter à de nouvelles tâches inconnues à partir de quelques démonstrations.

Ce manuscrit abordera donc différentes questions : *Comment représenter une scène 3D et garder une trace de l'expérience passée dans un environnement ? – Comment s'adapter de manière robuste à de nouveaux environnements, scénarios et potentiellement de nouvelles tâches ? – Comment entrainer des agents à des tâches séquentielles à horizon long ? – Comment maîtriser conjointement toutes les sous-compétences requises ? – Quelle est l'importance de la perception en robotique ?*

Acknowledgements

Acknowledgements in French

Je tiens à remercier tout d'abord mes encadrants, Laëtitia, Olivier et Christian. Je garderai de bons souvenirs de nos réunions, discussions au sujet de nouvelles idées, articles scientifiques et résultats d'expériences. Vous m'avez apporté une vraie stimulation scientifique et m'avez permis d'apprendre beaucoup. Mention spéciale à Christian que j'ai recontré lors d'un projet en dernière année d'école d'ingénieur, et avec qui j'ai poursuivi pour un stage de fin d'étude et finalement... une thèse ! Merci Christian pour ta bienveillance et ton enthousiasme, et surtout ton implication scientifique sans faille. Même si toutes nos idées n'ont pas toujours aboûti, j'ai pris beaucoup de plaisir à les imaginer, les discuter avec toi et les implémenter au cours de ces dernières années.

Je remercie également les membres de mon jury, Ivan Laptev, Karteek Alahari, Nicolas Thome et Georgia Chalvatzaki pour leurs questions et retours constructifs.

Merci au LIRIS et au CITI de m'avoir accueilli durant cette thèse et de m'avoir permis de côtoyer des gens formidables. Côté LIRIS, merci à Corentin (avec qui j'ai aussi eu la chance de collaborer durant mon stage de fin d'étude d'école d'ingénieur – j'en profite donc pour glisser un *merci* à Moez et Grigory qui m'ont co-encadrés durant ce stage et aux côtés de qui, avec Corentin, j'ai beaucoup appris sur le travail de recherche !), Théo, Quentin, Steeven, Edward, Assem, Guillaume, Olivier, Eric, Aurélien et Juliana dont j'ai croisé la route. Côté CITI, jai eu la chance de rencontrer Antoine, Théotime, Benoit, Benjamin, Florian, Simon, Idham, Aurélien, Johan, Alessandro, Romain, Damien, Thierry, Alix, Maxime, Guillaume, et bien d'autres, avec qui j'ai pu discuter de nombreux sujets passionnants. Mention spéciale à Xiao avec qui je me suis entrainé pendant plusieurs mois pour courir notre premier marathon (*Run in Lyon 2023*). Je me souviendrai surtout de nos *footings* au *Parc de la Tête d'Or* et des discussions qui les accompagnaient !

Et bien sûr, pour finir, un grand *merci* à ma famille, en particulier mes parents et ma soeur pour leur soutien. Un exemple de celui-ci: de temps en temps pendant la thèse, je recevais un petit *meme* sur le quotidien des doctorants de la part de ma soeur me rappelant que les périodes difficiles existent et doivent être vécues avec le sourire ! Ceci n'est qu'un exemple

parmi d'autres de l'impact positif que vous avez tous les trois eu, merci ! J'ai également partagé de bons moments avec mes amis pendant ces dernières années. Mention spéciale à Josué pour nos longues discussions, Hugo pour les sorties sportives (et l'intérêt que tu as porté à ce manuscrit !), et Anthony qui est venu me rendre visite en Californie durant mon stage de thèse !

Acknowledgements in English

I first wanted to thank the Computer Vision team at Huawei Noah's Ark Lab in London where I interned in 2019. This was my first experience with research and the work I conducted with people in the team, among which were Sean, Greg, Sarah and Steven, made me realise I really enjoyed research and wanted to pursue a PhD.

This time, during my PhD, I interned in the Embodied AI team at Meta AI in Summer 2022. I would like to thank all members of the team for making this internship a great experience, and in particular Dhruv and Devendra who made it happen. Devendra, I enjoyed our discussions about research and other topics, thank you! I also had a lot of fun spending time with other interns such as Jacob, Theo, Nicklas, Claire and Corentin, so thanks to you all!

Contents

List of Figures

List of Tables

Acronyms

AI	Artificial Intelligence
BC	Behavior Cloning
CNN	Convolutional Neural Network
CV	Computer Vision
DRL	Deep Reinforcement Learning
GRU	Gated Recurrent Unit
LSTM	Long-Short Term Memory
MAE	Masked Auto-Encoding
MDP	Markov Decision Process
MLP	Multi-Layer Perceptron
NLP	Natural Language Processing
POMDP	Partially Observable Markov Decision Process
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
ViT	Vision Transformer

Chapter

Introduction

Perceiving the world and acting in it are key abilities of any autonomous entity. More importantly, learning to master them robustly, i.e. adapting to changes in the surroundings, is necessary to accomplish any goal of interest. In this manuscript, we will first study how to train artificial neural networks to act in unknown 3D environments, more specifically navigate efficiently thanks to scene mapping strategies to keep track of explored areas and important landmarks. We will also consider the adaptability to multiple tasks involving visuomotor control by modulating visual features extracted from observations. The conducted work spans the topics of *Computer Vision*, *Reinforcement Learning*, *Robotics*, and the more recent field of *Embodied AI* that we will describe in more detail later.

Now that we have presented a high-level view of what will be discussed in this manuscript, let us take a step back and consider a larger picture that goes beyond the scope of the studies that will be presented but helps motivate the followed research direction. We will come back to the precise questions we will ask ourselves in the last two sections of this chapter, but before this, we will see why the ability to autonomously interact with the world is a relevant objective when targeting some form of artificial intelligence, and also why it might be interesting to address the sub-problems of perception and action together at the same time. For now, we will start by defining two general concepts: *agent* and *skill*.

Agent: *an entity able to take actions in a given environment, and have effects on it.*

A powerful abstraction that will often be used in this manuscript is the one of the *agent*. We will consider specific implementations of agents such as *policies* in Reinforcement Learning

Figure 1.1: **Illustration of the *coffee dilemma*** – generated by a Deep Learning model (Podell et al., 2024). This is not presented as an illustration only, but will serve another purpose: one should pay attention to the wrong details in these images to get a hint at how hard modeling the physical world can be for current Artificial Intelligence (AI) models.

(RL), or *robots* in Robotics, but we will often refer to *agents* to keep drawn conclusions general enough.

Skill: *the ability for an agent to transform an input into a relevant output.*

As for the agent, we keep the definition of a *skill* quite general as it will depend on the considered tasks and scopes of the studies. However, what is important is that an agent, to successfully interact with a 3d environment, will have to combine different skills together (sometimes referred to as *sub-skills* later in this manuscript), each one providing partial information to solve a task. Examples of skills could be detecting obstacles on a visual input, planning a high-level path to a location from a representation of an environment, detecting important semantic objects, or predicting a sequence of local actions to reach a high-level target.

Let us now consider a simple thought experiment that we will call the *coffee dilemma* (see Figure 1.1 for an illustration): you arrived yesterday at a house where you will spend your holidays. You have already been here for a day, so you start being familiar with the place. This morning, you need a coffee! You have two options: (i) either prepare it yourself, or (ii) make a bet with your friend – if you beat them at chess they will bring you a coffee, if not you will not have a coffee this morning. Importantly, this friend is actually a professional chess player. Most people will probably pick option (i) as it appears much easier: you would just need to grab a cup and make coffee with the coffee machine, while option (ii) seems out of reach for most humans.

However, our current AI agents, also called *neural models*, if tasked to get a coffee as an instruction, might rather pick option (ii). After all, some models have already achieved chess professional-level performance in the past (Campbell et al., 2002; Silver et al., 2017; Ruoss et al., 2024). And if we really think about it, option (i) might not be as easy as we might assume in the first place. Re-visiting the well-known *Moravec's Paradox* (Moravec, 1988a), stating that tasks that seem the simplest to humans are often the most difficult to solve by algorithms, one should not underestimate the difficulty of getting a cup of coffee in the morning. Indeed, an agent should first remember the location of cups, coffee, and the coffee machine. It then has to navigate to these different landmarks in the right order (first to the cup and coffee before the coffee machine) while avoiding obstacles and following an efficient path for the task to be solved in a reasonable amount of time. Another hard part is about precisely manipulating all considered objects and potential containers. Thus, making coffee requires properly perceiving its surrounding environment (vision), planning navigation and manipulation paths (high-level reasoning), and executing the paths (low-level control). One should also not underestimate the amazing human haptics that an autonomous agent might need to emulate, i.e. sensing and manipulating through touch, requiring an accurate model of physical surface contact interactions. Even if we, humans, do most of these things unconsciously, it is very challenging for an AI model to perform. More importantly, it is even harder to do it reliably, in any house, with any lighting condition, any type of coffee machine, and any shape of cup. This thus motivates the interest in the research direction followed in our work.

Current AI models can now generate stunning images (Figure 1.1), that most humans would not be able to create from scratch, and yet are far from solving robotic tasks. In fact, if you actually look at the details of generated images in Figure 1.1, you might get a hint at why we are still far from having agents interacting with the world. Many errors, even if tiny, reveal something important: these generative models have a hard time building a robust model of the physical world, which is primordial to then act in the world. However, it is important to acknowledge that such generative models are not trained to model physics and dynamics explicitly, but we could assume that such implicit understanding would be necessary to generate convincing images. Is it just a question of supervision, or could it be due to them not being able to take actions in an interactive environment? These are only open questions, but we will come back to them later.

1.1 Autonomy in the world as a measure of intelligence

Intelligence is still considered today a controversial notion (Legg et al., 2007). In this manuscript, we will not propose another definition of this concept, but will instead think about a fuzzy version of it from a specific point of view, i.e. the ability to behave autonomously in human-like environments. Indeed, the world we live in is highly unstructured and full of different variants of given concepts. Being able to navigate such environments thus constitutes

a relevant test-bed to some form of intelligence, that might not be considered proper general intelligence but comes close to a subset of human capabilities, mainly the ability to interact with the physical world and generalize to new or poorly represented scenarios and concepts.

This manuscript studies autonomy through the lens of algorithms, more specifically Machine Learning approaches where artificial neural networks are trained to address different tasks of interest. We consider different design choices, both in terms of neural architectures and supervision signal, that can lead to autonomous agents taking actions in 3D environments. In this context, learning to behave in scenes that would look realistic to a human has another interest: indeed, a great example of a neural network being able to robustly generalize to new scenarios is the human brain, and its development was likely influenced by the types of tasks and environments human beings have to deal with. Some contributions presented in this manuscript are indeed inspired by practices or theories from the field of neuroscience.

Properly behaving in realistic environments requires two main components: *perception*, the ability to extract information from high-dimensional sensory data, and *agency*, the capacity to take correct actions influencing the world to reach goals of interest. We will consider how the former is necessary to construct structured representations of the surrounding environment, but is not enough alone without the latter, as perception and agency are tightly related.

1.2 Perceiving the world is crucial but not enough alone

Perception is the first step towards understanding the world by extracting the underlying structure from high-dimensional signals coming from the surroundings. Even if not restricted to it, this work mainly considers visual perception as we study how autonomous agents can behave from visual information as input. This choice is explained by the richness of the visual modality, from which a lot of information can be recovered. Other modalities such as language, proprioception, radar, lidar, or touch are obviously useful and very likely necessary for autonomous agents to behave interestingly, but we believe vision is a modality we can hardly operate without.

Visual perception with algorithms is often referred to as Computer Vision (CV), which is an active field of research, trying to answer a fundamental question: *How to extract rich information from low-level raw visual data?* One of the first references to CV was the *Summer Vision Project* in 1966 (Papert, 1966), where a group of researchers from the Massachusetts Institute of Technology (MIT) decided to spend a summer working on a computer-based vision system able to distinguish objects from the background, and from what they mentioned as "chaos". This work was full of great ideas, such as dividing this hard goal into easier sub-problems, or studying shapes and surface properties, but needless to say one summer was not enough to obtain a robust object detector. This shows there is a gap between our own evaluation (even as researchers!) of the difficulty of providing human-level vision to algorithms and what it

really takes. Indeed, only today, 58 years after the end of the *Summer Vision Project*, we start reaching strong object detection performance, with, however, a lack of robustness to long-tail scenarios. The road is still long before reaching human-level generalization in vision systems, even if we have witnessed some amazing progress in the field during the past few years.

CV has evolved since the *Summer Vision Project* with a growing number of challenging sub-problems being proposed and new solutions to them, relying more and more on artificial neural networks being trained to extract visual information. A more thorough review of the literature will be given in chapters 2 and 3, but an important characteristic of many CV problems, already during the *Summer Vision Project* and in more recent tasks, is that perception is considered separately from action. We often view our vision algorithm as a passive observer of the world without any effect on it. CV models will be trained to understand the content of images from a fixed set of data, without the ability to interact with the world.

However, in the past few years, a subdomain known as *Embodied AI* has started to emerge in the CV community, where agents learn to process visual information by interacting with environments. Such considerations are not new, and this appears more as a convergence with other fields such as *Robotics* or *Reinforcement Learning*. Indeed, the long-term objective appears to be similar to the one of robotics, i.e. building autonomous embodied agents able to interact with the world, but the difference might rather be about the starting point. The Embodied AI community finds its roots in Computer Vision and Deep Learning fields and approaches the problem from a learning-focused point of view. While robotics has been incorporating learning-based components, Embodied AI is motivated by the goal of training agents on large-scale datasets to allow the implicit emergence of required sub-skills. Despite the introduction of robotics-related inductive biases, e.g. explicit environment mapping, the *scaling hypothesis* is also heavily studied, i.e. whether all skills can be learned end-to-end when scaling neural architecture complexity and data size. Similar to what has been done recently in Natural Language Processing (NLP) with neural network approaches trained from large-scale text databases, a part of the Embodied AI community believes the same path can be followed to get autonomous agents acting in the physical 3D world.

We will now discuss why training perception and action simultaneously, or at least taking each one into account when considering the other, might be important.

1.3 Knowledge arises from interaction

We must perceive in order to move, but we must also move in order to perceive. – Gibson (1979)

As stated by Gibson and studied through the lens of his theory of affordances (Gibson, 1966), perception and action are tightly related. This strong connection has indeed been an important research subject in psychology with other contributions such as the ideomotor theory of William James (James et al., 1890) stating that specific actions are taken by humans as

they anticipate the sensory signal they will receive based on their behavior. These concepts have then been re-used in cognitive sciences, artificial intelligence, and robotics to better understand the behavior of autonomous agents perceiving their environment and interacting with it.

According to Gibson, navigating the world smoothly requires perceiving it, but at the same time, being unable to act would greatly reduce the perceptual diversity of one's experience. Such a statement could seem wrong in the era of the internet: a lot of data from everywhere in the world, under diverse conditions is stored and available. This is particularly important when considering learning algorithms. However, again, what would be lacking here is the ability to intentionally explore the surrounding environment to look for specific data. The web is large but is still static.

The importance of interacting with the world to understand and improve perception is also motivated when considering two timelines: the one of human evolution, and the one of a single human lifetime. As done by Jitendra Malik in some of the great talks he gave (e.g. *The Sensorimotor Road to Artificial Intelligence*, *Martin Meyerson Berkeley Faculty Research Lecture*, March 2023), looking back at our evolutionary timeline is quite informative. A particularly important moment is known as the *Cambrian explosion*, i.e. a central period in evolution when many developed species started to emerge with diverse specificities and abilities. If focusing on the *Hominidae* family, this was the period when bipedalism and opposable thumbs appeared, allowing one to move and use tools more efficiently. An important part of evolution from the first living organisms to current human beings could thus be seen as a search for the necessary abilities to adapt to the surrounding environment, as this is crucial for survival. This is particularly true for perception: our perceptual system has evolved to help us act more efficiently in the world. Indeed, Parker (2003) studied how vision played an important role in the Cambrian explosion. Now, even considering the lifetime of a human, it always starts with hands-on interaction. This is indeed what childhood is all about: right before being able to read and thus access knowledge this way, children spend most of their time playing, i.e. manipulating objects and moving around to understand the core concepts of the world they behave in (Thelen and Smith, 1994; Thelen et al., 2001; Smith and Gasser, 2005). The underlying motivations for this behavior are not yet clearly understood, but the concept of intrinsic curiosity is often mentioned and has been studied in algorithms (Gottlieb et al., 2013; Oudeyer and Kaplan, 2007; Oudeyer et al., 2007; Pathak et al., 2017).

Neuroscience also shows there is a strong relationship between perception and action through the existence of bottom-up and top-down processes in our visual system (Corbetta and Shulman, 2002; Buschman and Miller, 2007). The first ones, bottom-up processes, were shown to extract all information from available visual signals, while the second, top-down processes, seem to adapt perception to the task at hand. It seems that the way we perceive is impacted by our prior knowledge and the task we want to solve. Indeed, adapting the

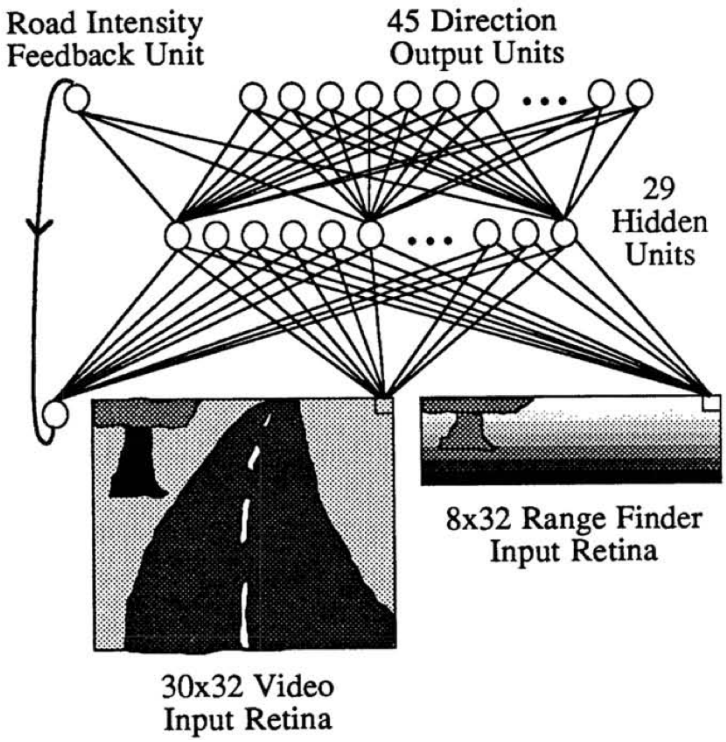


Figure 1.2: **ALVINN architecture** – reproduced from Pomerleau (1988)

structure of the information we extract to the problem we are trying to solve appears as a relevant approach. Some tasks might rather rely on semantics, others on 3D geometry for example.

When designing learning algorithms, using interaction as a test-bed to improve perception thus appears as an appealing direction. Embodied AI, bridging CV and robotics, with a specific focus on learning from egocentric interactions, then becomes a great framework to ask many important questions that we will present in the next section.

1.4 Artificial neural agents can learn to interact

Training neural networks to navigate the physical world is not that new. Already back in 1988, ALVINN (*Autonomous Land Vehicle In a Neural Network* – Pomerleau (1988)) was developed at Carnegie Mellon University. This system (Figure 1.2) was based on a fully end-to-end trained neural network taking images as inputs and outputting the direction a car should follow to stay on a road. What has changed since then are the new advances in the field of *Deep Learning*, with new architectures or supervision strategies. For example, we are now able to train deep neural networks composed of hundreds of layers while the one in ALVINN was only composed of three neural layers.

However, another important component when training artificial neural networks is the access to data. Great advances were made in NLP and CV when large-scale datasets were introduced (e.g. Imagenet (Deng et al., 2009) in CV). Accessing this amount of data in robotics

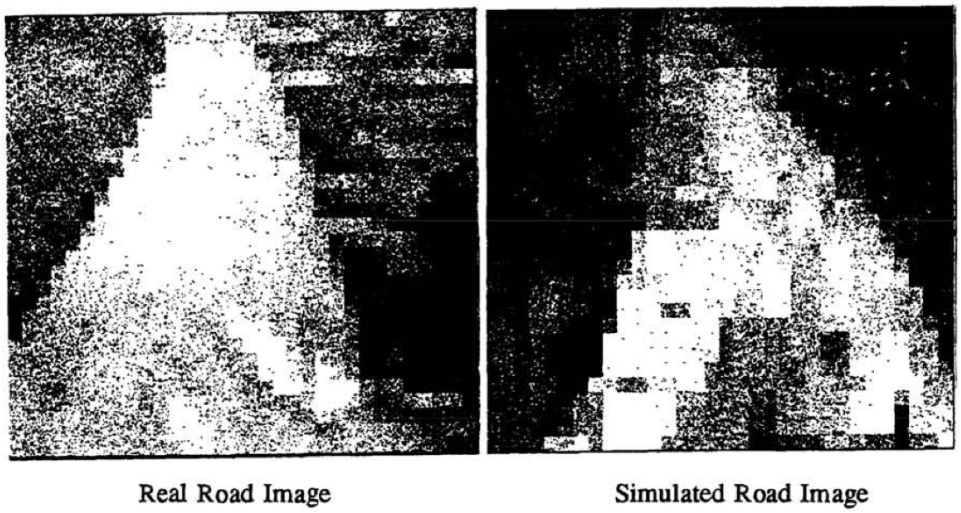


Figure 1.3: **ALVINN simulation** – reproduced from Pomerleau (1988)

or Embodied AI is however much harder, particularly if we want our systems to be trained by interacting with an environment. A great alternative is simulation, where we try to emulate a given environment, with different levels of realism (sensing, photo-realism, etc.). Recent improvements in Embodied AI systems, which will be reviewed in chapter 3, are partly due to the increase in simulation quality and dataset sizes, and new generations of neural models will likely benefit from further advances. Interestingly enough, simulation was already a topic in the ALVINN paper! Indeed, the neural network was trained from simulated images of the road (Figure 1.3). It is humbling to witness how prior work such as ALVINN was ahead of its time.

Let us now conclude this introduction by going back to our current times, and considering what challenges are still to be faced, some of which we address in this manuscript. Many questions indeed start to emerge when considering neural agents autonomously interacting with the world:

- *How to represent a 3D scene and keep track of previous experience in an environment?* When perceiving the world from an egocentric viewpoint, building specific representations allowing to interact with it is an important requirement. This is also true for keeping track of the past: previous experience, e.g. detected landmarks or specific geometry of the environment, can turn out to be very informative later on and should be stored efficiently. This manuscript (chapters 4 and 5) will study how to explicitly teach neural agents to keep track of important landmarks, and how to build neural-based representations that can be queried efficiently to retrieve semantic and geometric information of interest.

- *How to robustly adapt to new environments, scenarios, and potentially new tasks?* We could hardly consider an agent as *autonomous* if it was not able to address given tasks in new environments or even adapt to variations of known tasks. An important part of this

manuscript is thus about evaluating these generalization abilities: all neural agents will be trained to navigate in environments that are different from the ones seen at test time, in order to study how well they can generalize (chapters 4, 5 and 6). Generalization to new manipulation tasks will also be covered (chapter 7) as it is another important ability to have. Transferring acquired knowledge and sub-skills to new domains and tasks is indeed a great ability of living species and a challenging direction to pursue in AI.

- *How to train agents on long-horizon sequential tasks?* Most considered tasks in this manuscript require an agent to take many actions sequentially to be considered successful. This poses several challenges, both at inference and training time. When deploying a trained agent in a new scene, the length of sequences of actions to take will introduce memory challenges, i.e. properly remembering the past (cf. previous point) will become harder and yet more important than for shorter sequences. At training time, as we will often rely on RL, a well-known challenge will be the *credit assignment problem* (presented in more detail in the next chapter), i.e. the difficulty in identifying the actions or combinations of actions that had the strongest influence on a success or a failure. We will have to overcome the difficulty of training agents with RL on such long-horizon tasks from sparse rewards by following standard practices in the literature.

- *How to jointly master all required sub-skills?* Some parts of this manuscript (chapters 4 and 5) will deal with end-to-end neural agents trained with RL to navigate. In this case, many sub-skills have to be learned implicitly from reward alone: *perceiving the environment* (detecting landmarks and obstacles), *mapping the environment*, *remembering previous experience*, *mastering the available actions* to navigate properly. Learning all these abilities at the same time can be challenging, but also simplifies the problem of specifying necessary sub-skills as those are learned automatically given the task to solve. Another technique that will be considered in the manuscript (chapter 6) will be to decompose the policy into specific modules with only some of them being trained to perform a specific sub-task.

- *What is the importance of perception in robotics?* This manuscript will be mainly biased towards studying the impact of perception on different robotic tasks. We will indeed consider the relationships between perception-based representations of space and navigating agents (chapters 4, 5 and 6) along with the impact of task-specific visual features to address different tasks of interest based on a single pre-trained visual backbone (chapter 7).

These questions make the Embodied AI problem interesting and will be studied in this manuscript. We will now provide more details about the content of the next chapters, along with their associated contributions.

1.5 Organization of the manuscript and contributions

The manuscript is divided into eight chapters (including the current one). We will start with a literature review in chapters 2 and 3, before to present our contributions in chapters 4 - 7. Finally, chapter 8 concludes this manuscript and takes a step back from presented concepts, considering relevant future perspectives. A more detailed presentation of each chapter is given below:

- Chapter 2 introduces preliminary concepts the next chapters will build on top of. We will cover topics such as *Deep Learning*, *Computer Vision*, or *Sequential decision-making*. We will review common approaches in different subdomains, highlighting current trends and state-of-the-art methods and algorithms that will later be leveraged in this manuscript. More than listing concepts, we will try to mention which role they play in the context of the work presented later in this manuscript.

- Chapter 3 reviews recent progress in Embodied AI. Once the core concepts will have been presented in the previous chapter, it will be time to zoom in on the field of Embodied AI to present current advances and directions. More specifically, we will talk about *simulation* (datasets, simulators, tasks and benchmarks), methods to *encode past experience* (episodic memory, recurrent hidden state, topological or metric maps), different agent training methods such as *reinforcement learning*, *imitation learning*, *pre-trained visual backbones* helping the generalization to new tasks, or *auxiliary supervision* to guide the training of autonomous agents. We will also review the use of *implicit representations* in robotics. As done in the previous chapter, more than listing papers and approaches, we will try to position the work presented in the next chapters with respect to existing work.

- Chapter 4 studies the emergence of mapping abilities in RL-trained agents when augmenting their supervision signal with auxiliary tasks. We focus on training neural agents to navigate new 3D environments to explore them and detect landmarks whose location should be memorized to reach them later. In such a setting, vanilla RL-based approaches would leverage reward signals to incentivize the agent to implicitly learn to map the location of objects of interest. While this can work to some extent, we show in this chapter that explicitly training the agent to memorize the location of targets in addition to the original RL supervision signal improves navigation efficiency. More importantly, this finding transfers to different agent types, either fully neural or equipped with an explicit map.

> **Related contributions**
>
> **Paper**: *P. Marza, L. Matignon, O. Simonin, C. Wolf, Teaching Agents how to Map: Spatial Reasoning for Multi-Object Navigation, IROS 2022* (Marza et al., 2022)

Project page: https://pierremarza.github.io/projects/teaching_agents_how_to_map/
Code: https://github.com/PierreMarza/teaching_agents_how_to_map
Application: Winning entry in the *MultiON Challenge*, *Embodied AI Workshop*, *CVPR 2021* (http://multion-challenge.cs.sfu.ca/2021.html).

- Chapter 5 proposes to learn implicit representations of 3D scenes (geometry and semantics), but more importantly to train RL agents to use them to navigate. While focusing on the same task as the previous chapter, it will be about designing specific neural-based representations of an environment to be used by an agent instead of augmenting its supervision signal. We introduce two dynamically-updated representations of scenes, one mapping semantic objects, the other unexplored area and geometry of known parts of the environment. An important challenge is to efficiently integrate such representations in the forward pass of a neural agent so that it can be trained with RL to use them properly. We show that the proposed representations improve navigation efficiency at test time on new unknown scenes.

Related contributions
Paper: *P. Marza, L. Matignon, O. Simonin, C. Wolf, Multi-Object Navigation with dynamically learned neural implicit representations*, *ICCV 2023* (Marza et al., 2023)
Project page: https://pierremarza.github.io/projects/dynamic_implicit_representations/
Code: https://github.com/PierreMarza/dynamic_implicit_representations

- Chapter 6 presents a method to autonomously collect training data for neural implicit representations of 3D space (NeRF – will be presented in chapter 2) with modular agents. Implicit representations have been used to perform novel view synthesis in 3D scenes, but often require a manual acquisition of training data. Unlike the previous chapter that studies how to use implicit representations to help with navigation, we focus here on the opposite: how can neural agents navigate to autonomously collect data to build neural representations of scenes in a second stage? We also reflect on relevant robotics-related metrics to evaluate the underlying quality of an implicit scene representation and present a use-case of our method to scan a new scene and safely fine-tune a policy of interest in simulation.

Related contributions
Paper: *P. Marza, L. Matignon, O. Simonin, D. Batra, C. Wolf, D.S. Chaplot, AutoNeRF: Training Implicit Scene Representations with Autonomous Agents*, *IROS 2024* (Marza et al., 2024a)
Project page: https://pierremarza.github.io/projects/autonerf/
Code: https://github.com/PierreMarza/autonerf

- Chapter 7 considers the adaptation of visual features of pre-trained vision models con-

ditioned on a task to solve, and generalization to new tasks in the context of training robotic multitask policies. General vision models have been studied in different domains including robotics: a single versatile feature extractor that can be shared among tasks is indeed appealing. However, we assume in this chapter that visual features should be dependent on the task at hand. We introduce task-conditioned adapters to modulate visual features based on the downstream task and validate their interest experimentally. We additionally show that new task embeddings used to condition the proposed adapters can be optimized provided a few expert demonstrations, to address new tasks unseen at training time.

> **Related contributions**
>
> **Paper**: *P. Marza, L. Matignon, O. Simonin, C. Wolf, Task-conditioned adaptation of visual features in multi-task policy learning*, *CVPR 2024* (Marza et al., 2024b)
>
> **Project page**: https://pierremarza.github.io/projects/task_conditioned_adaptation/
>
> **Code**: https://github.com/PierreMarza/task_conditioned_adaptation

- Chapter 8 takes a step back and draws conclusions from the conducted studies in this manuscript. More than this, we will also mention interesting perspectives and current challenges that still need to be solved in Embodied AI, with a particular focus on *simulation*, *fluid intelligence*, i.e. the ability to robustly adapt to new tasks from few examples, and *real-world experiments*.

Chapter

Preliminary concepts

If I have seen further, it is by standing on the shoulders of giants. – Newton, 1675.

Science is indeed a collaborative process, where each new work builds on top of previous ones to increase a global knowledge pool. In this chapter, we will thus introduce general concepts and methods in Deep Learning, Computer Vision, and Sequential Decision-Making, before reviewing the recent progress made in the field of Embodied AI in the next chapter.

More than listing scientific notions, this chapter will try to motivate the use of the considered techniques in our work and highlight how our contributions are inspired and/or different from previous work. Context from work presented later in the manuscript will be presented in boxes like the one below,

> **Context**
> We will mention here how the reviewed concepts are considered in this manuscript and how our contributions relate to previous work.

2.1 Deep Learning

Deep Learning (DL) is a subfield of Machine Learning (ML), itself included in the broader field of Artificial Intelligence (AI). The main goal of Deep Learning is to train artificial neural networks to extract patterns from high-dimensional data to obtain compact and informative representations of complex signals.

2.1.1 *Supervised learning*

Artificial neural networks must be trained, and the choice of the learning signal to do so is often at least as important as the choice of neural architecture. We will here present the simplest learning approach, i.e. supervised learning, before introducing other methods later in this chapter.

Trade-off between guidance and flexibility – When training neural networks, there is a subtle equilibrium to find: providing enough information for the model to learn required concepts within reasonable time and compute constraints, without overly constraining the optimization process with simplified or approximate assumptions. In supervised learning, training samples are labeled to guide the learning of specific skills. Self-supervised learning does not require specific annotations but deals with training a neural model to extract the underlying structure and patterns from data without explicit labels. Finally, in reinforcement learning, a sparse learning signal allows an autonomous agent to explore and implicitly discover important concepts autonomously to solve a task of interest. Self-supervised and reinforcement learning will be presented in later sections when more relevant.

Supervised training signal – The training dataset $S = \{\mathbf{z}_i\}_{i=1}^n = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^n$ is composed of inputs $\{\mathbf{x}_i\}_{i=1}^n$ and labels $\{\mathbf{y}_i\}_{i=1}^n$ that were explicitly crafted for the task at hand. Prior knowledge about the task to solve is thus used to provide informative training labels. Levels of annotation details can vary depending on the task: in image classification, each training image is associated with a single $1-in-K$ vector denoting the corresponding high-level class among the K ground-truth categories, while in dense prediction tasks such as image segmentation, pixel-level supervision is provided with a semantic class associated with each image pixel.

Training is optimization – To train a neural network, we optimize a parametrized mapping from an input space $\mathcal{X}$, where our initial signal lies, to an output representation space $\mathcal{Y}$. More specifically, we can represent a neural network as a function f_θ predicting an output $\hat{\mathbf{y}} \in \mathcal{Y}$ from an input $\mathbf{x} \in \mathcal{X}$, conditioned on parameters θ, also called neural *weights*,

$$\hat{\mathbf{y}} = f_\theta(\mathbf{x}; \theta). \tag{2-1}$$

The objective is to minimize a loss function measuring the error between the predictions of the neural network $\{\hat{\mathbf{y}}_i\}_{i=1}^n$ and ground-truth labels $\{\mathbf{y}_i\}_{i=1}^n$, where $\hat{\mathbf{y}}_\mathbf{i} = f_\theta(\mathbf{x}_\mathbf{i}; \theta)$ following eq. 2-1. The optimization is done by Stochastic Gradient Descent (SGD), where we iteratively sample a batch of data $s \subset S$ to update θ (Bottou, 2010). If we denote the loss function evaluating the error of f_θ on samples from s as $\mathcal{L}(\theta, s)$, and the gradient of this loss function w.r.t parameters θ as $\nabla_\theta \mathcal{L}(\theta, s)$, one update of neural network weights can be performed as,

$$\theta := \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta, s), \tag{2-2}$$

where η is the learning rate. This step is repeated until a stopping criterion is satisfied. SGD is the simplest optimization algorithm, but more sophisticated optimizers exist (Ruder, 2016). A well-known example is the Adam optimizer (Kingma and Ba, 2014) which, instead of having a single learning rate η shared among all neural weights, adapts specific learning rates for all optimized parameters from first and second orders of the gradients.

2.1.2 *Neural architectures*

Artificial Neural Networks (ANN) are defined as a sequential computation graph mixing parametrized learnable blocks, i.e. neural blocks, and unparametrized non-learnable operations. Such a computation graph is known as an *architecture*. We will briefly review the core neural architectures along with their associated assumptions about the underlying function to approximate. The common pattern of all architectures is a sequence of neural layers, each receiving as input the output from the previous layer, and outputting a tensor fed to the next layer as input.

Inductive biases – Deep Learning can be framed as a search in the space of parametrized functions. However, such a space can be large, resulting in a hard optimization problem. To make the latter simpler, an important and useful concept is called *inductive biases* (Mitchell, 1980). These can be defined as the set of assumptions on the function to search for, or the constraints to apply to our search space. Inductive biases can also be seen as the set of prior knowledge injected within the neural architecture. We will now consider different types of neural architectures, highlighting their associated inductive biases.

Multi-Layer Perceptron – Adapted from the original *Perceptron* (Rosenblatt, 1957), the Multi-Layer Perceptron (MLP) is the simplest and most flexible neural network as it is not equipped with any particular inductive bias. It is defined as a sequence of fully-connected layers, composed of a set of neurons, each computing a weighted sum of outputs from all neurons in the previous layer, where weights are the learned parameters. The output of a neuron is passed through a non-linear activation function to be able to approximate non-linear mappings. If we denote the learnable weight matrix and bias for a given layer as W and $\mathbf{b}$, the activation function as σ and its input vector as $\mathbf{x}$, the layer output vector $\mathbf{h}$ is computed as $\mathbf{h} = \sigma(W\mathbf{x} + \mathbf{b})$. An MLP is then built as a sequence of such feed-forward layers, where the output of one layer is the input of the next one. *Universal approximation theorems* (Hornik et al., 1989; Csáji et al., 2001) state that for any function f and closeness criteria $\epsilon > 0$, there exists an MLP network with *enough neurons* to approximate f within ϵ. However, it is important to keep in mind that these are limit theorems, and there are thus no guarantees about the existence of a finite size, i.e. number of neurons, for an MLP to map certain functions.

> **Context**
> MLPs are present in many neural architectures considered in our work, acting as a classifier, a data fusion block, or as a probing tool.

Convolutional Neural Network – The Convolutional Neural Network (CNN) has initially been introduced to encode images (LeCun et al., 1998) but has then been used to extract representations from various modalities. If we focus on image processing, CNNs were designed with two important inductive biases in mind: (i) locality, i.e. image pixels spatially close share information, (ii) translation equivariance, i.e. when associating features with different parts of the image, such features are shifted spatially if the parts of the image are themselves shifted. The core operation in CNNs is the learned convolution, where a convolution kernel is passed on the input image and performs a weighted sum of the neighboring pixels. As for the MLP, weights used when performing the sum are learned. Other operations, such as pooling, can also be used to reduce the resolution of the image representation, increasing the receptive field of neurons.

> **Context**
> CNNs are parameter-efficient image encoders and are thus widely used in our work, often to extract compact information from visual agent observations.

Recurrent Neural Network – Mentioned by Rumelhart et al. (1986), but finding its roots in other domains such as neuroscience (McCulloch and Pitts, 1943), the Recurrent Neural Network (RNN) is a simple architecture, very close to the MLP. The only difference is that it is used to extract information from sequential data and thus, maintains a vectorial hidden memory, fed as an additional input. If we denote the input vector and hidden memory at time t as $\mathbf{x}_t$ and $\mathbf{h}_t$, we have,

$$\mathbf{h}_{t+1} = \sigma(W\mathbf{h}_t + U\mathbf{x}_t + \mathbf{b}), \tag{2-3}$$

where W and U are weight matrices, $\mathbf{b}$ a bias term, and σ an activation function (fully-connected layer). An underlying inductive bias is related to the *Markov property*: the next hidden state of the network, and thus its output, only depends on the previous hidden state without taking the full history of past inputs into account. Simple RNNs will often struggle to memorize information over a long horizon in practice. Improved architectures such as the Long-Short Term Memory (LSTM – Hochreiter and Schmidhuber (1997)) or the Gated Recurrent Unit (GRU – Cho et al. (2014)) were thus introduced to better deal with long sequences of data. They are based on gating mechanisms allowing to explicitly forget old information and memorize new important experiences. These are other inductive biases enforcing LSTM or GRU to predict useless information to forget and data to remember.

> **Context**
> Leveraging temporal information is very valuable when interacting with the world. Recurrent neural networks of different kinds (GRU, LSTM) are thus a common block of many of the methods that will be discussed later as they allow keeping track of the past, e.g. previously seen landmarks.

Transformer model – The Transformer model was introduced by Vaswani et al. (2017), and has had a strong impact first in the field of NLP replacing common RNN-based methods, before being applied in many other fields such as CV. The original Transformer (Vaswani et al., 2017) is composed of an encoder and a decoder. We will here only present the Transformer encoder (right part of Figure 2.2). The latter can be decomposed into a sequence of blocks, each made of the same operations: batch normalization, multi-head self-attention, and a MLP. The core part of the Transformer block is the multi-head attention that we will now detail.

A Transformer block receives a set of embeddings $\{\mathbf{x}_i\}_{i=1}^{n}$, each representing the information associated with an element or *token*, as input. The number of elements in an embedding vector will be denoted as d_e. The goal is to refine the representation of each token through a *multi-head self-attention* mechanism: a similar self-attention mechanism is performed n_h times in parallel, where n_h is the number of attention heads, and the final representations of a given token for all heads are concatenated and fused with a fully-connected layer. For a given head, the self-attention mechanism in itself is about predicting a query $\mathbf{q}_i \in \mathbb{R}^{d_k}$, a key $\mathbf{k}_i \in \mathbb{R}^{d_k}$ and a value $\mathbf{v}_i \in \mathbb{R}^{d_v}$ from each embedding $\mathbf{x}_i$. This is done with fully-connected layers parametrized by matrices $W_q \in \mathbb{R}^{d_e \times d_k}$, $W_k \in \mathbb{R}^{d_e \times d_k}$, $W_v \in \mathbb{R}^{d_e \times d_v}$. If we introduce $E \in \mathbb{R}^{n \times d_e}$, a matrix where the i-th row contains $\mathbf{x}_i$, we can compute the matrices of queries $Q \in \mathbb{R}^{n \times d_k}$, keys $K \in \mathbb{R}^{n \times d_k}$ and values $V \in \mathbb{R}^{n \times d_v}$ as $Q{=}EW_q$, $K{=}EW_k$, $V{=}EW_v$. A similarity matrix $S \in \mathbb{R}^{n \times n}$ can then be computed by performing a dot product between all queries and all keys,

$$S = \frac{QK^T}{\sqrt{d_k}}. \tag{2-4}$$

A softmax operation is applied to S to obtain an attention matrix A containing a probability distribution over all keys for each query. Finally, if we denote the matrix containing the new representations of all tokens as $\tilde{E}$, we compute it as the sum of values weighted by attention scores,

$$\tilde{E} = AV. \tag{2-5}$$

The default Transformer model is powerful at extracting rich representations for input elements belonging to an unordered set. The underlying inductive bias is permutation invariance as the set of output representations will not change if input tokens are permuted within

the input set. Another assumption behind the design of the Transformer encoder is the importance of explicitly performing a token-to-token attention operation based on the specifically chosen *query*, *key*, and *value* quantities.

However, when dealing with ordered data, such as sequences of language tokens in NLP, encoding the position of elements in their initial embedding fed to the Transformer becomes very important (e.g. the order of words in a sentence brings a lot of information to understand it). Thus, Vaswani et al. (2017) introduced *positional encodings*, which are specific embeddings to encode the position of input elements. These can take the form of sine and cosine functions of different frequencies, known as *Fourier features*, but can also be learned from data. Transformers have also been applied to images, and the associated adaptations will be detailed in the next section.

> **Context**
> The Transformer model will be used in chapter 5 to extract a latent representation from a set of neural weights (more details will be presented later), but also as a backbone vision model in chapter 7.

RNN vs Transformer – If we focus on two approaches that have been extensively used to process temporal sequences, i.e. the RNN and Transformer, we can witness a few differences. On the one hand, the RNN (and variants) only maintains a single hidden vectorial memory that is updated at each timestep based on new information. The update is fast but can suffer from long-context forgetting. On the other hand, the self-attention operation in Transformers processes the full input sequence in one pass, suffering from a quadratic complexity with respect to the number of tokens: Vaswani et al. (2017) report a per-layer $\mathcal{O}(n^2 d)$ complexity for the self-attention operation alone, where n is the number of input tokens and d their dimension. However, another important operation is the projection of input tokens into queries, keys and values which has a $\mathcal{O}(nd^2)$ complexity, leading to a $\mathcal{O}(n^2 d + nd^2)$ for a single-head self-attention Transformer forward pass, compared with $\mathcal{O}(nd^2)$ for a recurrent layer. There is thus a trade-off between the efficiency of the RNN and the compactness of its memory representation, and the better ability of the self-attention to model long-range dependencies. Finally, in a decoding context, an advantage of a Transformer decoder is that a full sequence of output tokens can be predicted in a single forward pass (in parallel) at training time, while it is not the case for a RNN.

2.2 Computer Vision

Computer Vision (CV) aims at extracting relevant information from visual signal (Torralba et al., 2024). Recently, many CV applications have been involving Deep Learning methods, and this section will thus be biased towards the latter (while keeping in mind that CV can not only be summarized as recent contributions involving Deep Learning).

Context

An important point to note is that many original CV applications can be considered as *disembodied*: if we see the CV algorithm as an agent, the latter will receive observations of the environment from a visual sensor and will be tasked to extract information from them without having any impact on the environment. While such tasks remain very informative, this is an important difference with problems considered later in this manuscript that mainly involve visual inputs but where agents can take actions in 3D worlds and interact with the latter. We will start by introducing disembodied standard vision tasks and will focus on embodied tasks in the next chapters.

2.2.1 *Encoding 2D images*

Feature extraction from 2D images is an important application. While earlier works in CV leveraged SIFT (Lowe, 2004) or HoG (Dalal and Triggs, 2005) features (still powerful and used in certain applications), the current trend consists in extracting high-level features with neural networks such as a CNN or Transformer model presented previously.

Standard vision tasks – Such features can then be used to solve various tasks of interest. Well-known CV tasks include:

- Image classification: a class among a dictionary of semantic concepts must be associated with an input image.
- Semantic segmentation: a class must be associated with every pixel in the image depending on the type of object it belongs to, without differentiating instances of a given concept.
- Instance segmentation: in addition to predicting the class of a pixel, the instance of this object class should also be provided.
- Panoptic segmentation: semantic segmentation for non-countable concepts (e.g. grass) and instance segmentation for other object types.
- Object detection: Objects belonging to a dictionary of concepts should be detected in the image. The output is a set of bounding boxes defined by their coordinates and the related semantic class.

Context

Some of our work will involve segmentation as a sub-task to solve when detecting objects of interest is necessary (object segmentation will be explicitly tackled in chapters 5 and 6 but will appear as an implicit requirement in other chapters). Some probing experiments will also involve image classification.

The above tasks are solely based on vision but many problems involving mixing this modality with others such as text or audio have been studied with more and more attention. Examples are Visual Question Answering (VQA – Antol et al. (2015); Ben-Younes et al. (2017); Cadene et al. (2019)), image captioning (Chen et al., 2015), image generation from text (Schuhmann et al., 2022), or eventually from language and vision (Zhang et al., 2023).

> **Context**
> Vision-Language tasks have also been introduced in the Embodied AI space. Even if we do not directly tackle such tasks in our work, they will be mentioned in chapter 3 presenting progress in Embodied AI.

CNN models as feature extractors – More generally, convolutional neural networks have been used extensively in CV, following the unprecedented performance of the CNN-based AlexNet method (Krizhevsky et al., 2017) on the now famous Imagenet classification dataset (Deng et al., 2009) in 2012. A following trend has then be to increase the depth of CNN models to increase their expressivity power. Examples are the VGG network (Simonyan and Zisserman, 2014), or the ResNet model (He et al., 2016) that introduced residual connections to make the training of deeper CNN models easier. ResNets are still in use today as they are lightweight and efficient models providing a strong performance on many challenging tasks. Some works such as MobileNet (Howard et al., 2017) have also studied how to make CNN models more efficient to run on a low-resource platform while maintaining performance. Task-specific architectures have then been proposed depending on the requirements of the problems to solve.

Transformer models as feature extractors – A more recent trend is about using Transformer-based networks to extract features from images. Khan et al. (2022) survey such a trend and divide approaches into two categories: (i) CNN models augmented with local or global single-head self-attention and (ii) Transformer-based models in vision that do not use convolutional layers but multi-head self-attention blocks instead.

Indeed, the first works (i) integrated single-head attention inside CNN architectures. A potential issue with convolutional layers is that they process information in a local neighborhood, and thus make it hard to model long-range pixel dependencies. Wang et al. (2018) thus proposed a non-local operator computing a similarity (attention) between all positions in a CNN feature map (this operator can also be applied in recurrent networks along the time axis) to share information between far spatial locations. Figure 2.1, reproduced from Wang et al. (2018), shows a specific instance of the non-local operator applied to a space-time input ($T \times H \times W \times 1024$, where T, H and W respectively denote time, spatial height and width). We see that tensors θ, ϕ and g are projected from the input. θ and ϕ are then compared with a dot-product operation followed by a softmax, as done with *queries* and *keys* in attention mechanisms. Finally, a sum of values in g weighted by obtained similarity scores is produced. g

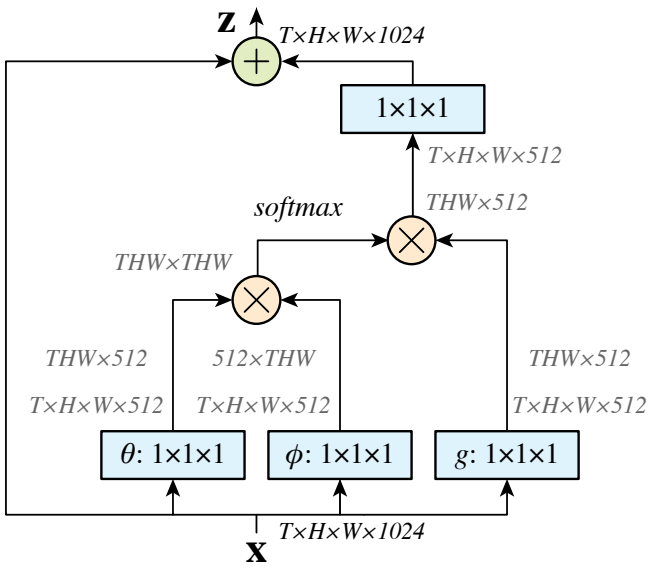


Figure 2.1: **Non-local attention block** – reproduced from Wang et al. (2018)

thus acts here as the *values* tensor in the previously presented attention mechanism. However, by computing a dense attention map for each cell in the feature map, the non-local operator is computationally and memory intensive. Huang et al. (2019) introduced criss-cross attention, allowing to only compute a sparse attention map for each pixel. Another potential shortcoming of the CNN is that, after training, it will process the input with the same set of fixed weights no matter the values in the input. Hu et al. (2019) propose a local attention mechanism that computes local aggregation weights based on the similarities between features in a local window. Finally, some works completely replace convolutional layers with self-attention layers, while keeping similarities with standard CNN architectures. For example, Ramachandran et al. (2019) replace convolutions with local self-attention layers. Despite no convolution being applied, the local self-attention layers still resemble convolutional kernels as they are only applied in a local neighborhood.

Following this direction, many multi-head attention architectures (ii) have been proposed in CV. We will present here the Vision Transformer (ViT) from Dosovitskiy et al. (2020), shown in Figure 2.2. A main challenge was to transform an image into a set of tokens: this is done by dividing an input image into patches, each transformed into a vector embedding through a linear projection. After adding positional encoding to patch vectors, the final embeddings are fed to a standard Transformer encoder, outputting a final representation of each token, i.e. image patch.

> **Context**
>
> The ViT model will be used as a visual backbone in chapter 7 to extract visual features from observations in different visuomotor control tasks.

The ViT model was pre-trained on a large proprietary dataset before being fine-tuned

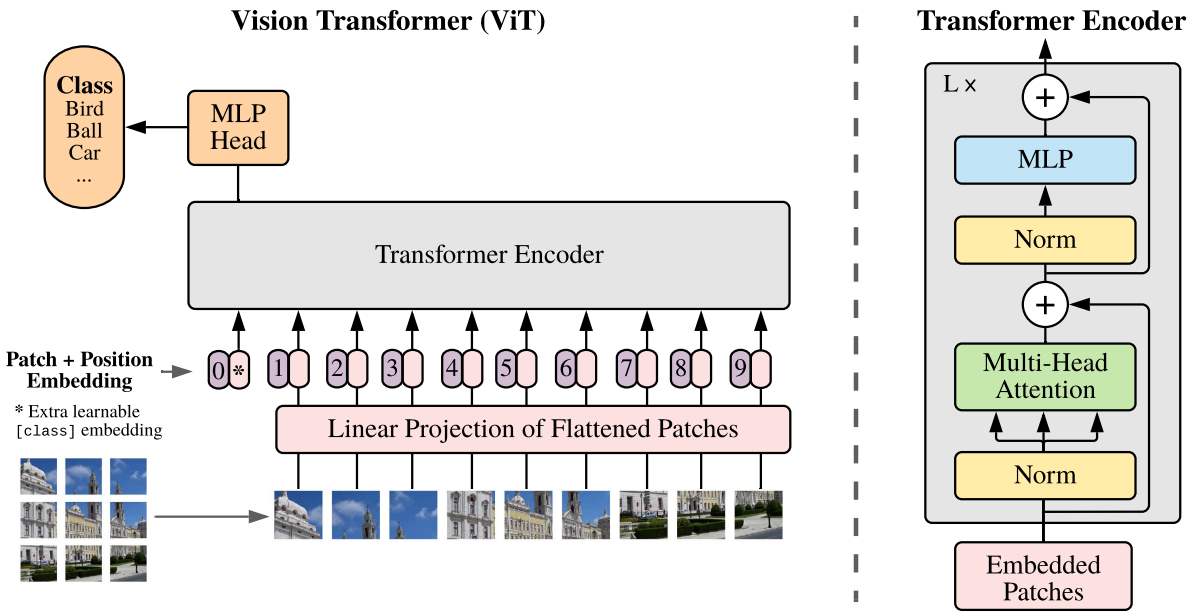


Figure 2.2: **Vision Transformer** – reproduced from Dosovitskiy et al. (2020)

on smaller downstream datasets, leading to the first strong performance of a fully attention-based vision model on medium-size datasets. Indeed, Transformer-based approaches, unlike CNNs, are only equipped with a weak inductive bias (permutation equivariance over patches), and thus need to learn additional image priors from large-scale datasets. DeiT (Touvron et al., 2021) then showed that Transformer-based vision models could be trained on medium-sized datasets by using augmentation and regularization techniques, but more importantly, performing a teacher-student distillation where the teacher is a CNN model. Unlike ViT which was pre-trained on a dataset of 300 million images, DeiT only requires 12 million ImageNet (Deng et al., 2009) images.

Transformer models are also extensively used in multi-modal applications involving vision and other modalities such as language. The best example is the very active space of image generation from text. A few examples are DALL-E (Ramesh et al., 2021), Stable Diffusion (Rombach et al., 2022; Podell et al., 2024) or Imagen (Saharia et al., 2022).

CNN vs Transformer as feature extractors for images – A current belief is that Transformer architectures might scale better than CNN models to large amounts of training data. Even if CNNs might thus still be used in lower resource applications, a recent study (Smith et al., 2023) shows that CNN models trained on web-scale datasets match the final performance of a ViT model. This shows that with a proper design, CNN models might still be a great choice for any application, even when it requires learning patterns from large-scale data.

2.2.2 *General pre-trained vision models*

A promising direction is in pre-training general vision models whose features – sometimes still requiring weight fine-tuning or adaptation techniques – can be used in various downstream tasks. To this end, self-supervised learning is an appealing alternative compared with supervised learning presented earlier, as it allows training vision models (also true in other domains such as NLP) on large unlabeled datasets.

Self-supervised learning – does not require any explicit ground-truth labels. Instead, a neural network is trained to extract patterns from unlabeled data. It allows training models on large diverse datasets to learn general feature extraction abilities, without requiring any labeling process. Unlike what is done in supervised learning, we only consider access to a training dataset $S = \{\mathbf{x}_i\}_{i=1}^{n}$, where $\mathbf{x}_i$ is an input, e.g. a sentence in NLP or an image in CV.

Auto-Encoding – One of the standard self-supervised learning approaches is auto-encoding (Hinton and Salakhutdinov, 2006). The idea is to learn to encode an input signal as a compact latent code from which the input signal can be reconstructed. Let us consider an input $\mathbf{x} \in \mathbb{R}^{d_x}$, an encoder e_θ and a decoder d_ϕ. e_θ will be trained to predict a latent code $\mathbf{h} \in \mathbb{R}^{d_h}$ from $\mathbf{x}$ so that d_ϕ can reconstruct $\mathbf{x}$ from $\mathbf{h}$ as,

$$\mathbf{h} = e_\theta(\mathbf{x}; \theta) \tag{2-6}$$

$$\hat{\mathbf{x}} = d_\phi(\mathbf{h}; \phi). \tag{2-7}$$

$\hat{\mathbf{x}}$ will be compared to $\mathbf{x}$ with a chosen loss function $\mathcal{L}(.)$ (e.g. Mean Squared Error) to provide a supervision signal to train e_θ and d_ϕ. The training objective is thus as follows,

$$\hat{\theta}^* = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(d_\phi(e_\theta(\mathbf{x}_i; \theta); \phi), \mathbf{x}_i) \tag{2-8}$$

$$= \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(d_\phi(\mathbf{h}_i; \phi), \mathbf{x}_i) \tag{2-9}$$

$$= \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\hat{\mathbf{x}}_i, \mathbf{x}_i) \tag{2-10}$$

However, without an important constraint, this setup will not allow learning to extract meaningful representations. Indeed, for auto-encoding to be successful as a self-supervised learning method, the latent representation should be a compact version of the input ($d_h << d_x$). This is the necessary constraint to force e_θ to learn data patterns. Compression with limited loss of information is thus here a great self-supervised objective also called *pre-text task*.

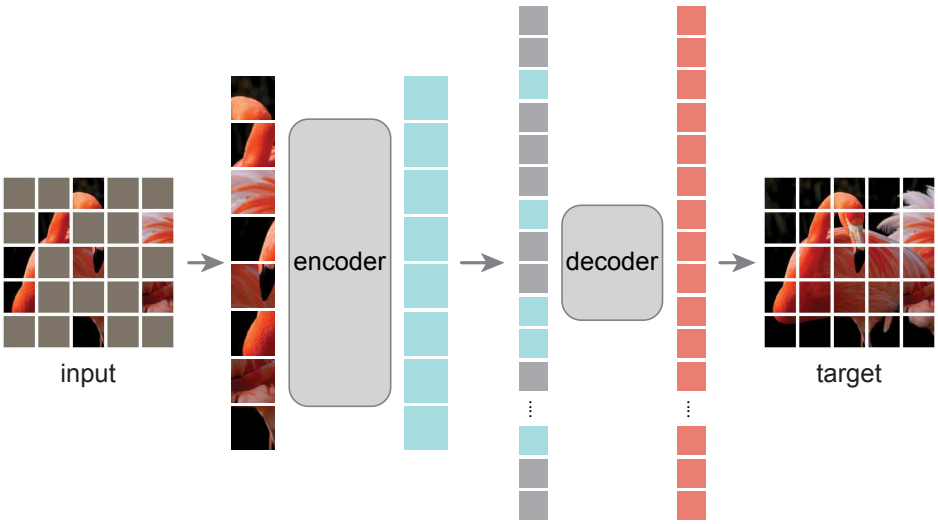


Figure 2.3: **Masked Auto-Encoding** – reproduced from He et al. (2022)

A variant of auto-encoders is known as *denoising auto-encoders* (Vincent et al., 2008) where the encoder-decoder network is fed with a corrupted version of the signal and asked to reconstruct the original (uncorrupted) signal. Such an approach allows the encoder to become more robust to noisy perturbations. A balance must then be found between perturbing the input enough to make the task challenging, but still preserving necessary information.

Masked Auto-Encoding – Recent approaches such as BERT (Devlin et al., 2019) in NLP introduce a type of corruption to the input signal called *masking*, consisting in removing some parts of the input and reconstructing them from latent representations of unmasked parts. In the same spirit as BERT, but this time in CV, another approach known as Masked Auto-Encoding (MAE – He et al. (2022)) was presented. MAE was applied to images, but the concept is easily generalizable to other modalities. As shown in Figure 2.3, MAE follows an encoder-decoder scheme, where the encoder is trained to extract a meaningful latent representation of the input. The decoder will only be used at training time and discarded later. More specifically, the encoder and decoder are Transformer-based models and the encoder, a ViT, is trained to encode an image, i.e. predicting latent embeddings for different patches at different locations. The idea is to mask some input tokens and only encode non-masked input patches (Figure 2.3). Then, the masked tokens should be decoded based on information from non-masked token representations. To properly reconstruct masked regions, the encoder should learn to produce informative and semantically rich embeddings.

Contrastive learning – Other self-supervised learning methods do not rely on reconstructing signals based on a compact latent representation. Examples are contrastive learning methods (Chen et al., 2020; He et al., 2020). The main objective of contrastive learning is to train an encoder e_θ to extract general concepts from high-dimensional inputs. There are many variations of contrastive objectives but, as shown in Figure 2.4, given an input signal $\mathbf{x}$, the original

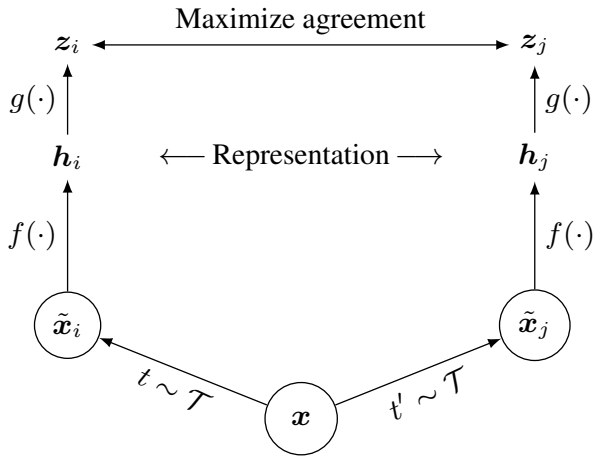


Figure 2.4: **SimCLR training objective** – reproduced from Chen et al. (2020)

idea is to produce two augmented versions of it, $\tilde{\mathbf{x}}_i$ and $\tilde{\mathbf{x}}_j$, and to train e_θ to extract similar latent representations $\mathbf{h}_i$ and $\mathbf{h}_j$ from both inputs. Examples of augmentations used on images in Chen et al. (2020) are *cropping*, *resizing*, *color distortion*, *rotation*, *cutout*, *gaussian noise*, *gaussian blur*, or *sobel filtering*. More specifically, in Chen et al. (2020), the training loss is not exactly computed between $\mathbf{h}_i$ and $\mathbf{h}_j$, but rather between embeddings $\mathbf{z}_i$ and $\mathbf{z}_j$ predicted by a projection head g_ψ ($\mathbf{z}_i = g_\psi(\mathbf{h}_i; \psi)$, $\mathbf{z}_j = g_\psi(\mathbf{h}_j; \psi)$) as this was shown beneficial. In contrastive learning, $(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ is called a *positive pair* as it is composed of two variants of the same input that should thus lead to close representation vectors in the latent embedding space. However, the presented setup is not enough as it could lead to a mode collapse where e_θ would output the same representation no matter the input. An important part of contrastive learning is thus known as *hard negative mining*, i.e. finding negative samples that are different from $\mathbf{x}$ to train e_θ. If the same input $\mathbf{u}$ is considered different from $\mathbf{x}$ – this notion of difference should be defined based on the task to solve, but an example could be a difference in the semantic class the main element of an image belongs to – then $(\tilde{\mathbf{x}}_i, \mathbf{u})$ and $(\tilde{\mathbf{x}}_j, \mathbf{u})$ are called *negative pairs* and e_θ is thus trained to extract different embedding vectors for both inputs in such pairs. Contrastive learning is thus about bringing closer representations for inputs from positive pairs and pushing away latent vectors for inputs from negative pairs. The success of these methods is thus highly dependent on two points: (i) the types of augmentations used in order to define what e_θ should be invariant to, (ii) the mining of negative pairs that should be challenging enough, i.e. $\mathbf{u}$ should not be too different (depending on how such a notion of difference is defined) from $\mathbf{x}$.

Non-contrastive learning – Some approaches (Grill et al., 2020; Zbontar et al., 2021; Bardes et al., 2022) try to get rid of the necessary negative mining from contrastive learning. BYOL (Grill et al., 2020) proposes to not have a single encoder e_θ that will be trained but rather an online network e_θ and a teacher network $e_{\hat{\theta}}$, where $\hat{\theta}$ is updated as a moving-average of θ and the online network is trained to match embeddings predicted by the teacher network leading to a strong performance without negative mining. Methods such as Barlow Twins (Zbontar

et al., 2021) or VICReg (Bardes et al., 2022) are known as non-contrastive as they only focus on bringing closer representations of positive pairs while naturally avoiding any mode collapse by reducing the redundancy between components of the latent embeddings. Finally, a pre-trained vision model called DINO (Caron et al., 2021; Oquab et al., 2023) has been extensively used and studied recently. Inspired by the beneficial use of self-supervised learning on Transformer-based models in NLP, the original DINO paper (Caron et al., 2021) was one of the first to study the impact of self-supervised learning on ViT networks. It takes inspiration from the direction followed in BYOL, as it employs the same student-teacher approach. However, they propose another loss function inspired by knowledge distillation (Hinton et al., 2015) and leverage a multi-crop augmentation strategy (Caron et al., 2020). Importantly, they show the emergence of attention towards whole objects, with attention masks following object boundaries similarly to what would be expected from a segmentation model, without enforcing this at training time. They also show that extracted features lead to strong performance when processed by a simple nearest neighbors classifier, leading to high Imagenet classification performance without training any task-specific linear classifier, or finetuning the feature extractor, as often done. The more recent DINOv2 (Oquab et al., 2023) mostly focuses on improving the speed and stability of known self-supervised learning methods when trained on larger scale datasets than what has been done in the vision community before. Another important part of their work is about efficiently building a large curated dataset of images.

> **Context**
> We will later present work using auto-encoding as a pre-training method in chapter 5, and modules to adapt a large vision model pre-trained with MAE in chapter 7.

Approaches such as SimCLR (Chen et al., 2020), MoCo (He et al., 2020), BYOL (Grill et al., 2020) or MAE (He et al., 2022) have led to strong pre-trained vision models that have been used in downstream applications such as image classification.

> **Context**
> General models are strong tools to solve diverse tasks requiring high-level and rich representations. Chapter 7 of this manuscript will be dedicated to the adaptation of pre-trained vision models in robotics to solve multiple tasks with a single policy.

Multi-modal pre-trained models – Many backbone models used in CV are also multi-modal, often vision-language models, i.e. trained to project both text and images in a common embedding space. An example of a large-scale vision-language model is CLIP (Radford et al. (2021) – see Figure 2.5) that leveraged a contrastive learning objective to simultaneously train a text and vision encoders to respectively project language and vision inputs into a common embedding space, leading to close embeddings for matching inputs (e.g. an image and its associated caption). Other backbone vision-language models are trained with a generative objective such as the already mentioned Stable Diffusion (Rombach et al., 2022; Podell et al.,

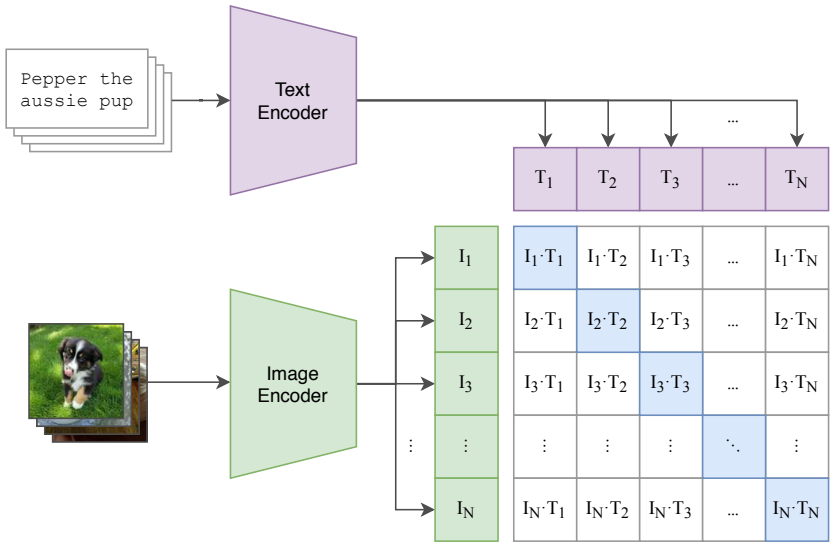


Figure 2.5: **CLIP pre-training** – reproduced from Radford et al. (2021)

2024) or Imagen (Saharia et al., 2022). Indeed, being able to generate realistic images requires learning about the semantics and geometry of images, and thus internal representations of such generative models can provide strong representations of images (or text as they handle both modalities). For instance, Li et al. (2023) show that a generative diffusion model can be used as a zero-shot, i.e. without any additional training, image classifier.

2.2.3 *Representing 3D space*

3D scene representations are another important topic in CV. Indeed, representing 3D space is both interesting from a scientific point of view as, we, humans, are able to greatly represent our surrounding environment in different ways, but also because it is necessary for any downstream task involving some scene understanding. We will thus present different ways to represent 3D information, summarized in Figure 2.6 along with neural-based approaches to build and/or use such representations.

Depth maps – are a first way of representing the geometry of a part of a 3D scene visible from a given viewpoint. Let us consider a standard RGB camera located at a specific position and orientation in a 3D scene. Depending on the specifications of the camera, it will be able to capture a 2D RGB image $i \in \mathbb{R}^{H \times W \times 3}$, where H and W correspond to the pixel height and width of the image. A corresponding depth image with the same resolution will be $d \in \mathbb{R}^{H \times W \times 1}$. Instead of the three channels of an RGB image, a depth map only has a single channel associating the distance from the camera to the closest surface in the world to each pixel. This is also known as a 2.5D representation since 3D distance is represented as a 2D structure. Depth maps can be obtained with specific sensors (e.g. Microsoft Kinect or Intel Realsense depth camera), or estimated from RGB images. There are two main paradigms when it comes to estimating depth from RGB: (i) *stereo depth estimation* and (ii) *monocular depth*

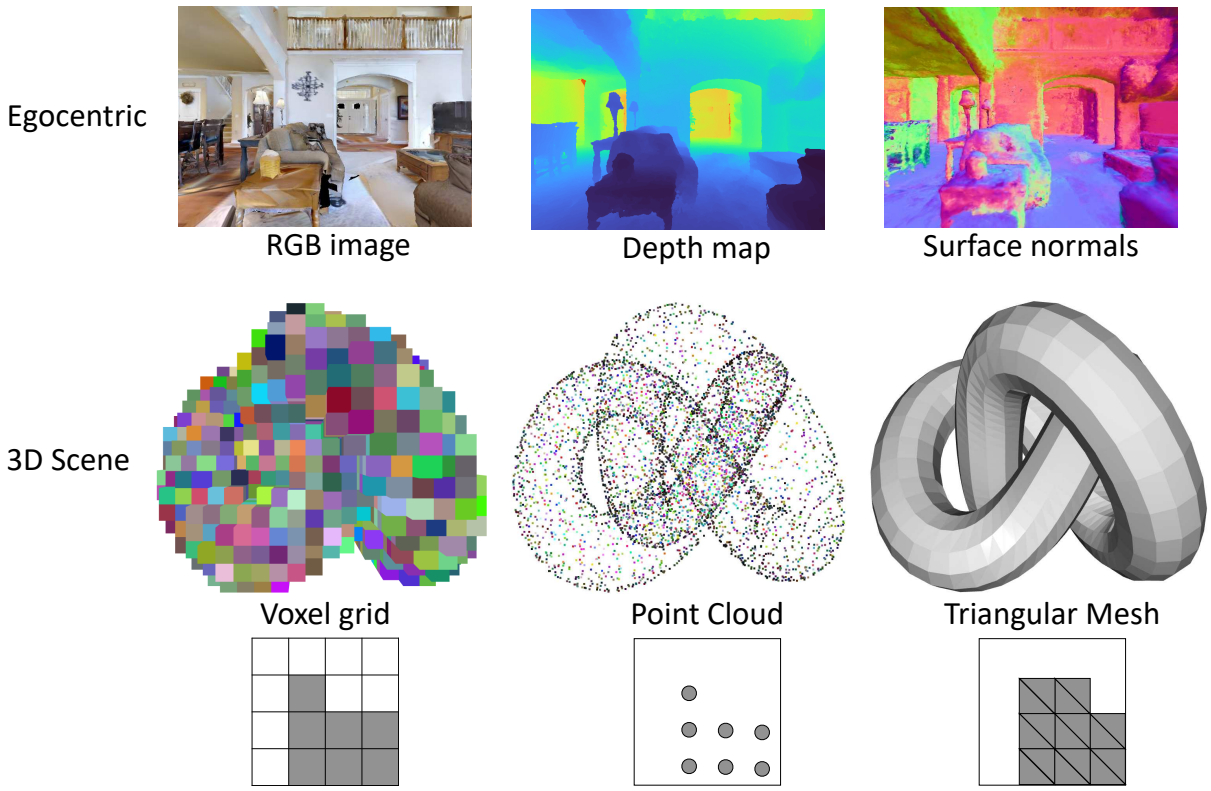


Figure 2.6: **3D scene representations** – the depth map and surface normals are predicted by AutoNeRF (see chapter 6).

estimation. In (i), we use at least two RGB images to retrieve depth information. This is similar to how humans perceive 3D (with our two eyes acting as two sensors, allowing us to perform stereo vision). A popular method is known as *Structure-from-motion* (Schonberger and Frahm, 2016) and can be used to retrieve depth maps from a set of images by extracting features (e.g. SIFT or Deep Learning-based features) from the latter to then find correspondences between them, providing information to estimate the underlying geometry. Structure-from-motion is not limited to generating depth maps, but most of the 3D representations that we will present can actually be obtained from it provided a dense enough set of images. Deep Learning-based methods have also been proposed to perform stereo depth estimation (Li et al., 2021; Smolyanskiy et al., 2018; Tankovich et al., 2021). In (ii) however, we consider only a single RGB image, making the task more challenging. Recent work predicts depth maps from single RGB images using Deep Learning (Eigen and Fergus, 2015; Bhat et al., 2021; Kim et al., 2022; Ranftl et al., 2020, 2021; Yin et al., 2021).

Context

Depth maps are popular in robotics and Embodied AI, as accessible and reliable sensors are available with robots and implemented in simulators. Much of the work that will be presented in this manuscript (chapters 4, 5 and 6) will thus deal with depth maps as compact representations of the surrounding 3D geometry.

Surface normals – When perceiving the world from an egocentric viewpoint, another representation of a visible 3D scene is known as *surface normals*. Again, it takes the form of a 2D map but with three channels as this time each pixel is not associated with the distance to the closest surface, but rather with the coordinates of the vector normal to the latter. As for depth maps, surface normals can be predicted with deep neural networks (Eigen and Fergus, 2015; Wang et al., 2015; Lenssen et al., 2020).

Depth and surface normal maps both represent a part of a 3D scene perceived from a given location. Their 2D structure makes depth and normal maps easily usable as inputs to a vision neural network. However, we might want to construct a representation of a full 3D environment, and not only represent the currently visible scene. We will now review the structures to do this.

Voxel grids – are the simplest representation: it is a 3D tensor with as many channels as required depending on the information to store. In the same way that we divide a 2D image into pixels, here the 3D world is divided into voxel cubes with a certain size. Each voxel will be associated with channels: there is always one to represent occupancy, i.e. 0 if the location is free space and 1 if it intersects an obstacle. Then, we could have additional channels, for instance to store semantic information such as object classes (simple extension of the semantic segmentation problem from 2D to 3D). Voxel grids can be processed with neural models using 3D convolutional operations (Wu et al., 2015). Interestingly, voxel grids can also be generated from a 2D image of the scene of interest. For example, Choy et al. (2016) present a neural method to generate a voxel representation of a 3D scene provided one or more 2D views of the said scene. First, a 2D CNN predicts a latent feature vector for each input view. Then, they introduce *3D-LSTM* units, which are LSTM units spatially arranged as a 3D grid, each responsible for representing a part of the 3D scene. The recurrent structure allows the 3D-LSTM units to update their representation as features from a new view are provided, and importantly, each unit only receives the hidden states of its neighbors. Finally, the high-level 3D representation from the 3D-LSTM is fed to a 3D decoder outputting an occupancy grid.

As already mentioned, an advantage of the voxel representation is its simplicity as it is nothing more than a 3D tensor. However, scaling voxel grids to fine resolutions can be memory intensive depending on the size of the scene to represent. A great alternative is to represent voxel information with an *octree* structure (Tatarchenko et al., 2017). It is a tree where each node represents a part of 3D space and has exactly eight children representing sub-parts of its associated space (that is thus subdivided into eight regions at the next tree level). An octree provides various levels of resolution (the more one goes down the tree, the finer the structure is) and, more importantly, allows to adapt the level of details required for different parts of space (e.g. an empty part would only require the first level of resolution), leading to memory savings.

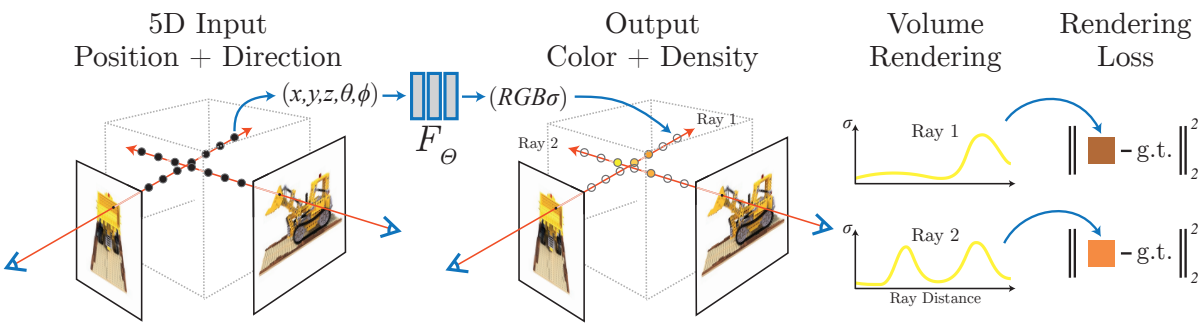


Figure 2.7: **NeRF training** – reproduced from Mildenhall et al. (2020)

Because of its simplicity, the voxel representation is also popular in robotics and Embodied AI, and we will see it can be used to represent free space, obstacles and objects in a scene.

3D point cloud representations – are another popular method, where the geometry of a 3D scene is represented as a set of points all associated with a 3D euclidean coordinate of a non-empty location. Points can be associated with other features than only their coordinates, e.g. a color or a semantic label. Processing a point cloud requires specific neural architectures such as PointNet (Qi et al., 2017a) or PointNet++ (Qi et al., 2017b). PointNet treats the input point cloud as a set, i.e. without any order between points in the representation, by extracting a point-specific representation with a MLP. The different point representations are then pooled to obtain a global representation of the whole point cloud. A point cloud can also be generated from a 2D input image as shown by Fan et al. (2017).

Point clouds are also popular in robotics and Embodied AI, as they are easily computable from available sensors. Indeed, a depth map can be directly converted into a point cloud representing the visible geometry by projecting each 2D pixel depth to its associated 3D coordinate, i.e. a 3D point, from camera intrinsics.

Triangular meshes – are a final 3D representation we will consider. A 3D scene is now represented as a set of vertices V and a set of triangles over these vertices. Compared with voxel grids or point clouds, they are less straightforward to process with neural networks. However, this has been successfully achieved with specialized CNN models (Hanocka et al., 2019) or graph neural networks (Pfaff et al., 2021).

Meshes are a standard representation in computer graphics. Even if they might be less used in robotic applications, they are however often chosen to represent 3D scenes and assets to be loaded in simulators.

More recently, a new scene representation method known as *neural field* based was proposed and will be presented in more detail below.

2.2.4 *Neural fields*

Neural fields were introduced by Mescheder et al. (2019); Park et al. (2019); Chen and Zhang (2019) as an alternative to discrete scene representations such as voxels, point clouds and meshes, with the idea of encoding information of a 3D scene using a continuous neural mapping. A neural network F_Θ, taking the form of a simple MLP, can be queried with a position $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$ to obtain some information of interest i_x about the location $\mathbf{x}$ in the scene as $i_x = F_\Theta(\mathbf{x}; \Theta)$. For example, Mescheder et al. (2019) introduce *Occupancy networks* that map a 3D coordinate to a probability of occupancy ($i_x \in [0, 1]$). Park et al. (2019) propose a neural signed distance field representation, thus $i_x \in \mathbb{R}^+$ represents the predicted distance between a given 3D coordinate ($\mathbf{x}$ as input) and the closest surface in the scene.

NeRF – An important work in this emergent field is the one by Mildenhall et al. (2020) that proposed Neural Radiance Fields (NeRFs). They kept the idea of encoding a 3D scene with the weights of a simple neural network but mainly introduced a new supervision signal based on volume rendering to train it.

Volume rendering is a computer graphics method (Kajiya and Von Herzen, 1984) used to compute a 2D projection of a 3D scene. NeRFs were initially introduced to perform *novel view synthesis* where a 3D scene is described from a set of 2D frames along with camera poses from which the images are obtained. The goal is then to build a representation of the 3D scene to render new 2D views from novel viewpoints (camera poses unseen before). The idea behind the NeRF paper was thus to use volume rendering to supervise the neural representation of the scene by training it to render 2D ground-truth images provided associated camera poses.

In particular, NeRF models were new as they introduced the notion of *radiance* into neural fields, i.e. considering the impact of viewing direction on color when performing RGB rendering. Thus, with NeRFs, the input to the neural network is denoted $(\mathbf{x}, \mathbf{d})$, where $\mathbf{x} = (x, y, z) \in \mathbb{R}^3$ still represents a location in the 3D scene and $\mathbf{d} \in \mathbb{R}^3$ is the 3D unit vector describing the direction of the viewing camera. The NeRF will output the density $\sigma \in \mathbb{R}$ at the 3D location (x, y, z) and the color $\mathbf{c} = (r, g, b) \in \{0, 1\}^3$ of this point seen from the camera orientation $\mathbf{d}$. The forward pass of a NeRF model F_Θ can then be formulated as,

$$(\sigma, \mathbf{c}) = F_\Theta(\mathbf{x}, \mathbf{d}; \Theta) \tag{2-11}$$

More specifically, a NeRF model is an MLP that can be divided into two blocks. The first block is composed of 8 fully-connected layers and only takes the 3D location $\mathbf{x}$ as input to predict the density σ (which is independent of the camera direction) and a 256-dimensional embedding. The latter is concatenated to the camera direction $\mathbf{d}$ and fed to the second block composed of a single fully-connected layer predicting the color $\mathbf{c}$.

In order to understand the training scheme of NeRFs, we can first consider we have a

trained model and wonder how to render a 2D frame provided an arbitrary camera pose. A pixel in the final image to render has a location $\mathbf{l}_p \in \mathbb{R}^3$ in the scene (that can be retrieved from the camera pose and specifications). Following principles in volume rendering, for each pixel in the image, we will emit a ray $\mathbf{i}_p(t) = \mathbf{l}_p + t\mathbf{d}$ starting at position o and following direction $\mathbf{d}$. As formalized by Mildenhall et al. (2020), if we define near and far emission bounds t_n and t_f, the expected color of the ray $C(\mathbf{i}_p)$ is,

$$C(\mathbf{i}_p) = \int_{t_n}^{t_f} T(t)\sigma(\mathbf{i}_p(t))\mathbf{c}(\mathbf{i}_p(t), \mathbf{d})dt, \text{ where } T(t) = \exp\left(-\int_{t_n}^{t} \sigma(\mathbf{i}_p(s))ds\right), \tag{2-12}$$

where $T(t)$, the accumulated transmittance along the ray from t_n to t, denotes the probability for the ray to travel from t_n to t without encountering a dense particle.

In practice, this continuous integral is numerically estimated by sampling K random quadrature points $\{t_k\}_{k=1}^{K}$ between near and far bounds. More specifically, this is done with *stratified sampling*, i.e. dividing the space between bounds into K evenly-spaced bins and sampling one quadrature point uniformly inside each bin, leading to more diverse queries compared with querying the NeRF model at fixed positions along the ray. The approximated expected ray color is then,

$$\hat{C}(\mathbf{i}_p) = \sum_{i=1}^{N} T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \mathbf{c}_i, \text{ where } T_i = \exp\left(-\sum_{j=1}^{i-1} \sigma_j \delta_j\right), \tag{2-13}$$

with $\delta_i = t_{i+1} - t_i$.

Context

NeRF models, and more generally neural-based representations of 3D scenes, will be used in chapters 5 and 6 in the context of navigating agents.

Two important concepts were also presented by Mildenhall et al. (2020): *positional encoding* and *hierarchical volume sampling*.

Positional encoding – Representing high-frequency color and geometry details with a neural network taking a low-dimensional 3D location and camera orientation as inputs is hard. A NeRF model is thus fed with a higher-dimensional transformed version of such input values, obtained by separately applying an encoding function γ to all coordinates of $\mathbf{x}$ and $\mathbf{d}$. The encoding function used by Mildenhall et al. (2020) is described as,

$$\gamma(p) = \left(\sin\left(2^0\pi p\right), \cos\left(2^0\pi p\right), \cdots, \sin\left(2^{L-1}\pi p\right), \cos\left(2^{L-1}\pi p\right)\right) \tag{2-14}$$

This operation is similar to what was done with the Transformer (Vaswani et al., 2017) that was presented earlier and already called *positional encoding*, mapping low-dimensional input coordinates to a high-dimensional space. A follow-up paper (Tancik et al., 2020) studies further the importance of these high-dimensional embeddings, also called *Fourier features*, to learn high-frequency functions.

> **Context**
> Some work presented in this manuscript deals with training neural implicit representations of scenes, and we also witnessed the significant gains brought by positional encoding when it comes to reconstruction quality in chapter 5.

Hierarchical volume sampling – Performing stratified sampling along a ray to estimate the color of the corresponding pixel can be inefficient as useless parts of the scene, such as free space or occluded regions, will be queried extensively while not bringing any important information. Taking inspiration from previous work in volume rendering (Levoy, 1990) advocating for a hierarchical representation of space, two different NeRF models are trained: a *coarse* and a *fine* model. K_c points will be selected by stratified sampling as presented previously and be used to query the *coarse* model as described in equation 2-13. We can re-write the latter as a weighted sum of all predicted colors along the ray,

$$\hat{C}_c(\mathbf{i}_p) = \sum_{i=1}^{K_c} w_i c_i, \quad w_i = T_i \left(1 - \exp\left(-\sigma_i \delta_i\right)\right) \tag{2-15}$$

Then, another sampling of K_f query points can be performed, informed by the predictions of the *coarse* network, favoring locations playing an important role in the final color to render. After normalizing weights $\{w_i\}_{i=1\ldots K_c}$ as $\hat{w}_i = w_i / \sum_{j=1}^{K_c} w_j$, we can get a piece-wise constant probability density function (PDF) along the ray, and sample K_f new points according to this PDF. The *fine* network can finally be queried with all points from the two sampled sets $(K_c + K_f)$ to render the pixel color following equation 2-13.

Improving over the vanilla NeRF model – Follow-up work has addressed some drawbacks of the original NeRF model, mostly training and rendering speed, rendering quality or sensitivity to noisy camera poses, and sparse training views.

An example of work tackling training and rendering speed is Instant-NGP (Müller et al., 2022), illustrated in Figure 2.8. The objective in Instant-NGP is to provide a new encoding of the location input $\mathbf{x}$ allowing to have a faster training while conserving the rendering quality. The scene space is first divided into L sets of voxels of varying resolutions. For each voxel set, the voxel containing $\mathbf{x}$ is selected, and indices of its corner are retrieved. A look-up table is then queried with the corner indices to get an embedding per corner. The embeddings are linearly interpolated based on the relative position of $\mathbf{x}$ in the voxel, leading to a final embedding per voxel set. The L embeddings are then concatenated along with potential

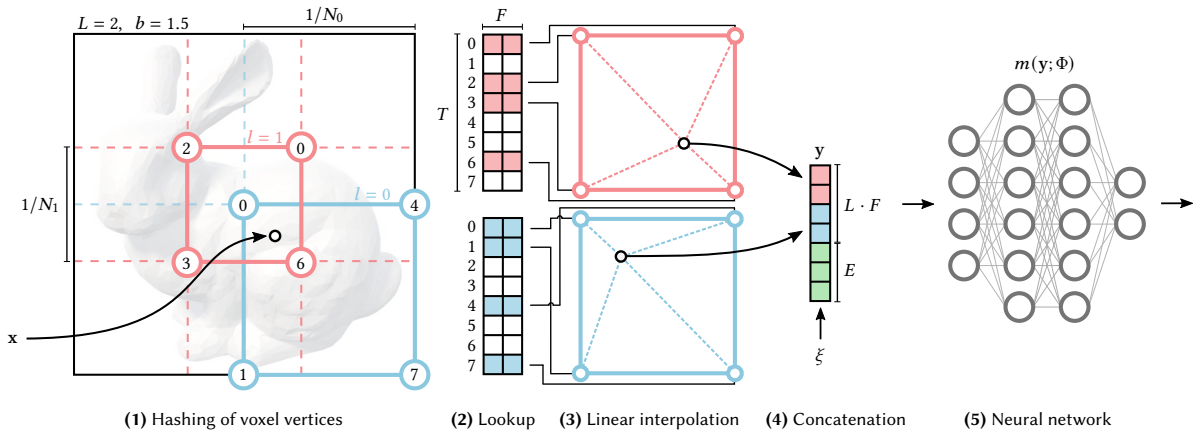


Figure 2.8: **Instant-NGP** – reproduced from Müller et al. (2022)

additional inputs such as the encoded camera direction. The obtained vector is finally fed to the MLP model. Importantly the hash table entries are trained along the MLP weights by backpropagating the loss gradients through the concatenation and linear interpolation.

A concurrent work, Plenoxel (Fridovich-Keil et al., 2022), provides another approach to faster training by modeling the neural field with a sparse voxel grid instead of a neural network. They keep the idea of a differentiable volume rendering but show that a sparse voxel grid is faster to optimize. Fast-NeRF (Garbin et al., 2021) does not try to improve the training but the rendering speed. They show they can reach up to 200FPS by caching NeRF outputs to render new pixels faster, following practices in computer graphics.

Another line of work is about representing scenes from only a few images. PixelNeRF (Yu et al., 2021) proposes to jointly train an image encoder and a NeRF model on a dataset composed of different scenes to learn scene priors. The encoder extracts a feature map from an image of a new scene. The NeRF model is then queried with the standard 3D location and camera direction, but also a feature associated with the location, obtained by projecting the 3D point to the 2D camera plane and selecting the corresponding feature on the map from the encoder. They also extend their approach to the multi-view setting. This allows PixelNeRF to generalize to new scenes, and thus to represent any new environment from a limited set of views.

Finally, a last interesting direction is about making NeRF models more robust to noisy camera poses. BARF (Lin et al., 2021) optimizes input camera poses along with the NeRF weights during training. They propose a coarse-to-fine view registration method consisting in masking all dimensions in the positional encoding vector at the beginning of training and gradually activating dimensions corresponding to higher frequencies. This allows to ignore fine details when performing camera registration at the beginning of training, and refine poses based on higher-frequency details. Follow-up work called SPARF (Truong et al., 2023b) focuses on training NeRF from noisy poses but also sparse training views. Their main contribution

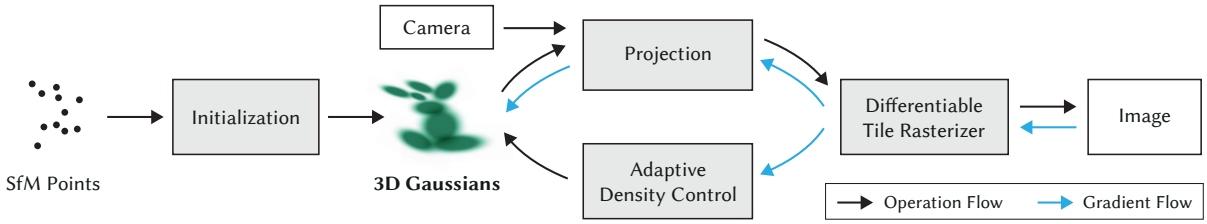


Figure 2.9: **Gaussian Splatting** – reproduced from Kerbl et al. (2023)

is about augmenting the vanilla NeRF supervision with a multi-view consistency objective to better optimize camera poses: provided with a set of 2D images, the idea is to match pixels in different views with similar features, and then optimize camera poses so that when backprojecting a 3D point into the 2D plane of two views, we obtain two matched pixels. This is performed while also training the NeRF model to render training views.

Gaussian Splatting – More recently, a novel approach, known as 3D Gaussian Splatting (Kerbl et al., 2023), competes with NeRF models, by replacing the underlying neural network as a scene encoding method with a mixture of gaussians. As shown in Figure 2.9, the process starts from a set of camera poses along with a sparse scene point cloud (as presented earlier) estimated with Structure-from-motion. The scene is not represented by an MLP but instead a set of 3D gaussians. Each gaussian is defined by a position p (mean), a covariance matrix Σ, an opacity coefficient α, and a view-dependent RGB color represented by spherical harmonic coefficients s_h. 3D gaussians are differentiable and can be projected to 2D to render an image. This work employs a tile-based rasterizer (Lassner and Zollhofer, 2021) to do so. A 2D view can then be rendered at training time to be compared to a provided ground-truth image to compute loss gradients to be backpropagated to the gaussians to adapt their parameters (p, Σ, α, s_h). Another important contribution is the adaptive density control: during training, gaussians are regularly added to densify certain areas, and gaussians with an alpha value lower than a certain threshold are pruned. Gaussian Splatting allows reaching high-quality visual rendering while training rapidly, and the tile-based rasterization leads to real-time rendering.

Extracting information from neural field weights – Neural implicit representations are instances of function spaces, which are represented through their trainable parameters. We can thus question whether one can extract global information from their learned weights.

Related work has already been performed on different types of neural networks (different from NeRF models): previous work performed analyses by predicting the classification accuracy of neural networks from their weights as input (Unterthiner et al., 2020; Martin and Mahoney, 2020; Martin et al., 2021) or estimating the generality gap between train and test performance from hidden neural activations (Jiang et al., 2019; Yak et al., 2019).

Another related direction pioneered by Hypernetworks (Ha et al., 2016) is about predict-

ing neural network weights. Recently, Zhmoginov et al. (2022) generate the weights of a CNN from support samples in the context of few-shot learning. More related to our interests, Pan et al. (2023) learn to predict the weights of an implicit representation based on external factors in the context of spatio-temporal dynamics encoding.

Finally, some recent work has been studying how to extract information from the weights of neural implicit representations. Dupont et al. (2022) represent implicit neural representations (INR) as data points called *functa* and train models to perform downstream tasks ranging from classification to generative modeling (generating new functa). They use SIREN (Sitzmann et al., 2020) as the base architecture to represent different signals as neural functa. Rather than directly using weight values as the representation of an INR, they rely on what is known as *modulations*: representing a neural network as its activation variations compared with a base network (Perez et al., 2018; Chan et al., 2021). More specifically, they use *latent shift modulations*, by mapping shift variations in activations to a latent embedding (Chan et al., 2021). Very related to this subject, Zhou et al. (2024) introduce a method to automatically construct weight-space models that are equivariant to neuron permutations in a network. Such a universal approach leveraging prior knowledge about permutation symmetries in neural models could be used to extract a compact representation of a neural field.

> **Context**
> Chapter 5 will involve a neural-based reader estimating a global embedding from the weights of a neural scene representation.

2.2.5 *3D representations for robotics*

Representing 3D space has not only been a focus in CV but also in robotics. Indeed, keeping track of the surrounding 3D world is essential to navigate an environment and/or manipulate objects.

Scene representations – We will focus here on two important structures used to represent space: *metric* and *topological* maps, that can eventually be combined (Thrun, 1998). Metric maps are reminiscent of voxel grid representations in CV as they often have a grid-like structure, but are more frequently 2D (projected from 3D voxel grids or point clouds), although 3D metric maps also exist. In robotics, specific channels can be introduced to map information of interest, e.g. free space, obstacles, eventually semantic channels to map the location of objects. The particular case of metric maps representing obstacles and free space is known as *occupancy grids* (Borenstein et al., 1991; Schiele and Crowley, 1994; Kortenkamp et al., 1998; Thrun, 1998; Konolige and Chou, 1999; Thrun, 2003). Topological representations (Kortenkamp and Weymouth, 1994; Yamauchi and Beer, 1996; Shatkay and Kaelbling, 1997, 2002) are based on graphs where nodes might represent important landmarks of interest and edges relationships between nodes (e.g. distances). We can associate features with the nodes depending on the

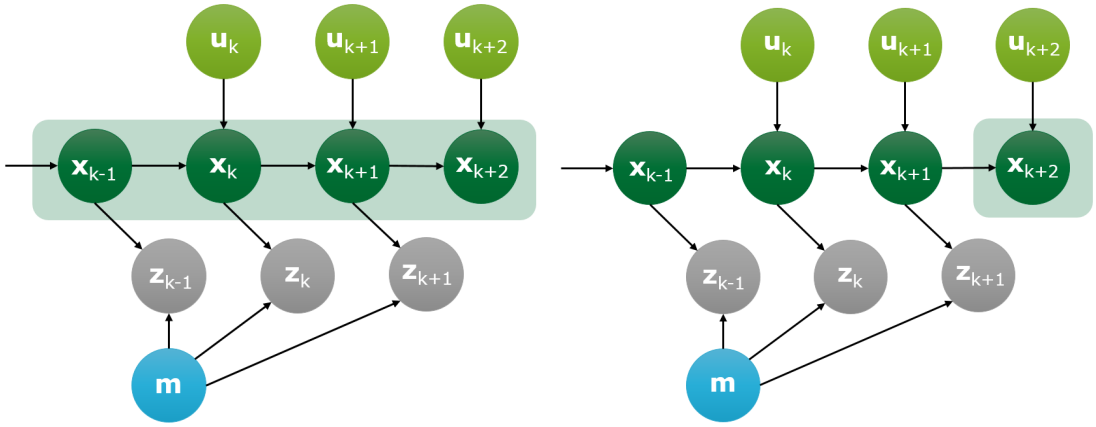


Figure 2.10: **Original and online SLAM problems**: Graphical illustration of the original SLAM problem (left) and the online SLAM problem at time k+2 (right) – reproduced from Bresson et al. (2017)

type of information to solve (e.g. semantic information to describe each landmark).

Simultaneous Localization and Mapping – In robotics, as an agent is navigating inside an environment, the goal can be to both map such a scene but also to localize inside it. To this end, an important research direction is known as *Simultaneous Localization And Mapping* (SLAM). Introduced by Smith and Cheeseman (1986), SLAM is an important topic in robotics. Bresson et al. (2017) review advances in this field. Figure 2.10 reproduced from Bresson et al. (2017) illustrates the original SLAM problem (left) and the variant online SLAM problem (right). Let us denote the pose of an agent at time step k as $\mathbf{x}_k$ and the map of the environment it explores as $\mathbf{m}$. The goal in SLAM is to estimate both variables as precisely as possible to have as much information as possible about the environment (with the estimation of $\mathbf{m}$) and our location inside it (with the estimation of $\mathbf{x}_k$). To this end, we can rely on control inputs $\mathbf{u}_k$ that provide an estimate of the motion between timesteps $k-1$ and k. However, we generally also leverage sensors providing readings denoted here as $\mathbf{z}_k$, i.e. additional information to locate and build a map of the environment. These sensors could be RGB, depth cameras, or LIDAR sensors.

In full SLAM (left part of Figure 2.10), we have access to the full sequence of control inputs and sensor readings and thus estimate the joint posterior over the whole sequence of poses and the map from sensory data. This internal estimation is also called *belief* and noted as,

$$bel\left(\mathbf{x}_{0:k}, \mathbf{m}\right) = p(\mathbf{x}_{0:k}, \mathbf{m} \mid \mathbf{z}_{0:k}, \mathbf{u}_{0:k}) \tag{2-16}$$

In online SLAM (right part of Figure 2.10), we estimate the location of the agent at time k and the map from previous data,

$$bel\left(\mathbf{x}_k, \mathbf{m}\right) = p(\mathbf{x}_k, \mathbf{m} \mid \mathbf{z}_{0:k}, \mathbf{u}_{0:k}) \tag{2-17}$$

There are many SLAM variants, relying on metric or topological maps, but we will mostly

consider here how cameras can be used to improve mapping and localization in SLAM. The subdomain addressing this problem is known as Visual SLAM (Macario Barros et al., 2022), and methods relying only on visual inputs can be roughly divided into two types of approaches: feature-based and direct methods. An important point to note is that the main pipeline in Visual SLAM is very similar to Structure-from-motion in CV as the goal will be to match parts of different views that represent the same locations in 3D space to optimize the associated camera poses. The difference between feature-based and direct methods will be about the nature of the frame representations used to perform matching. Feature-based approaches will extract a sparse set of features from visual frames, and match similar features in different images. A famous feature-based method is ORB-SLAM (Mur-Artal et al., 2015; Mur-Artal and Tardós, 2017; Campos et al., 2021) that extracts ORB features (Rublee et al., 2011) from each new image and matches them with the ones from previous frames. More recently, Deep Learning features have been leveraged as well (Sarlin et al., 2020, 2021; Lindenberger et al., 2023). On the contrary, direct methods perform mapping in pixel space, by comparing RGB or depth frames. Unlike feature-based methods that extract sparse features from images, direct methods such as DTAM (Newcombe et al., 2011) perform a denser reconstruction of the 3D scene.

2.3 Sequential Decision-Making

Now that we have covered concepts related to visual perception, we will dive into another important concept in this manuscript, i.e. sequential decision-making, where an agent takes actions sequentially in an environment to achieve some goals of interest.

2.3.1 *Markov Decision Processes*

Introduced by Bellman (1957), a Markov Decision Process (MDP) provides a framework to formalize sequential decision-making processes. Let us dive into the different core elements characterizing it.

The agent-environment interaction – As illustrated in Figure 2.11, the two central blocks are the agent and the environment. The agent is the decision-maker that takes actions in an environment. The agent and environment interact sequentially with discretized time. At each timestep t, the agent receives a description of the environment known as state $s_t \in \mathcal{S}$, where $\mathcal{S}$ is the set of states describing all possible configurations of the environment. Given s_t, the agent will pick an action $a_t \in \mathcal{A}$, where $\mathcal{A}$ is the set of all possible actions.

State transitions – Taking the action a_t will affect the environment that will transition to state s_{t+1} at the next timestep $t+1$. An important property of a MDP is thus the transition function $\mathcal{P}(s_t, a_t, s_{t+1}) = \mathbb{P}(s_{t+1} \mid s_t, a_t)$ modeling the probability of transitioning from a state s_t at time t to a state s_{t+1} at time $t+1$ given an action a_t. Importantly, the probability to reach state s_{t+1}

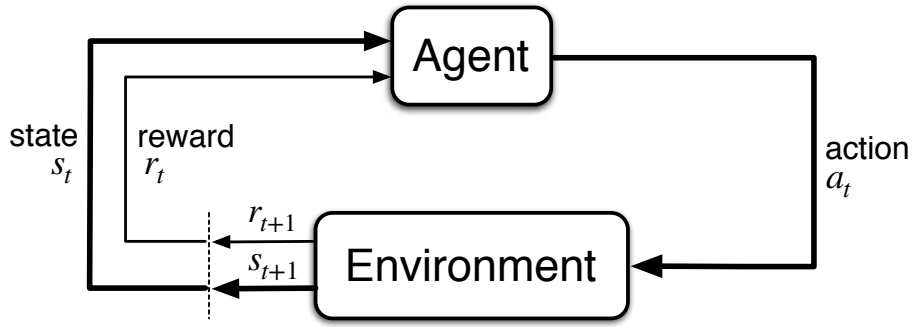


Figure 2.11: **The agent-environment interaction** – reproduced from Sutton and Barto (2018)

does not depend on previous states (s_0, ..., s_{t-1}) as the environment describing a MDP follows the *Markov property*. This could be re-written as $\mathbb{P}(s_{t+1} \mid s_0, ..., s_{t-1}, s_t, a_t) = \mathbb{P}(s_{t+1} \mid s_t, a_t)$.

Reward – A MDP is also defined by a reward function $\mathcal{R}(s_t, a_t, s_{t+1})$, which will output a scalar reward value r_{t+1} given s_t, a_t and s_{t+1}, quantifying how helpful the taken action a_t is in solving the problem formalized by the MDP.

Trajectories – The maximal length of a sequence of interactions is known as the *horizon* H, which could be infinite ($H{=}\infty$) for an infinite-horizon MDP or an integer for a finite-horizon MDP. We refer to the sequence of states and actions as a *trajectory* τ, also called *episode*, where $\tau = (s_0, a_0, s_1, a_1, ..., s_H)$. s_0 is the initial state and is obtained by sampling an initial state distribution μ ($s_0 \sim \mu$).

Policy – The agent implements a mapping from states to actions called a *policy* and denoted as π. In this manuscript, we will particularly consider *stochastic policies* mapping states to probability distributions over all possible actions. The process of choosing action a_t with a stochastic policy π can now be written as,

$$a_t \sim \pi(\cdot \mid s_t). \tag{2-18}$$

Return – In order to quantify the performance of a policy, an important quantity is the *return* $R(\tau)$, i.e. the discounted sum of rewards in a trajectory τ,

$$R(\tau) = \sum_{t=0}^{H} \gamma^t r_t, \tag{2-19}$$

where $\gamma \in [0, 1]$ is the *discount factor*, and allows to decrease the importance of atomic rewards with time. This is particularly important when dealing with infinite-horizon MDPs, but even with finite-horizon MDPs where the goal is to maximize the cumulated reward along an episode, a given reward quantity should often be better obtained early than later in the sequence.

The final goal is to maximize the expected return across episodes $J(\pi)$ defined as,

$$J(\pi) = \mathop{\mathrm{E}}_{\tau\sim\pi}[R(\tau)] \tag{2-20}$$

MDP description – To summarize, a MDP can thus be characterized by,

- A set of states $\mathcal{S}$ the environment can be in.
- A set of actions $\mathcal{A}$ the agent can take.
- A transition function $\mathcal{P}(s_t, a_t, s_{t+1})$ modeling the transition from a state s_t at time t to a state s_{t+1} at time $t+1$ given an action a_t.
- A reward function $\mathcal{R}(s_t, a_t, s_{t+1})$ modeling the quality of a taken action a_t leading to a transition from s_t to s_{t+1}.
- An initial state distribution μ.
- An horizon H that describes the maximum length of an episode.
- A discount factor γ to control the decrease in the impact of atomic rewards with time on the return $R(\tau)$.

Extension to Partially Observable Markov Decision Processes – In an MDP, the environment is considered fully observable as the agent receives its full state s_t at each timestep t. Partially Observable Markov Decision Processes (POMDP – Kaelbling et al. (1998)) assume that the environment dynamics follow a MDP, but that states are not directly observable by the agent. Instead, it receives an observation $o_t \in \Omega$, where Ω defines the set of possible observations, partially describing the state s_t. When describing a POMDP, an observation function $\mathcal{O}(s_t, a_t)$ outputting a probability distribution over observations from the state s_t and action a_t as inputs should be specified. Such an observation process could for example be defined by a set of sensors capturing partial state information.

A POMDP is thus characterized by the same quantities as an MDP, to which the set of observations Ω and observation function $\mathcal{O}(s_t, a_t)$ should be added.

Context

The navigation problems we will consider in this manuscript can be formalized as POMDPs as trained agents do not have access to the state of the environment, but rather partial observations of its state coming from sensors such as RGB-D cameras or odometry sensors. Aggregating observations from previous timesteps will thus become necessary to act properly in the considered environments. More importantly, studying this process of converting partial observations into a well-suited representa-

tion of the world to solve a problem of interest is a main target of this manuscript.

2.3.2 *Reinforcement Learning*

Reinforcement Learning (RL) is about training agents to interact with an environment providing observations and rewards depending on taken actions. More specifically, any RL problem can be formulated as an MDP or, as for the tasks we will consider later in this manuscript, a POMDP. A challenge in RL, particularly when dealing with long-horizon trajectories and sparse reward signals, is known as the *credit assignment problem*. Indeed, policy training will be about determining which decisions, i.e. actions or combinations of actions, have led to a delayed success or failure, with the final goal to obtain a policy picking optimal actions leading to success.

An important question in RL is the following: *How good is it for a policy to reach state s_t in order to maximize the expected episode return?* Such a question can be answered with the help of *value functions*, i.e. the *state-value function* and the *action-value function*.

The *state-value function* $V^\pi(s_t)$ is equal to the expected return when starting in state s_t and following policy π until the end of the episode,

$$V^\pi(s_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{H-t} \gamma^k r_{t+k+1} \mid s_t \right], \text{ for all } s_t \in \mathcal{S}. \tag{2-21}$$

The *action-value function* $Q^\pi(s_t, a_t)$ outputs the expected return when starting in state s_t, taking action a_t and then following policy π,

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[\sum_{k=0}^{H-t} \gamma^k r_{t+k+1} \mid s_t, a_t \right], \text{ for all } s_t \in \mathcal{S} \text{ and } a_t \in \mathcal{A}. \tag{2-22}$$

It can also be useful to know how much an action is better than the others. The *advantage function* $A^\pi(s_t, a_t)$ quantifies this by computing the difference between the action-value function and the state-value function as,

$$A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t). \tag{2-23}$$

In RL, a policy π^1 can be considered as better than another policy π^2 if it leads to a higher expected return. In this sense, we can then define the *optimal policy* π^* as the one reaching the highest possible expected return for a given task ($\pi^* = \arg\max_\pi J(\pi)$). The state-value and action-value functions of the optimal policy are called the optimal state-value function $V^*(s_t)$ and the optimal action-value function $Q^*(s_t, a_t)$, and can be defined as,

$$V^*(s_t) = \max_\pi V^\pi(s_t), \tag{2-24}$$

$$Q^*(s_t, a_t) = \max_\pi Q^\pi(s_t, a_t). \tag{2-25}$$

The optimal action-value function can also be defined recursively through the Bellman Equation,

$$Q^*(s_t, a_t) = \sum_{s_{t+1} \in \mathcal{S}} \mathcal{P}(s_t, a_t, s_{t+1}) \left(\mathcal{R}(s_t, a_t, s_{t+1}) + \gamma \max_{a_{t+1} \in \mathcal{A}} Q^*(s_{t+1}, a_{t+1}) \right). \tag{2-26}$$

Our goal is not to provide an exhaustive taxonomy of all existing RL algorithms, but rather to focus on a few important characteristics allowing to compare many of them: *model-based*/*model-free*, *online*/*offline*, *on-policy*/*off-policy*. First, we should differentiate *model-based* and *model-free* methods. *Model-based* methods are either provided with or try to estimate a model of the environment, more specifically of state transitions ($\mathcal{P}$) and rewards ($\mathcal{R}$), while *model-free* algorithms do not explicitly build a model of the environment. Then, an RL algorithm can either be *online* or *offline*, depending on how the training data is collected: it can either be collected interactively (*online*) or come from a pre-recorded dataset (*offline*). Finally, an important difference is between *on-policy* and *off-policy* methods. In *on-policy* algorithms, the trained policy is also used to collect the training data, while *off-policy* methods are about using a different policy to collect data used to update the policy of interest. Note that in *off-policy* methods, the policy used to gather data can be the same as the trained policy, but earlier in the training process (i.e. with an old version of neural weights).

2.3.3 *Deep Reinforcement Learning*

Neural policies – When dealing with problems that can be formalized as MDP or POMDP, the objective is to find a policy π that maps a state (MDP) or an observation (POMDP) to a distribution over actions. Such a policy can be a neural network that takes a state or observation represented as a tensor as input and outputs a probability distribution over actions. A neural policy π_θ is parametrized by neural weights θ and finding the optimal policy will be implemented as a training, or weights optimization.

> **Context**
> All considered policies in our work are neural-based: neural networks predict the best actions given an observation (POMDP) as input.

In this manuscript, we will focus on a subset of *model-free* algorithms, known as *policy gradient methods*, where a policy is a neural network π_θ and parameters θ are directly optimized

to maximize the expected return $J(\pi_\theta)$ as introduced in equation 2-20 where π is replaced by π_θ.

In order to optimize θ, we need to compute $\nabla_\theta J(\pi_\theta)$. The *Policy Gradient Theorem* (Sutton et al., 1999) lets us write it as,

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{(s_t,a_t)\sim\pi_\theta}\left[Q^{\pi_\theta}(s_t,a_t)\nabla_\theta \ln \pi_\theta(a_t \mid s_t,\theta)\right]. \tag{2-27}$$

A simple policy gradient method is called *REINFORCE* (Williams, 1992), or *Monte-Carlo policy gradient* (algorithm 1). At each optimization step, we generate a full-length episode (as it is a Monte-Carlo method) with π_θ. Then, for each time step t in the episode, we compute the discounted return $G_t = \sum_{k=0}^{H-t} \gamma^k r_{t+k+1}$ that is used as an unbiased sample of $Q^{\pi_\theta}(s_t,a_t)$ in equation 2-27. Parameters θ are then updated by gradient ascent with a learning rate α as $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(a_t \mid s_t)$.

Algorithm 1: REINFORCE

Data: Learning rate α, discount factor γ, episode length H

Initialize θ at random.
while *stopping criterion is not met* **do**
  $(s_0, a_0, r_0, ..., s_{H-1}, a_{H-1}, r_{H-1}) \sim \pi_\theta$
  for $t = 0$ *to* $H-1$ **do**
    $G_t = \sum_{k=0}^{H-t} \gamma^k r_{t+k+1}$
    $\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(a_t \mid s_t,\theta)$
  end for
end while

The REINFORCE algorithm is a great method because of its simplicity, but suffers from high return variance between trajectories, either because of the stochasticity of the environment and/or of the policy (particularly at the beginning of training with a random policy). This leads to a high variance between gradient updates. The simplest solution is to collect more trajectories and aggregate their associated gradients. Indeed, algorithm 1 presents the vanilla REINFORCE algorithm where we generate a single trajectory at a time and update the policy at each step of the trajectory. However, we could generate a batch of trajectories and average gradients between trajectories and steps within these trajectories. Increasing the batch size might reduce sample efficiency, so there is a balance to find.

Another technique to reduce variance substracts a *baseline* value $b(s_t)$ from the computed return G_t. This allows to reduce the variance of gradient estimates while leaving their bias unchanged. The update at time t now becomes,

$$\theta \leftarrow \theta + \alpha\gamma^t(G_t - b(s_t))\nabla_\theta \ln \pi_\theta\left(a_t \mid s_t\right). \tag{2-28}$$

A popular choice for the baseline is the estimated state value: $b(s_t) = v_w(s_t \mid w)$, where v_w can be a neural network parametrized by weights w. Algorithm 2 presents REINFORCE with the state value estimate as a baseline. We now have two learning rates (α_θ and α_w) as both θ and w must be updated.

Algorithm 2: REINFORCE with a state value estimate baseline

Data: Learning rates α_θ and α_w, discount factor γ, episode length H

Initialize θ and w at random
while *stopping criterion is not met* **do**
  $(s_0, a_0, r_0, ..., s_{H-1}, a_{H-1}, r_{H-1}) \sim \pi_\theta$
  for $t = 0$ *to* $H-1$ **do**
    $G_t = \sum_{k=0}^{H-t} \gamma^k r_{t+k+1}$
    $\delta_t = G_t - v_w(s_t \mid w)$
    $w \leftarrow w + \alpha_w \delta_t \nabla_w v_w(s_t \mid w)$
    $\theta \leftarrow \theta + \alpha_\theta \gamma^t \delta_t \nabla_\theta \ln \pi_\theta\left(a_t \mid s_t, \theta\right)$
  end for
end while

REINFORCE with baseline only uses the estimated state value of the first state in each state transition ($s_t \rightarrow s_{t+1}$). There is another class of policy gradient methods known as *Actor-critic* approaches. They are n-step Temporal Difference (TD) methods as they leverage n-step state-value bootstrapping and do not collect full trajectories before updating the policy. In these non-Monte-Carlo methods, we train a neural policy π_θ called the *actor* and a state-value function estimator v_w (or an action-value function estimator q_w) known as the *critic*. Both can either be two distinct neural networks or often share their first layers. Algorithm 3 shows the vanilla one-step actor-critic algorithm, but as for the REINFORCE algorithm, weight updates could be performed based on batches of actions (from different trajectories). Specific implementations of actor-critic methods are A2C and A3C (Mnih et al., 2016).

Finally, we will present a last popular policy gradient method called *Proximal Policy Optimization* (*PPO* – Schulman et al. (2017)). While most policy gradient methods perform a single gradient update per data sample, the objective of PPO is to allow several gradient updates to be performed on each collected trajectory to improve the policy. The risk of iterating several times on the same trajectory without any constraint on the magnitude of weight changes is to perform an unreasonably large policy update in a given gradient direction, leading to potential training instabilities. A method called *Trust Region Policy Optimization* (*TRPO* – Schulman et al. (2015)) was introduced to avoid too strong updates by introducing a constraint on the KL divergence between $\pi_{\theta_{old}}$ before the gradient update and π_θ after. However, TRPO was con-

Algorithm 3: One-step Actor-Critic

Data: Learning rates α_θ and α_w, discount factor γ, episode length H

Initialize θ and w at random
while *stopping criterion is not met* **do**
  Initialize $t \leftarrow 0$, $s_t \sim \mu$, $I \leftarrow 1$
  while s_t *is not terminal* **do**
    $a_t \sim \pi_\theta(\cdot \mid s_t, \theta)$
    $s_{t+1} \sim \mathbb{P}(\cdot \mid s_t, a_t)$
    $r_t = \nabla(s_t, a_t, s_{t+1})$
    $\delta = r_t + \gamma v_w(s_{t+1} \mid w) - v_w(s_t \mid w)$
    $w \leftarrow w + \alpha_w \delta \nabla_w v_w(s_t \mid w)$
    $\theta \leftarrow \theta + \alpha_\theta I \delta \nabla_\theta \ln \pi_\theta(a_t \mid s_t, \theta)$
    $t \leftarrow t + 1$, $I \leftarrow \gamma I$, $s_t \leftarrow s_{t+1}$
  end while
end while

sidered a complicated method and was not compatible with neural operations dealing with noise (e.g. dropout) or with parameter sharing between networks (e.g. between actor and critic). The objective of PPO was thus to keep the benefits of TRPO while having a simpler method to implement, and that is more general and scales better to more problems. Let us now present this approach.

PPO alternates between sampling and optimization phases. At sampling time, the policy π_θ is used to collect a set of trajectories that are stored for the optimization phase. During the latter, batches of trajectories are replayed to compute a loss that will be differentiated with respect to weights θ to be updated. The same trajectory will be involved in several backward passes while ensuring each update does not move the new weights too far from the previous ones. More specifically, at sampling time k, a set $\mathcal{U}_k$ of trajectories τ with length T are collected using the latest policy π_θ. Note that T is smaller than the length of a full episode. If we write the PPO loss to minimize by gradient descent as done in most Deep Reinforcement Learning (DRL) implementations (instead of an objective to maximize with gradient ascent), we have,

$$\mathcal{L}_{PPO} = \frac{-1}{|\mathcal{U}_k|\, T} \sum_{\tau \in \mathcal{U}_k} \sum_{t=0}^{T-1} \left[\min\left(\rho_t(\theta) \hat{A}_t^{\pi_\theta}, \mathcal{C}(\rho_t(\theta), \epsilon) \hat{A}_t^{\pi_\theta} \right) \right] \tag{2-29}$$

where,

- ϵ is a hyperparameter to fix.
- $\rho_t(\theta) = \frac{\pi_\theta(a_t|s_t,\theta)}{\pi_{\theta_{\text{old}}}(a_t|s_t,\theta_{old})}$ is the probability ratio between the updated and old versions of the policy.
- $\mathcal{C}(\rho_t(\theta), \epsilon) = \text{clip}\left(\rho_t(\theta), 1-\epsilon, 1+\epsilon\right)$

- $\hat{A}_t^{\pi_\theta}$ is an estimate of the advantage function $A^{\pi_\theta}(s_t, a_t)$ at time t.

There are two terms inside the $min(.)$ operation. The first one ($\rho_t(\theta)\hat{A}_t^{\pi_\theta}$) is the objective function to maximize, i.e. increasing the probability associated to actions that lead to an increased reward (positive estimated advantage). The second one ($\mathcal{C}(\rho_t(\theta), \epsilon)\hat{A}_t^{\pi_\theta}$) clips $\rho_t(\theta)$ between $1-\epsilon$ and $1+\epsilon$. The minimum between both unclipped and clipped objectives is then taken to ensure the final objective is a lower bound on the unclipped objective. Interestingly, when considering the gradients computed from such a loss, this leads to updating the policy only either when the probability ratio is within the chosen range, or lower than $1-\epsilon$ with a positive advantage, or higher than $1+\epsilon$ and with a negative advantage (in both cases, bringing the ratio closer to the chosen range after the update).

> **Context**
>
> DRL is used in different chapters of this manuscript (chapters 4, 5 and 6) to train autonomous navigating agents. In particular, we use the PPO algorithm in these studies because of its practical simplicity and efficiency.

2.3.4 *Imitation Learning*

A particularly interesting type of supervised learning when training agents to interact is imitation learning and, more specifically, Behavior Cloning (BC). The goal in BC is to generate expert trajectories and train an agent to mimic them. If we consider a problem that can be formalized as a POMDP, we will consider a neural policy π_θ implemented as a neural network mapping an input observation to a distribution over actions. More importantly, with BC, we do not consider the reward function $\mathcal{R}$, but instead train π_θ from a dataset of demonstrations obtained by executing an expert policy π^* (the term *policy* includes all types of experts including human demonstrators). Here, $\mathbf{x}_i$ will be a sequence of observations ($\mathbf{x}_i = \{o_{i,0}, ..., o_{i,H}\}$) and $\mathbf{y}_i$ a sequence of expert actions ($\mathbf{y}_i = \{a^*_{i,0}, ..., a^*_{i,H}\}$). The Behavior Cloning objective can then be defined as,

$$\hat{\theta}^* = \arg\min_\theta \sum_{i=1}^{n} \sum_{h=0}^{H} \mathcal{L}(\pi_\theta(o_{i,h}; \theta), \pi^*(o_{i,h})) \tag{2-30}$$

$$= arg\min_\theta \sum_{i=1}^{n} \sum_{h=0}^{H} \mathcal{L}(\pi_\theta(o_{i,h}; \theta), a^*_{i,h}), \tag{2-31}$$

where $\mathcal{L}(.)$ is a loss function (Mean Squared Error to regress continuous actions, or Cross Entropy to classify discrete actions).

However, a first challenge in BC is to access an expert agent to generate training trajectories. Such an agent could be a human operator (Shafiullah et al., 2023; Iyer et al., 2024; Fu et al., 2024), or sometimes task-specific policies reaching state-of-the-art performance on a single task (Majumdar et al., 2023). Task-specific policies can be used as experts to gather

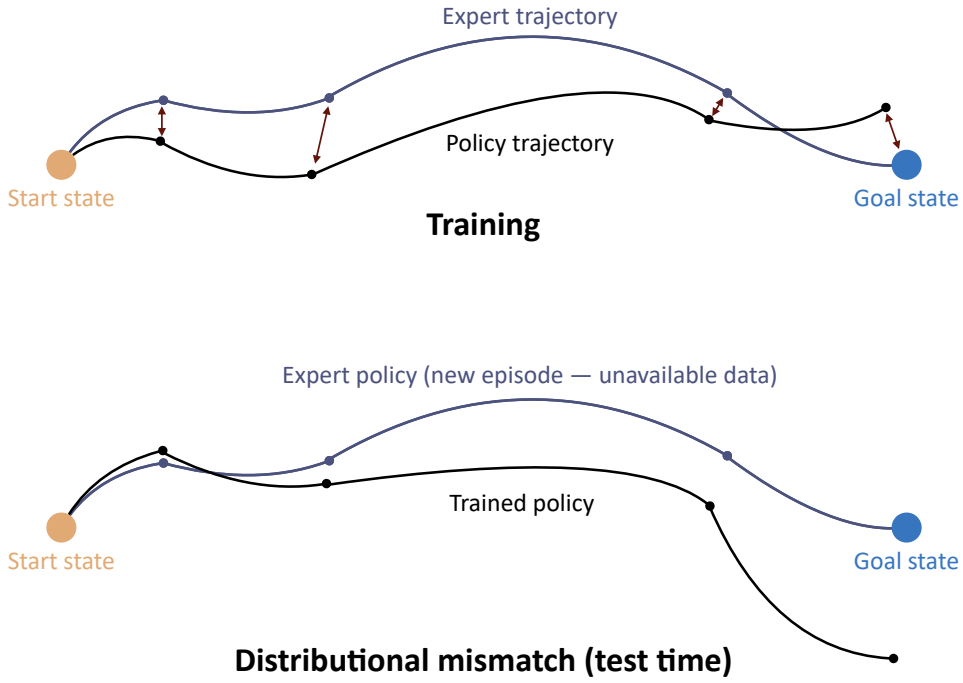


Figure 2.12: **Behavior Cloning objective and distributional mismatch**: Illustration of the Behavior Cloning objective at training time and the problematic distributional mismatch that may occur at test time.

trajectories from different tasks and training a single general policy. Importantly, it is not because an expert performs better than another on the task at hand that data collected with it will lead to better downstream policy performance. Indeed, the problem of expert demonstrations that can not be reproduced by a policy without access to privileged information has been studied (Ramrakhya et al., 2023). It is thus crucial to collect realistic trajectories that can be followed with available information.

A second well-known challenge with BC is generalization to states which have a low likelihood to be visited by π^*. Indeed, a severe drawback of BC is that π_θ will only be trained on states visited by π^*, reducing its generalization to out-of-distribution states (Figure 2.12). In particular, after training, when acting with π_θ from new starting states, it will be able to imitate the behavior of the expert policy, but will eventually drift after some time: this is when it will start visiting unknown states, leading to poor performance. Regarding this last issue, a simple yet powerful idea, presented by Ross et al. (2011), is to collect new training data as needed. A quite strong assumption is the availability of the expert policy when required. Then, whenever π_θ encounters new states, π^* can be queried to generate ground-truth actions for these states. The introduced method is called DAgger (Dataset Aggregation).

The issues encountered with BC are largely related to the inability of π_θ to interact with an environment of interest. Despite the simplicity of framing the problem of learning to act as a supervised learning task, taking actions and observing the impact of the latter on an

environment overcomes the need to find an expert and the potential mismatch between states at training and test times.

> **Context**
> In chapter 7, we use BC to train a multi-task policy on different tasks by leveraging diverse task-specific expert policies.

2.3.5 *Discrete planning*

Planning is an important module of the autonomy stack (LaValle, 2006): it is the ability to create a plan to execute in order to reach a goal. We will only consider a subset of planning methods, i.e. discrete planning, while many other approaches exist such as sampling-based planners like RRT* (Karaman and Frazzoli, 2011).

Discrete planners require a discretized environment, i.e. composed of a finite number of states (or at least a countably infinite state space). Dijkstra's algorithm (Dijkstra, 1959) is a path-finding method applied to problems formalized as weighted graphs where nodes represent locations in an environment and edge weights the cost to move from one location to another. Given an oriented graph $G{=}(V, E)$ defined by a set of vertices V, a set of edges E and a cost function $c : E \to R^+$ mapping each edge, i.e. a pair of linked vertices, to a positive cost, Dijkstra's algorithm allows to find the shortest path from a source node v_0 to all other vertices in the graph. Other discrete planning methods exist such as the A* algorithm (Hart et al., 1968, 1972; Dechter and Pearl, 1985). It could be seen as an extension of Dijkstra's algorithm, but instead of returning the shortest-path tree from a given node to all other vertices in the graph, A* only finds the shortest path from a source to a node as it uses goal-specific heuristics to guide the search for a shorter path. This goal-specific strategy allows A* to be more efficient in practice at finding the shortest path for a given (source, goal) pair.

However, discrete planning problems have also been addressed with numerical methods such as the Fast Marching Method (FMM – Sethian (1996)). FMM can be considered as an extension of Dijkstra's algorithm where the graph update is replaced by a local resolution of the Eikonal equation describing a wave propagation. Garrido et al. (2006) detail how fast marching methods can be used to perform path planning.

> **Context**
> The Fast Marching Method will be used in chapter 6 of this manuscript to plan a path from the position of an agent to a long-term goal to reach on a 2D top-down grid map of a considered environment.

2.3.6 *Robotic tasks involving sequential decision-making*

Finally, we present below two families of tasks that we will be interested in and that require performing some sequential decision-making, but also building scene representations.

Visual Navigation – is an important task in the field of robotics and has thus been extensively studied (Bonin-Font et al., 2008; Thrun et al., 2005). It consists of an agent placed in an environment, that can be unknown, and that must solve a specified task based on visual input. Bonin-Font et al. (2008) distinguish *map-based*, *map-building-based* and *map-less* navigation. This is indeed an important difference as in *map-based* visual navigation, the robot agent is provided with a model of the environment, with different levels of details depending on the application and the available information, while in *map-building-based* and *map-less* navigation, no prior information about the scene the agent will be navigating in is available. *Map-less* navigation approaches do not require any representation of the environment as they will navigate from environment observations only, with optical flow- and appearance-based navigation being two well-known techniques. Finally, *map-building-based* navigation methods will map their environment in real time while navigating as they rely on an explicit scene representation to achieve their goal. Another important difference is between indoor and outdoor navigation. Outdoor navigation might involve harder navigation conditions with more challenging terrains and longer routes to drive, but on the other hand, indoor navigation will require the agent to navigate in narrower spaces with potentially more obstacles and constrained layouts.

> **Context**
>
> In this manuscript, we consider a specific variant of visual navigation, i.e. indoor and map-building-based, where an agent is placed in a completely unknown indoor environment and must fulfill a goal. In particular, we consider two types of goals, presented later in more detail: Multi-object Navigation (chapters 4 and 5), where an agent must reach target objects in a specific order, and a variant of active scene exploration to collect NeRF training data (chapter 6).

Manipulation – is another important subdomain of robotics where an agent is tasked with interacting with physical objects to achieve a defined goal. Kroemer et al. (2021) review manipulation challenges from a learning perspective, i.e. *what are sub-problems to solve to train neural agents to manipulate objects?* A first one is learning object and environment representations. The notion of *objectness* is indeed a strong semantic prior humans are equipped with: when perceiving the world, we are able to decompose the visualized scene into entities with specific properties. A second core topic is about learning transition models, i.e. being able to model the changes in the states of objects from the actions of the agent manipulating them. Such representations are also referred to as world models (Ha and Schmidhuber, 2018) or dynamics models (Lesort et al., 2018) in the broader literature. A third and final important topic is about acquiring skills (e.g. with Reinforcement Learning or Behavior Cloning) that can be reusable between different tasks and with diverse levels of abstraction (high-level planning skills and lower-level control abilities).

Context

Chapter 7 in this manuscript will deal with a set of tasks where some of them involve manipulating objects. We will study how visual features should be adapted depending on manipulation tasks to target and whether a policy can be adapted to new manipulation tasks from a few demonstrations.

Chapter

Progress in Embodied AI

Now that we have introduced important preliminary concepts, we will review works in the field of Embodied AI related to the contributions presented in this manuscript. More specifically, we will start by presenting simulation methods, i.e. datasets, simulators and tasks, allowing to train and evaluate neural agents efficiently, before reviewing proposed neural agent architectures, with a specific focus on how to encode past experience and represent 3D scenes, and will finally present different ways such agents can be trained. Importantly, many of the presented concepts find their roots in prior literature (e.g. some considered Embodied AI tasks), but we will focus here on their specificities in the context of Embodied AI.

As done in the previous chapter, we will also put the presented work into perspective with respect to the studies that will be conducted in the next chapters with the same box notations. Although not limited to it, this section somewhat focuses on navigation problems. We will also refer to non-navigation tasks in the context of multi-task policy learning.

3.1 Simulation

Simulating the real world is an important ability to safely experiment with autonomous agents, in particular when the latter are neural-based and must be trained through trial and error. Indeed, directly experimenting in the real world can have several disadvantages: experimentation is slow and dangerous as the considered neural policies can be suboptimal during training or evaluation, and experimental control and reproducibility are hard to reach as the environment might be changing often.

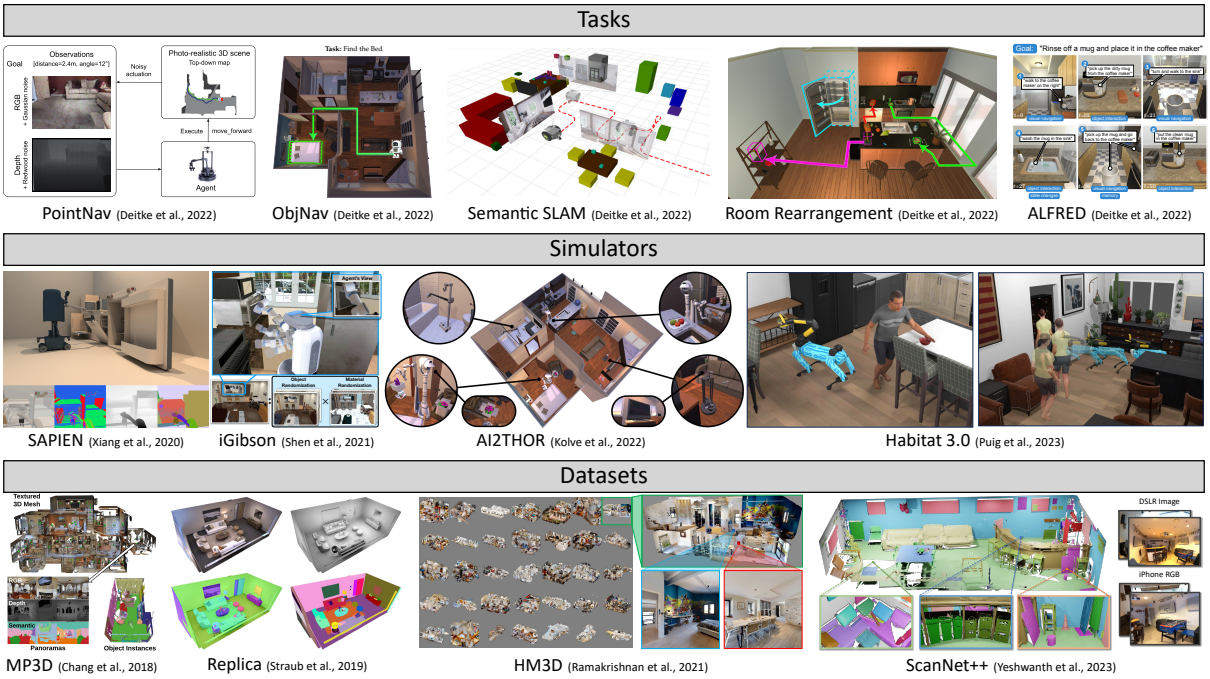


Figure 3.1: **The simulation *software stack*** from Savva et al. (2019) – inspired by Figure 1 in Savva et al. (2019).

Inspired by the *software stack* required to train embodied agents presented by Savva et al. (2019), we can indeed formalize the simulation setup as a hierarchy of three components: (i) at the root, *datasets* providing 3D assets for scenes, objects, eventually accompanied by semantic annotations (e.g. object classes, room names), (ii) then *simulators* that can load datasets and render observations (e.g RGB, depth, semantic masks) and emulate all types of sensors (e.g. odometry, LIDAR) and actuators, and (iii) finally *tasks* formalizing problems to solve involving datasets, observation spaces, associated sensors and goal specifications. Figure 3.1 provides an overview of this hierarchy of concepts.

Datasets – are an important component when training embodied agents in simulation. We will present the evolution in indoor scene datasets ranging from the ScanNet (Dai et al., 2017), Matterport3D (MP3D – Chang et al. (2018)), Gibson (Xia et al., 2018), Replica (Straub et al., 2019) and RoboTHOR (Deitke et al., 2020) datasets to the more recent HM3D (Ramakrishnan et al., 2021a; Yadav et al., 2023b) and ScanNet++ (Yeshwanth et al., 2023) datasets.

All mentioned datasets provide RGB mesh representations of indoor scenes along with semantic annotations. There are differences in how these datasets were collected, but we will rather focus here on another main difference, i.e. the evolution in the characteristics of introduced datasets. Table 3.1 is adapted from Ramakrishnan et al. (2021a) and shows such an evolution with regard to different metrics: *number of scanned scenes*, *total floor area covered*, *navigable area*, *navigation complexity* and *scene clutter*. As defined by Ramakrishnan et al. (2021a), *navigation complexity* measures how hard it is to navigate a scene and is computed as the maximal ratio between the geodesic and Euclidian distance between any two randomly

	Room-scale			Building-scale		
Dataset	ScanNet	Replica	RoboTHOR	MP3D	Gibson (4+ only)	HM3D
Year	2017	2019	2020	2018	2018	2021a
Number of scenes	1613	18	75	90	571(106)	1000
Floor area (m^2)	39.98k	2.19k	3.17k	101.82k	217.99k(17.74k)	365.42k
Navigable area (m^2)	10.5k	0.56k	0.75k	30.22k	81.84k(7.18k)	112.50k
Navigation complexity	3.78	5.99	2.06	17.09	14.25(11.90)	13.31
Scene clutter	3.15	3.4	8.2	2.99	3.14 (3.04)	3.90

Table 3.1: **Evolution in the characteristics of indoor datasets** – adapted from Ramakrishnan et al. (2021a)

sampled points in the scene. The *scene clutter* evaluates the amount of obstacles available in the scene and is computed as the ratio between the raw scene mesh area within 0.5m of the navigable regions and the navigable space. We should first differentiate two types of datasets: room-scale datasets where a scene corresponds to a single room and building-scale datasets where each scene spans multiple rooms and even different floors.

The study shown in Table 3.1 is mainly biased toward studying the relevance of available datasets to train and/or evaluate agents on navigation tasks. Room-scale datasets, because of the small size of their scenes, are thus losing on most of the considered metrics. However, such datasets are still very important as they provide additional data that can complement building-scale datasets and are potentially better suited to tasks involving both navigating and interacting with objects. Indeed, an interesting trend is the increase in scene clutter, particularly true for the RoboTHOR dataset (Deitke et al., 2020). It is important to notice how large the ScanNet dataset (Dai et al., 2017) was in terms of number of scanned scenes in 2017. Both Replica and RoboTHOR seem to indicate the same trend regarding room-scale datasets: the goal has shifted toward less but higher-quality scene scans, focusing on other characteristics such as navigation complexity or scene clutter.

The more interesting trend is when it comes to building-scale datasets: as greatly shown by the HM3D dataset (Ramakrishnan et al., 2021a), recent datasets have increased in size, both in terms of number of scenes, floor areas, and navigation areas. Indeed, training general embodied agents requires diverse data. Navigation complexity and scene clutter have not evolved significantly, showing that the focus has rather been on the scale of 3D scenes.

The ScanNet++ dataset (Yeshwanth et al., 2023) is not studied by Ramakrishnan et al. (2021a) as it came later, but should be classified as a room-scale dataset. While it contains fewer scenes (460) than the original ScanNet (Dai et al., 2017), the authors focus on the quality and resolution of the reconstruction, mainly focusing on novel view synthesis. This choice validates our assumption about the decrease in size and increase in quality of room-scale datasets.

Figure 3.2: **Semantic annotations in HM3DSEM** – reproduced from Yadav et al. (2023b)

Despite the scale, semantic annotations of 3D scenes can also become important when implementing tasks dealing with semantics. In this sense, the Habitat-Matterport 3D Semantics Dataset HM3DSEM (Yadav et al., 2023b) provides a semantic annotation layer on top of the HM3D scenes. Figure 3.2 provides visual samples. 143*k* object instances were annotated, covering 216 scenes and a total of 3100 rooms. Other presented datasets also provide semantic annotations, but HM3DSEM is currently the largest one to date.

> **Context**
>
> The work presented in this manuscript mainly leverages the Gibson and MP3D datasets (chapters 4, 5 and 6) as they both provide many 3D scenes with high navigation complexity, and were chosen standards in considered tasks.

Simulators – In the context of Embodied AI tackling the deployment of neural-based agents in the real world, simulation must have two important properties: (i) photo-realism, i.e. rendering realistic environments to narrow the *sim-to-real* gap, (ii) allowing fast simulation to be able to train neural networks with reasonable compute times. Simulators leverage previously presented datasets made of environment scans and/or 3D assets that they will load to render observations provided by emulated suites of sensors.

Many simulators were introduced in the past years. Just as datasets, simulators have evolved from Deepmind Lab (Beattie et al., 2016) or VizDoom (Kempka et al., 2016) rendering 3D mazes to newer simulators trying to come closer to real-world rendering and summarized in Table 3.2 adapted from Szot et al. (2021). As can be seen, simulators employ different solutions to perform rendering and handle physics. There are also differences in the complexity of handled scenes and rendering speed.

When thinking about simulating 3D space, one will likely consider the rendering and physics simulation quality, but the efficiency of the simulation is also primordial when considering training neural agents. Indeed, the realism of the rendering and the simulated physics is important, but if they come at the price of a too-slow simulation and/or difficulty in handling

	Rendering		Physics		Scene	Speed
	Library	Supports	Library	Supports	Complexity	(steps/sec)
Habitat (Savva et al.)	Magnum	3D scans	none	continuous navig. (navmesh)	building-scale	3000
AI2-THOR (Kolve et al.)	Unity	Unity	Unity	rigid dynamics, animated interactions	room-scale	30 - 60
ManipulaTHOR (Ehsani et al.)	Unity	Unity	Unity	AI2-THOR + manipulation	room-scale	30 - 40
ThreeDWorld (Gan et al.)	Unity	Unity	Unity (PhysX) + FLEX	rigid + particle dynamics	room/ house-scale	5 - 168
SAPIEN (Xiang et al.)	OpenGL/ OptiX	configurable	PhysX	rigid/articulated dynamics	object-level	200 - 400
RLBench (James et al.)	CoppeliaSim (OpenGL)	Gouraud shading	CoppeliaSim (Bullet/ODE)	rigid/articulated dynamics	table-top	1 - 60
iGibson (Shen et al.)	PyRender	PBR shading	PyBullet	rigid/articulated dynamics	house-scale	100
Habitat 2.0 (Szot et al.)	Magnum	3D scans + PBR shading	Bullet	rigid/articulated dynamics + navmesh	house-scale	1400
Habitat 3.0 (Puig et al.)	Habitat 2.0 + simulation of humans					

Table 3.2: **Comparison of indoor simulators** – adapted from Szot et al. (2021)

large enough scenes, this will not allow proper experimentation. A focus of some simulators such as the first version of Habitat (Savva et al., 2019) has been on simulation speed to allow training agents with RL in a reasonable (can still be considered as too long!) time. In their first version, they indeed decided to simplify physics to focus on fast simulation of large 3D scenes to train autonomous agents to navigate. Their original paper shows that end-to-end neural agents can actually reach greater performance than what had been previously shown (Mishkin et al., 2019) when scaling training experience. Habitat 2.0 (Szot et al., 2021) proposes a better physics simulation without sacrificing speed thanks to additional optimizations such as cleverly interleaving GPU-based rendering and CPU-based physics simulation. A newer version, Habitat 3.0 (Puig et al., 2023), now integrates simulated humans inside 3D scenes. Figure 3.3 shows examples of RGB-D observations rendered with the Habitat simulator, along with the computation of a navigable top-down map from the mesh of a 3D scene and of unexplored and explored areas by taking the location of the agent into account.

Physics simulation should still remain an important topic, particularly when targeting tasks involving fine control. Indeed, as Habitat is mainly designed for high-level navigation and object interaction, a more accurate simulation of physics might not be as critical. A famous example of a physics simulator is MujoCo (Todorov et al., 2012) which was specifically designed to efficiently simulate robot multijoint dynamics. It was originally designed to benchmark model-based control approaches, but can also be used to train neural agents to control actuators.

Figure 3.3: RGB-D observations and top-down maps rendered with the Habitat simulator

Context

We used the Habitat Simulator in our work dealing with autonomous visual navigation (chapters 4, 5 and 6) because of its fast and realistic rendering. MujoCo was also used to simulate tasks that were not compatible with the Habitat framework (chapter 7).

Tasks and benchmarks – An important concept in research is known as the *Common task framework* (Donoho, 2017), referring to the importance for a given field to follow a standard evaluation methodology in order to properly study the progress towards a common goal. A practical implementation of such a concept is the introduction of formalized tasks and benchmarks to compare methods. In the case of Deep Learning, it requires the use of common datasets and metrics to fairly evaluate the gain brought by a new approach compared with previous work. We will start by presenting some navigation tasks studied by the Embodied AI community along with their associated metrics, before mentioning another interesting benchmark focusing on the evaluation of visual representations for multi-task policy learning.

As already surveyed (Bonin-Font et al., 2008), many navigation tasks have been introduced by the robotics community. We will now review recent tasks considered in Embodied AI, which are sometimes specific implementations of existing general tasks to be better suited to neural-based approaches. We will provide implementation details of certain tasks as they were originally presented but note that follow-up work has sometimes modified them (observation, action spaces, success constraints, metrics) and that other specifications could lead to

identically-interesting results. It is also important to know that tasks are often implemented on a specific simulator and methods to address them can thus be restricted to it.

A series of tasks and metrics to evaluate embodied navigation agents were initially introduced by Anderson et al. (2018a). Most of the considered navigation tasks are framed as goal-reaching problems, where the nature of the goal to reach could be different. Deitke et al. (2022) present the evolution of the different challenges implementing these common tasks and the improvement of solutions over the years. We will only focus here on a subset of navigation tasks that have been extensively studied and are relevant in the context of this manuscript, including a non-goal-reaching one, i.e. scene exploration.

We will start by presenting the similarities between all these tasks before detailing their specificities. A first common trait between all these navigation tasks is about the environments used to train and evaluate neural agents: different scenes are used at training and evaluation time, with episodes varying based on the starting agent's location and the goal to find when we have a goal-reaching task. The purpose of the introduced tasks is thus to benchmark the ability of agents to generalize to any new 3D environment without any a priori knowledge about it. In all considered tasks, at the beginning of an episode, the agent is placed at a random location within the scene. The observation available to the agent at timestep t always includes an egocentric RGB image $\mathbf{r_t} \in \mathbb{R}^{H \times W \times 3}$ (H and W define the height and weight of the image) from an RGB camera with specific characteristics (e.g. height and field of view) and an optional depth frame $\mathbf{d_t} \in \mathbb{R}^{H' \times W'}$. Additional sensors can be provided depending on the task. Each task is associated with a maximum number of discrete timesteps before an episode is over.

PointNav is a visual navigation task requiring an agent to reach a specific location in a previously unseen 3D environment. There are actually two variants of this task: the point to reach is specified in terms of Euclidian distance and direction, either relative to the agent's starting location in the episode (*Static PointNav*), or updated to be provided relative to the current agent's position at each timestep (*Dynamic PointNav*). Additionally to RGB and optional depth, a sensor to specify the goal to reach (either always returning the same Euclidian distance and angle relative to the agent's start position, or updated at each timestep to compute these relative to the current agent's location) is included. An example action space (considered in the original *PointNav* task) is composed of 4 discrete high-level actions: *Move Forward 0.25m*, *Rotate Right* 30°, *Rotate Left* 30°, *Done*. Other action spaces, e.g. continuous, could be used instead of the specific one considered here. The *Done* action allows the agent to specify it considers being close enough to the goal to find and stops the episode here. An episode is successful if the agent calls the *Done* action while being closer to the goal location than a threshold distance (e.g. $0.2m$) and if the length of the episode is smaller than a maximum number of steps (e.g. 500 steps).

Scene exploration is a longstanding problem (Yamauchi, 1997) and has been studied in Embodied AI as a coverage maximization problem (Chen et al., 2018). Unlike in *PointNav*, an odometry sensor is also often provided, to estimate (sometimes perfectly) the agent's motion between two timesteps. Some work also introduces a bump sensor indicating a collision with an obstacle (Chen et al., 2018; Ramakrishnan et al., 2021b). The goal is to maximize the scene floor coverage within a fixed time budget. The action space introduced by Chen et al. (2018) was composed of 6 discrete actions: *Move Forward 0.25m*, *Move Backward 0.25m*, *Strafe Left 0.25m*, *Strafe Right 0.25m*, *Rotate Left* 9°, *Rotate Right* 9°. There is no goal to reach, the episode stops when the steps budget is over.

ObjNav is also about reaching a specified goal in a new 3D scene. However, instead of a location to reach as in *PointNav*, the agent must navigate to the closest instance of a given object class (e.g. *chair*, *bed*, *potted plant*). Providing a high-level abstract description of the goal instead of its specific location forces the agent to perform *scene exploration* to locate it before navigating towards it. A goal sensor specifying the object class to find is provided along with an optional odometry sensor. Depending on the implementation of the task (e.g. *RoboTHOR ObjNav Challenge* vs *Habitat ObjNav Challenge*), the same action space as in *PointNav* is used, with eventually additional actions such as *Look Up* and *Look Down* to allow the agent to rotate its camera around the horizontal axis. We consider an episode to be successful if the agent calls *Done* before the time limit while the goal is within the camera's field of view and the agent is closer to it than a threshold distance (e.g. $1m$).

ImageNav is another goal-reaching task where the goal to reach is specified as an image. The latter was initially an image taken from the target location to reach (Chaplot et al., 2020d; Hahn et al., 2021), with different specificities depending on the work, e.g. 360° panoramic images in Chaplot et al. (2020d). More recently, Krantz et al. (2022) proposed a new version of *ImageNav* called *InstanceImageNav* addressing two limitations of previous implementations of the *ImageNav* task. First, instead of randomly sampling goal locations which could lead to uninformative images (e.g. photo of a white wall), they only sample target images containing a central well-depicted object. Secondly, while target images were initially obtained from the same RGB camera as the one available on the agent, they follow a more general setup where target images can be captured from cameras with other settings. These two improvements allow the task to be more aligned with potential user requirements, i.e. taking a picture of a specific object a robot might need to navigate to with a camera that is not the one of the robot itself (e.g. a smartphone). The task thus now appears very similar to *ObjNav* with only a difference in the goal-specific sensor, as the agent also has to navigate as close as possible to an object that was captured by an external camera.

Sequential Tasks (Beeching et al., 2020a; Wani et al., 2020) were introduced with two impor-

tant characteristics, *(i)* their sequential nature, i.e. an episode is composed of a sequence of goals to reach, and *(ii)* the use of external objects as target objectives, i.e. the objects to find are not part of the scanned 3D scenes used as environments (unlike in *ObjNav*), but are for example randomly placed colored cylinders as in Wani et al. (2020).

Multi-Object Navigation (*Multi-ON* – Wani et al. (2020)) is a task requiring to sequentially retrieve objects, but unlike the *Ordered K-item* task (Beeching et al., 2020a), the order is not fixed between episodes. As in other tasks, the agent is initialized at a random location in an unknown 3D scene at the beginning of each episode. It has access to an RBG-D camera, an odometry sensor and a goal sensor specifying the target object to reach. Once a target is reached, the goal sensor will update its output with the new goal in the sequence. Importantly, the order between goals to find is not known a priori, i.e. the agent only knows about the target to find at the current timestep. The action space is composed of 4 actions: *Move Forward 0.25m*, *Rotate Right* 30°, *Rotate Left* 30°, *Found*. *Found* is similar to *Done* in *PointNav* as it allows to specify the current target has been reached. We will still use the name *Found* when mentioning *Multi-ON* as it is how it was introduced by Wani et al. (2020). An implementation of this task is known as *3-ON*, where the agent must navigate to 3 target objects in a row. There are 8 classes, corresponding to 8 possible colors for the cylinders. 3 cylinder colors are thus sampled for each episode, and the 3 cylinders are randomly localized in the scene. Figure 3.4 provides an overview of the task. An episode is considered successful if the agent correctly finds all goals within the time limit (2500 steps), where reaching a goal means calling the *Found* action while being close enough to it (1m).

A sequential task is interesting as it requires the agent to remember and to map potential objects it might have seen while exploring the environment, as reasoning on them might be required in a later stage. Moreover, using external objects as goals prevents the agent from leveraging knowledge about the environment layouts, thus focusing solely on memory. Exploration is another targeted capacity as objects are placed randomly within environments.

Navigation metrics are important to assess the quality of the paths taken by agents to solve a task. Anderson et al. (2018a) introduced two metrics, *Success Rate* (*SR*) and *Success weighted by Path Length* (*SPL*) that are used in *PointNav*, *ObjNav*, and *ImageNav*. If we denote s_i as the binary success (1 if successful, 0 otherwise) for episode i among N evaluation episodes, we have,

$$SR = \frac{1}{N} \sum_{i=1}^{N} s_i, \tag{3-1}$$

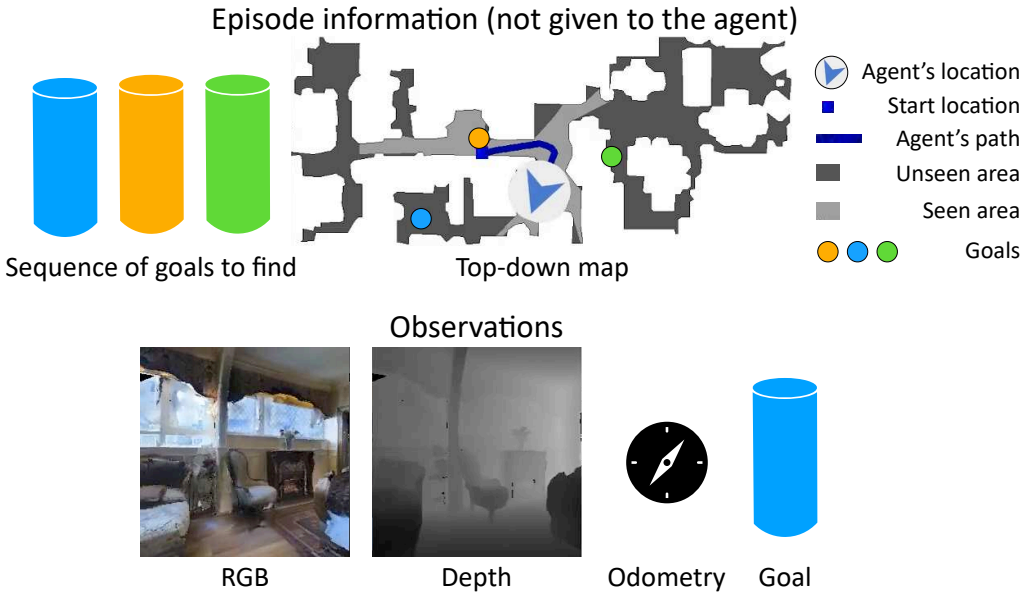


Figure 3.4: ***Multi-ON* (*3-ON*) task overview**

$$SPL = \frac{1}{N}\sum_{i=1}^{N} s_i \frac{l_i}{\max(p_i, l_i)}, \tag{3-2}$$

where l_i is the shortest path from the start to the goal location and p_i is the length of the path taken by the agent. While success rate is a useful metric, *SPL* is particularly interesting because it measures both the ability of the agent to reach goals, but also the efficiency of the paths it takes.

The *SR* metric can be used in *Multi-ON* based on the mentioned definition of a successful episode. As it is a sequential navigation problem, *SPL* must be adapted. The sum term for an episode i becomes $s_i\tilde{l}_i / \max(p_i, \tilde{l}_i)$, where p_i is still the distance of the agent's path, but $\tilde{l}_i = \sum_{k=1}^{K} d_{k-1,k}$ with $d_{k-1,k}$ denoting the shortest path from the $(k-1)$th to the kth goal ($d_{0,1}$ is the geodesic distance from the starting point to the first goal).

Multi-ON also introduced two new metrics: *Progress Rate* (*PR*) and *Progress weighted by Path Length* (*PPL*). The *PR* metric corresponds to the average percentage of found objects in all episodes ($PR = SR$ in the *1-ON* scenario where there is a single goal to find). *PPL* is an extension of *SPL* replacing success with progress,

$$PPL = \frac{1}{N}\sum_{i=1}^{N} \bar{s}_i \frac{\bar{l}_i}{\max(p_i, \bar{l}_i)}, \tag{3-3}$$

where $\bar{s}_i$ is the progress of episode i and $\bar{l}_i = \sum_{k=1}^{F} d_{k-1,k}$ with F denoting the number of correctly found targets ($PPL = SPL$ in the *1-ON* scenario where there is a single goal to find).

Finally, specific metrics are used to evaluate the quality of the scene coverage performed

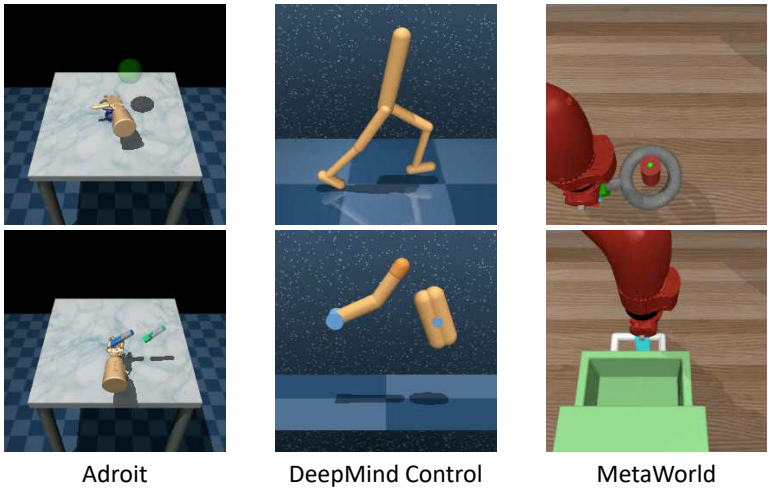


Figure 3.5: Task samples from *Adroit*, *DeepMind Control* and *MetaWorld*

by an agent in *scene exploration*. Two popular ones are the *absolute coverage area* in m^2, and the *percentage of area explored* computed as the ratio between the covered area and the total scene area.

Many other navigation tasks have been introduced, either specifying goals to reach as language queries in *Vision-Language Navigation* (*VLN*) (Anderson et al., 2018b; Shridhar et al., 2020; Thomason et al., 2020), or mixing navigation and object manipulation in *Rearrangement* tasks (Batra et al., 2020). An example of the latter is the *Mobile-Pick* task where an agent must navigate toward an object whose location is specified (3D center of mass), pick it, and bring it to an end location (specified again as a target 3D center of mass).

> **Context**
>
> Some work presented in this manuscript focuses on studying the ability of neural-based agents to map unknown environments. We thus used Multi-Object Navigation in chapters 4 and 5 as our evaluation test-bed based on its mentioned advantages to benchmark mapping abilities.

The previously introduced tasks evaluate the ability of agents to perform a specific navigation behavior robustly in previously unseen environments, thus benchmarking their generalization to new scenes. However, another interesting direction is about addressing different tasks with the same agent, or with the same agent architecture and training protocol. An assumption is that generalization abilities could come from large pre-trained vision models, allowing to extract relevant visual features for different tasks of interest. A new benchmark, called *Cortexbench* (Majumdar et al., 2023), was thus introduced to evaluate such an idea.

Cortexbench is not composed of any new task but is instead a gathering of known benchmarks spanning diverse robotic tasks. The chosen tasks come from 5 existing benchmarks:

- *Habitat*: Visual navigation tasks (*ObjNav*, *ImageNav* and *Mobile-Pick*).
- *Adroit* (Rajeswaran et al., 2018): 28-DoF hand control for manipulation tasks (*Relocate*, *Reorient-pen*).
- *DeepMind Control* (Tassa et al., 2018): continuous control tasks dealing with locomotion and manipulation (*Finger-Spin*, *Reacher-Hard*, *Cheetah-Run*, *Walker-Stand*, *Walker-Walk*).
- *MetaWorld* (Yu et al., 2020): robot arm (Sawyer robot arm) control for object manipulation tasks (*Assembly*, *Bin-Picking*, *Button-Press*, *Drawer-Open*, *Hammer*).
- *TriFinger* (Wüthrich et al., 2020) 3-finger hand (3-DoF per finger) control for manipulation tasks (*Push-Cube*, *Reach-Cube*)

Context

Chapter 7 of this manuscript studies the ability to adapt visual representations to the task at hand. To this end, we leveraged a subset of the *Cortexbench* benchmark (*Adroit*, *DeepMind Control*, *MetaWorld*) to evaluate our method on a standard set of tasks. Figure 3.5 presents two task samples from *Adroit*, *DeepMind Control* and *MetaWorld*.

3.2 Encoding past experience

Interacting with a 3D environment can be framed as a learning problem, leveraging the abilities of deep networks to extract regularities from training data. Agents can be reactive (Zhu et al., 2017a), but recent work tends to augment them with memory, which is a key component, in particular in partially-observable environments (Hausknecht and Stone, 2015; Oh et al., 2016). An important question is then: *How to encode past experience?*

3.2.1 *Recurrent memory*

The simplest representation of the past is the memory of an RNN. If we come back to the concept of inductive bias introduced earlier, such recurrent memory is built with very weak priors in mind, the only one being that the past should be encoded in a vectorial form. Wijmans et al. (2019) showed that *PointNav* could be solved with a simple policy equipped with only a recurrent memory, requiring however large training experience ($2.5B$ frames of experience). Interestingly, Wijmans et al. (2022) even showed that map-based structures emerged in the hidden memory of recurrent blind agents learning to navigate without access to vision. This thus indicates that a recurrent memory could be enough to learn to map a scene. This is echoing more recent work (Bono et al., 2023b) showing that in addition to training a recurrent agent to navigate, supervising the construction of its hidden memory by feeding it to a blind agent to navigate enriches the map-related stored information. However, depending on the task to solve, a more structured scene representation could still be beneficial.

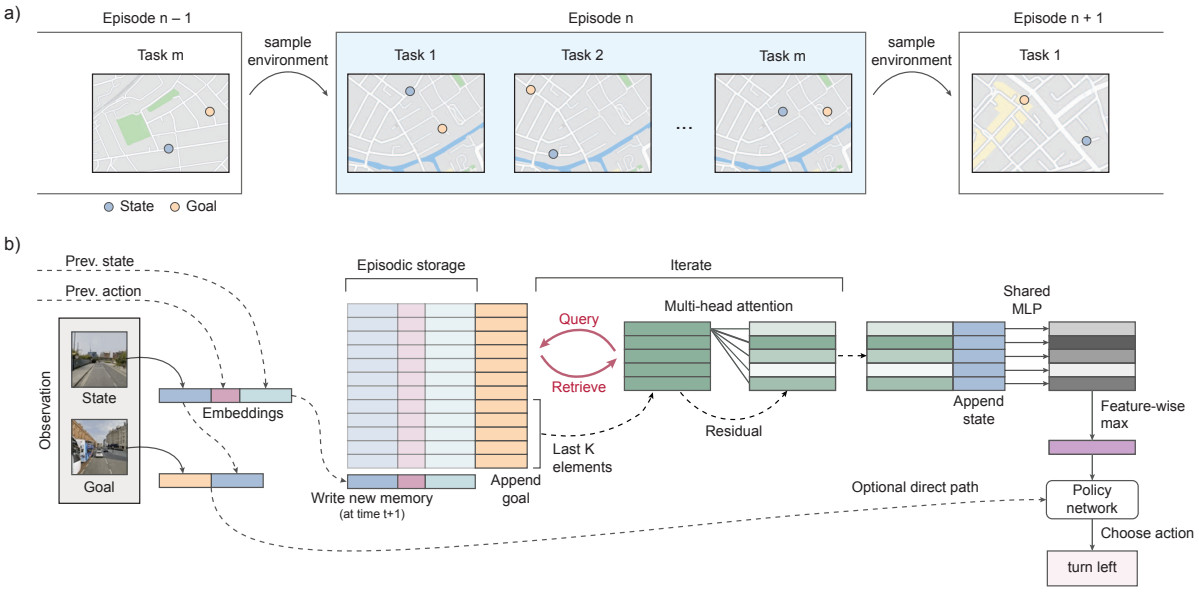


Figure 3.6: **Episodic memory** – reproduced from Ritter et al. (2021)

3.2.2 *Episodic memory*

A bit more structured than a recurrent memory, we can mention episodic memory as used by Ritter et al. (2021). As shown in Figure 3.6, each new encoded observation is appended to an episodic storage, allowing to retrieve information after querying the memory. Such retrieval is processed using a multi-head attention mechanism. In a very similar way, Fang et al. (2019) was already building a *Scene Memory* by appending observations when available and was using a Transformer encoder to predict the action to take. Using Transformers to process temporal information is also done in RL with approaches such as the *Decision Transformer* (Chen et al., 2021a) that is fed with a sequence of tuples (A_t, S_t, R_t), where A_t is the action taken at time t leading to the state S_t and reward R_t, and is trained to predict the next action to take. Other recent works have also leveraged Transformer models to learn to act (Du et al., 2021; Chen et al., 2021b, 2022b; Reed et al., 2022; Mezghani et al., 2023a).

3.2.3 *Topological memory*

Even more structured, neural agents can be equipped with a topological representation of the environment (Savinov et al., 2018a; Chaplot et al., 2020d; Beeching et al., 2020c; Wiyatno et al., 2022; Kim et al., 2023). This representation seems particularly suited to the *ImageNav* task as most recent work in Embodied AI using topological scene representations targets this task.

An example is the work done by Chaplot et al. (2020d) (Figure 3.7) in which the goal to find is presented as a panoramic image, showing the view of the surrounding scene the agent should have when reaching it. As such a task requires an agent to explore a scene, they build a topological memory to keep track of the information seen by the policy. At each timestep in an episode, the graph representation is updated, before being fed to a *Global*

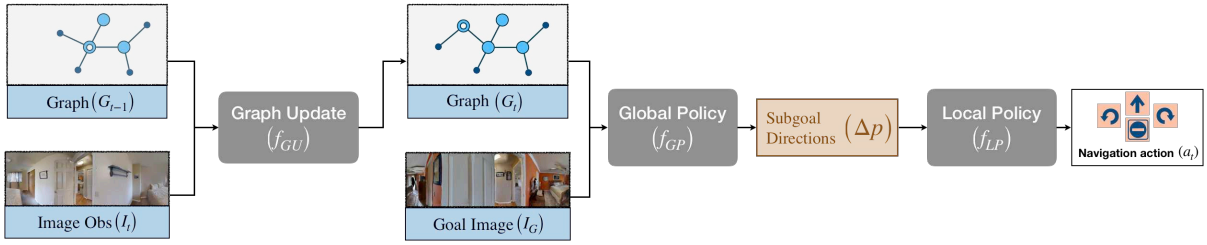
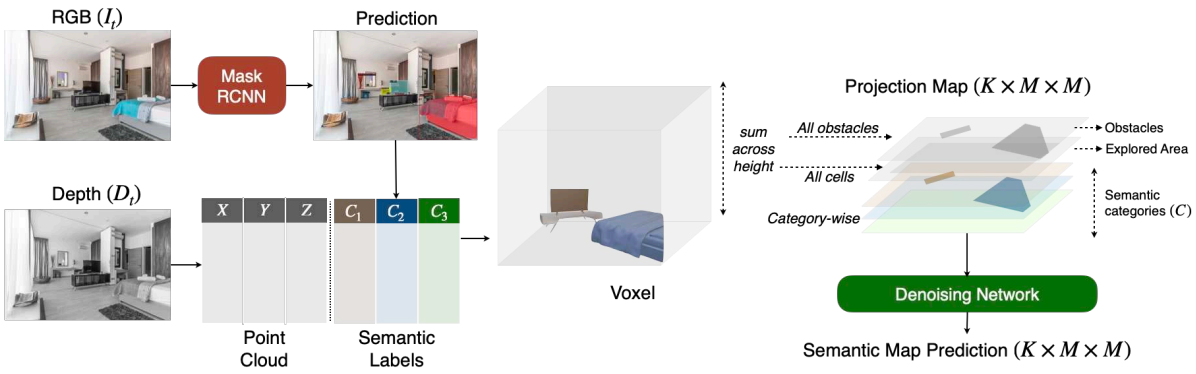


Figure 3.7: **Topological memory** – reproduced from Chaplot et al. (2020d)

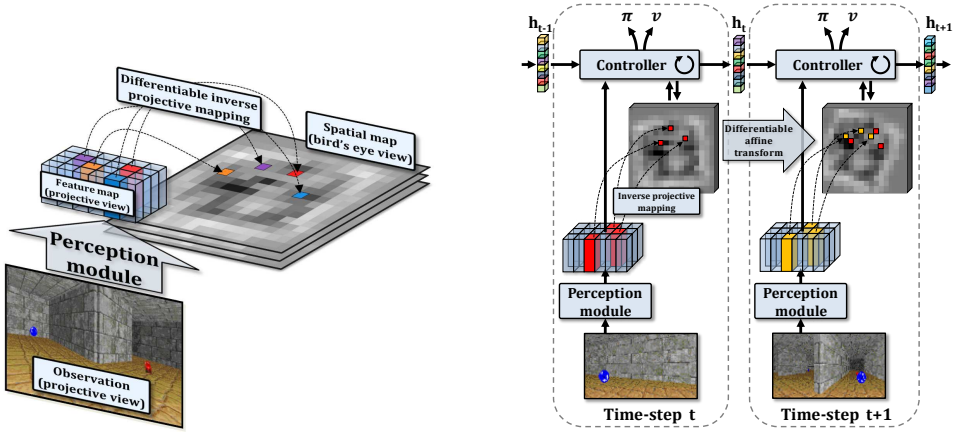
Policy predicting a sub-goal to be reached by a *Local Policy*. In the maintained graph, each node represents an area and an edge between two nodes indicates both areas are close and one can navigate from one to the other. There are two types of nodes: *Regular nodes* corresponding to areas that have already been explored since the beginning of the episode, and *ghost nodes* denoting explorable areas that have not yet been visited. Each regular node is associated with a panoramic image taken by the agent when being in the corresponding area. When visiting a new area, a regular node is added to the graph and directions leading to explorable areas are also predicted to add corresponding ghost nodes. Once the graph has been updated, a ghost node to be explored must be selected based on the specified goal image from the *ImageNav* task. In order to perform the graph updates and selections of the next ghost node to visit, four functions are proposed: (i) *graph localization* ($\mathcal{F}_L$), (ii) geometric explorable area prediction ($\mathcal{F}_G$), (iii) semantic score prediction ($\mathcal{F}_S$), (iv) relative pose prediction ($\mathcal{F}_R$). All these functions are implemented with a shared ResNet-18 encoder followed by a specific neural architecture for each function. $\mathcal{F}_L$ classifies a pair composed of an image I_s and the image I_{g_i} associated with the graph node i as whether I_s belongs to node i. $\mathcal{F}_G$ predicts the visible explorable area in an image I_s by dividing the latter into 12 vertical sections (corresponding to 12 angle bins as we have a panoramic image). From a source image I_s and a goal image I_g, $\mathcal{F}_S$ predicts the most likely direction in which the agent should move if observing I_s to reach I_g. Finally, $\mathcal{F}_R$ predicts the relative pose between two images I_s and I_g.

More recent work (Kim et al., 2023) also targets the *ImageNav* task with a method featuring a topological memory. However, in addition to representing each new area with a node in a graph (image graph) as in Chaplot et al. (2020d), they also maintain a second topological memory populated with object nodes (object graph) corresponding to specific objects detected by a Mask R-CNN model (He et al., 2017). They build a scene context by integrating information from the image and object graphs, using a cross-graph message passing approach inspired by MPNN (Gilmer et al., 2017). At navigation time, they select the most promising node to visit, i.e. the one that should be the closest to the goal to find, with an attention mechanism.

Beeching et al. (2020c) have also studied the use of topological memories in neural agents to address the *ImageNav* task. However, an important contribution and difference compared with prior such as Chaplot et al. (2020d) is their use of a Graph Neural Network (GNN) to learn to plan a path from a start to a target location from an *uncertain graph* whose nodes



Explicit map features (obstacles, explored area, semantic objects)



Implicit map features (extracted by a CNN model)

Figure 3.8: **Explicit** (reproduced from Chaplot et al. (2020a)) **vs implicit** (reproduced from Beeching et al. (2020b)) **map features**

are associated with visual features extracted by a CNN and node connectivity is estimated by a neural network as whether one node location is visible from another one. However, the important point is that such a graph is uncertain and can thus contain errors, either due to noise in the sensed data or from mistakes by the connectivity estimation module. Beeching et al. (2020c) thus train a GNN high-level planner to predict the best next node to visit in a topological representation of the explored area of a scene to come closer to a target specified as an image. Training is performed on a large set of pre-computed graphs ($72M$) created from exploration policy rollouts (Chen et al., 2018). They compare their high-level planner with optimal planners such as Dijkstra's algorithm and show their method better handles graphs including errors. At navigation time, the neural graph planner is coupled with a local policy trained to perform *PointNav* and reach high-level locations fed by the high-level planner, achieving strong *ImageNav* performance.

3.2.4 *Metric map memory*

Finally, many works have been equipping neural agents with grid-based representations (Henriques and Vedaldi, 2018; Parisotto and Salakhutdinov, 2018; Beeching et al., 2020b; Chaplot

et al., 2020b,a; Wani et al., 2020; Ramakrishnan et al., 2022; Sadek et al., 2023; Zhai and Wang, 2023; Chang et al., 2023), providing a more detailed picture of the scene. Navigating efficiently requires extracting features about the surrounding world that will be projected on a grid-based representation. We will distinguish two types of map features presented in Figure 3.8: *explicit* and *implicit* features.

Approaches populating their map with explicit features will select the ones that will be important to solve the task of interest (top part of Figure 3.8). In the case of Chaplot et al. (2020a) targeting the *ObjNav* task, each map cell is associated with channels indicating whether it contains an obstacle, and even more specifically one of the semantic objects among a fixed vocabulary of concepts, and whether it has already been visited. In this work, the agent receives an RGB and a depth frames with the same resolution at each timestep. The RGB observation is fed to a Mask R-CNN (He et al., 2017) model to segment semantic objects. The depth map is then used to create a 3D point cloud with a point per depth pixel. In addition to its 3D location, each point is associated with a semantic label coming from the prediction of Mask R-CNN for the corresponding RGB pixel. The point cloud is converted into a 3D voxel grid that will be projected to a 2D grid using a sum operation across the height. Such a map can then be used to find a next waypoint to navigate to. Chaplot et al. (2020a) do it by feeding it to a neural-based global policy trained with RL to output global goals to follow to explore the scene. More recent work (Ramakrishnan et al., 2022; Zhai and Wang, 2023; Chang et al., 2023) follows this direction to iteratively build a 2D map of explicit features while navigating, storing obstacles, visited cells, and important semantic landmarks.

Another direction is about storing implicit features of the observed environment (bottom part of Figure 3.8). These features are often extracted from a neural network such as done by Beeching et al. (2020b) projecting cells from feature maps into a 2D map using depth information. The interest of this approach is to learn features based on the task to solve without the need to design them explicitly. Indeed, the projection of 2D feature maps to 3D space is differentiable, allowing to train the perception module to extract useful features for the agent to navigate. Beeching et al. (2020b) for example show that features related to landmark objects to find are written on the map. Wani et al. (2020) also follow this direction of mapping a new environment with neural implicit features by proposing a baseline named *ProjNeuralMap* that we will present in more detail in chapter 4 along with other baseline methods.

There is thus a balance to find: implicit features leave more representation power to the neural modules to learn what are the important characteristics of perceived signals, while explicit features leverage our prior knowledge about important information to solve a task. Explicit features will then allow faster training but can be limited if more representational flexibility is required.

Context

In this manuscript, we have considered both mapping approaches. Chapters 4 and 5 will deal with implicit map features and chapter 6 will involve a map with explicit features.

3.3 Training neural agents

Independently of the method used to encode past experience in an environment, approaches can be trained differently to navigate. Formalisms to train navigation agents range from Deep RL (DRL) (Mirowski et al., 2017; Jaderberg et al., 2017; Zhu et al., 2017a; Wijmans et al., 2019; Chaplot et al., 2020b,d; Wijmans et al., 2022) to Imitation Learning (Ding et al., 2019; Ramrakhya et al., 2022), or even supervised learning of global goals to reach (Ramakrishnan et al., 2022; Zhai and Wang, 2023). This will not be covered in this manuscript but some approaches also leverage self-supervised learning to train goal-conditioned policies (Mezghani et al., 2023b).

3.3.1 *Reinforcement Learning*

Training the full agent – Previous work (Wijmans et al., 2019) has already shown we could train an agent end-to-end with RL only, more specifically with the PPO algorithm, to solve *PointNav* but, as already mentioned, required a large amount of experience data. Even if this method represents its environment with a simple recurrent memory, neural agents can still be trained end-to-end with RL when they involve more developed mapping strategies: this is the case of Beeching et al. (2020b) or Wani et al. (2020) that train map-based agents with RL, including the perception modules. Beeching et al. (2020b) and Wani et al. (2020) respectively leverage the A2C (Mnih et al., 2016) and PPO (Schulman et al., 2017) algorithms.

Such an end-to-end learning scheme is appealing because of its simplicity and flexibility but can have some drawbacks, such as sample inefficiency or lack of generalization outside the training domain. Reinforcement Learning indeed provides a sparse supervision signal which can lead to slow training, and not enough diversity in training data might lead to the policy overfitting to the training scenarios.

Training a specific function – A way to make RL training more efficient and the policy potentially more robust to distribution changes is to predict only high-level goals from domain-agnostic inputs and thus to relegate local navigation and state representation to other modules, that will either be neural-based or not. Modular approaches have indeed been introduced by Chaplot et al. (2020b) to learn to explore scenes and then adapted to the *ObjNav* task (Chaplot et al., 2020a). In both works, a grid-based map representation is estimated from sensory inputs (with semantic channels in Chaplot et al. (2020a) provided by a generic Mask R-CNN model as already presented before) and fed to a global policy that will output a global goal

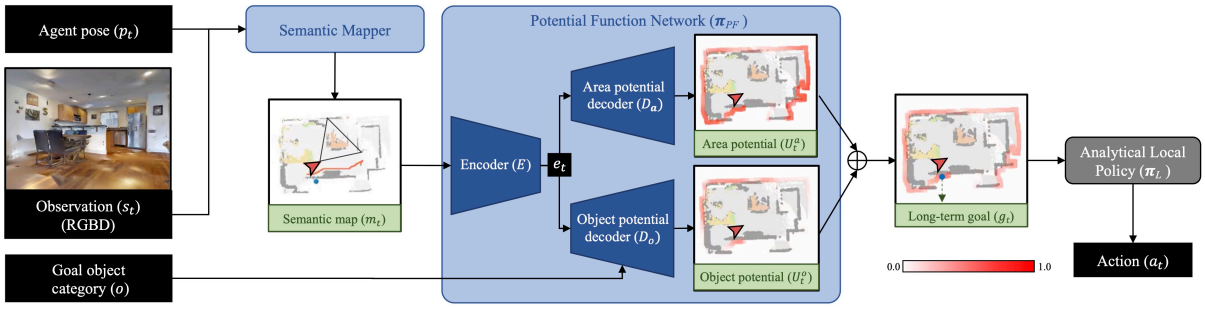


Figure 3.9: **PONI** – reproduced from Ramakrishnan et al. (2022)

to be reached by a local policy. The local policy can be trained with RL, but it was shown not to be better than analytical planning. Thus, the local policy is implemented by Chaplot et al. (2020a) with the FMM algorithm. Only the global policy is then trained with RL from a map input that is agnostic to visual conditions in the scene, leading to faster and more robust training. Gervet et al. (2022) showed this modular approach was also effective when deployed on real robots performing the *ObjNav* task in various real-world environments.

> **Context**
>
> Chapter 6 in this manuscript is about modular policies inspired by the ones presented here, with a global policy trained with RL to predict global goal locations.

While RL is a gold standard in Embodied AI, it has drawbacks as presented earlier. A solution could be to replace it with another training method such as imitation learning or supervised learning as we will present in the next subsection.

3.3.2 *Imitation learning and supervised global waypoint prediction*

Faster training can be achieved with supervised learning in two different ways: (i) imitation learning from expert trajectories, (ii) turning the prediction of global waypoints to follow as a supervised problem.

Imitation Learning – Ramrakhya et al. (2022) showed it was possible to train a policy with imitation learning on the *ObjNav* task from a set of human-generated demonstrations. To this end, they collected 70k demonstrations on training scenes and trained a simple recurrent agent to mimic human actions with Behavior Cloning, outperforming RL-based counterparts. Imitation learning is indeed a great strategy to provide a richer supervision signal but requires collecting expert trajectories.

Global waypoint prediction – Another way to turn policy training into a supervised learning problem is to go back to modular policies where only the global policy is trained. While Chaplot et al. (2020b) and Chaplot et al. (2020a) train the latter with RL, other work (Ramakrishnan et al., 2022; Zhai and Wang, 2023) decided to train the global policy to predict global goals in a supervised fashion. PONI (Ramakrishnan et al., 2022), illustrated in Figure 3.9,

introduces a potential function network that plays the role of the global policy as presented in previous work. A potential function associates a value between 0 and 1 to each location at the frontier between unexplored and explored area on a given map based on how promising they are to visit. This work proposed to train an encoder to extract features from an input semantic map and two decoders predicting frontier potential values: an area potential decoder and an object potential decoder, relatively predicting how frontier cells are likely to lead to a large unexplored area and to the goal to find. Both predictions are aggregated to select a long-term goal followed by a local policy. Interestingly, the map encoder and the two decoders are not trained by interaction, but in a supervised fashion. Indeed, it is possible to pre-compute semantic maps from annotations in 3D scene datasets along with ground-truth potential values as we have access to the positions of objects and sizes of unexplored areas. Instead of predicting frontier potentials, the more recent PEANUT (Zhai and Wang, 2023) directly predicts an object probability map where each cell is filled with a probability to find the goal object. Despite a more simple design, they show they reach competitive performance. These works thus question the initial assumption on the importance of interacting to learn to navigate. Other work (Hahn et al., 2021) also studies the ability to learn to navigate without RL and simulation.

In order to make policy training more efficient, alternative solutions could be to guide the learning process either with a preliminary pre-training of a part of the policy (often the perception module), or to augment the supervision signal with auxiliary tasks. This is what we will review in the next two subsections.

3.3.3 *Pre-training for Embodied AI*

Pre-training is extensively used in many domains as a great way to learn general abilities with supervised or more often self-supervised methods as presented previously (chapter 2), that will then be fine-tuned and specialized to a domain or task of interest.

Pre-training for visual navigation – Visual navigation also follows this direction. Early work (Zhu et al., 2017a) showed that it was possible to train a simple method composed of a frozen ResNet-50 encoder pre-trained on Imagenet and a fully-connected policy to navigate towards a target specified as an image (similar to *ImageNav*). Only the fully-connected policy was thus trained with RL, more specifically with the A3C algorithm (Mnih et al., 2016). Many works have been using pre-training as part of their solution since then, but we can mention Hao et al. (2020) or Bono et al. (2023a) that explicitly study pre-training approaches as their contributions. Hao et al. (2020) propose a pre-trained vision-language model to align images and textual instructions in the context of *VLN*. Bono et al. (2023a) show that the perception module of a policy strongly benefits from a cross-view completion pre-training to solve the *ImageNav* task. They indeed leverage a large pre-trained vision model called CroCo (Weinzaepfel et al., 2022) that was pre-trained with a variant of MAE where the encoder is fed with

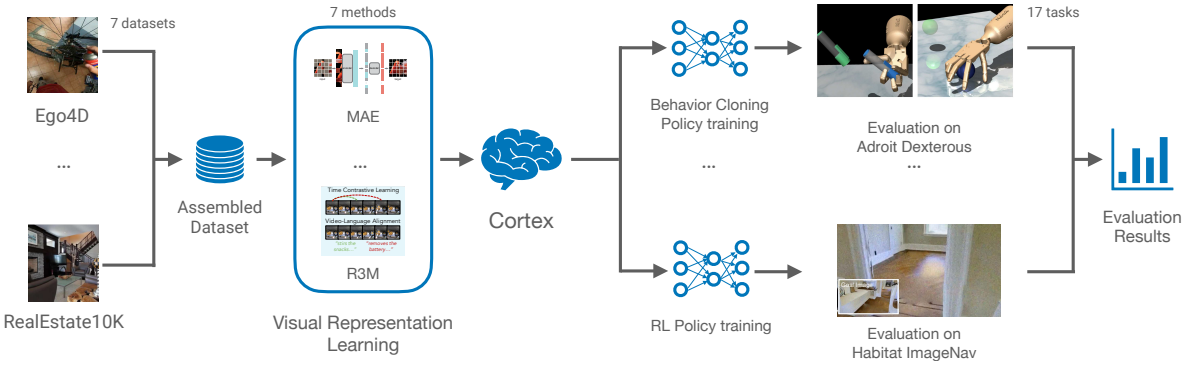


Figure 3.10: **Vision pre-training pipeline for robotics** – reproduced from Majumdar et al. (2023)

two views of a scene, where one of them is partially masked. The objective of the decoder is then to reconstruct pixels in the masked view. Such a pretext objective helps learning to extract robust geometric features, which are very useful in *ImageNav*. Bono et al. (2023a) go further than this by also finetuning their visual encoder on another pretext task about predicting the relative pose of two views, and whether a part of a view is visible from the other, i.e. if there is a content overlap between them. The encoder is finally frozen and only the policy along with a few visual adapters (we will present adapters in more detail later in this chapter) are trained with RL.

Pre-trained visual representations for multi-task learning – Backbone models pre-trained on large and diverse data have recently shown great promises in robotics when considering their generalization over different tasks (Majumdar et al., 2023; Parisi et al., 2022; Nair et al., 2023; Radosavovic et al., 2023; Ma et al., 2022; Khandelwal et al., 2022; Yadav et al., 2022, 2023a). Parisi et al. (2022) study visual pre-training methods for visuomotor control, showing the quality of self-supervised representations. Nair et al. (2023) introduce *R3M*, a general vision model pre-trained on egocentric video data to capture temporal dynamics and semantic features, improving downstream manipulation performance. Radosavovic et al. (2023) employ the MAE framework (He et al., 2022) to pre-train a single vision encoder (*MVP*) applied to robots in the real world. Ma et al. (2022) pre-train a visual model with a self-supervised value function objective on egocentric human videos to improve control policies. Finally, recent work (Khandelwal et al., 2022; Yadav et al., 2022, 2023a) has shown the great promise of pre-trained models, either *CLIP*-based (Khandelwal et al., 2022) or self-supervised (Yadav et al., 2022, 2023a), in visual navigation.

The work introducing *Cortexbench* (Majumdar et al., 2023) presented earlier explicitly studies the differences in visual representations from pre-trained backbones in the context of policy learning. More specifically, they both compare previous existing visual backbones and introduce a new one called *VC-1*. Figure 3.10 presents the general setup they follow: based on a visual dataset, a visual model will be pre-trained following one of various visual representation learning approaches. The given model will then be used as a frozen feature extractor to

generate observations fed to a neural policy for each task in *Cortexbench*. Depending on the task, the policy will have a specific architecture and tailored inputs, and will either be trained with BC or RL. Finally, task-specific policies will be evaluated to obtain a global score to assess the quality of the visual representations from the considered backbone model. Regarding existing pre-trained models, they show that no single one among *MVP* (Radosavovic et al., 2023), *CLIP* (Radford et al., 2021), *VIP* (Ma et al., 2022), or *R3M* (Nair et al., 2023) leads to consistent improvements on all tasks. They then introduce a new visual model called *VC-1*. It is a Vision Transformer (ViT) Dosovitskiy et al. (2020) model pre-trained from a set of 7 out-of-domain datasets with a focus on egocentric visual frames with a MAE representation learning objective. They show that data diversity helps improve the quality of produced visual features and reach strong performance on all considered tasks.

Transformer adapters – Transferring Transformer-based pre-trained models to new tasks or domains is an important topic. Methods involving adapter modules were introduced in NLP (Houlsby et al., 2019; Pfeiffer et al., 2021; Hu et al., 2021) to allow a fast and parameter-efficient transfer. Recent works employ the same methods in robotics (Sharma et al., 2022; Liang et al., 2022). Sharma et al. (2022) insert visual adapters in a Vision Transformer (ViT) pre-trained model. They introduce different types of adapter blocks ("bottom", "middle", "top") located at diverse places in the visual model and show that combining them improves performance. Liang et al. (2022) also show the positive impact of inserting task-specific adapters, trained with imitation learning, in a pre-trained Transformer-based model to adapt to robotics tasks.

> **Context**
>
> Chapter 7 will be dedicated to studying the adaptation of pre-trained visual features with a special conditioning on the task at hand. We will present adapters from Sharma et al. (2022) in more detail, and present our proposed improvement.

3.3.4 *Auxiliary supervision*

Instead of, or in addition to, pre-training, another approach is known as in-training auxiliary supervision. This is about augmenting the downstream objective to be optimized with additional training signals. It can be particularly interesting to speed up training and/or reach higher final performance. For this to be true, the additional supervision must be well-motivated to guide training towards a relevant solution. The training signal in RL tends to be sparser than with other training paradigms (supervised, dense self-supervised training), and thus can particularly benefit from denser auxiliary supervision.

Supervised auxiliary supervision – Mirowski et al. (2017) augment the vanilla reward-based RL training signal by learning to predict loop closure and reconstruct depth observations. In the same fashion, Lample and Chaplot (2017) also employ supervised auxiliary tasks, by aug-

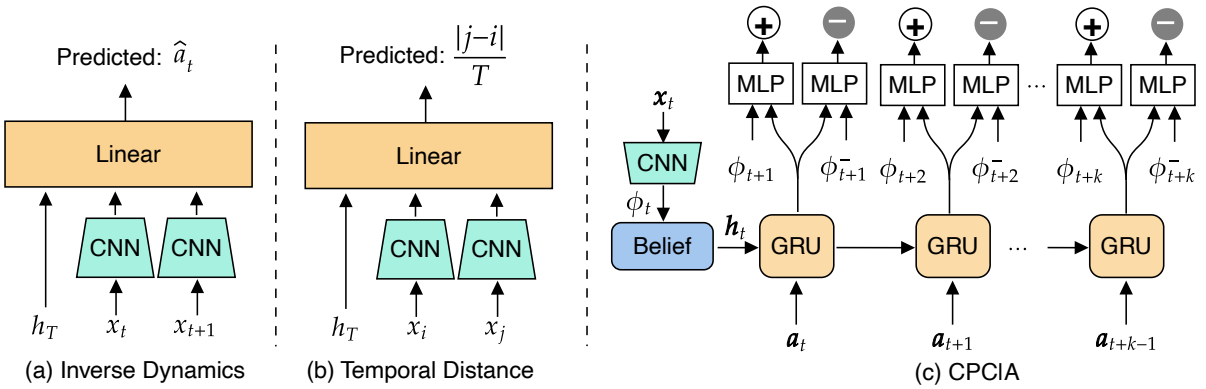


Figure 3.11: **Auxiliary supervision for the *PointNav* task** – reproduced from Ye et al. (2020)

menting the base Deep Recurrent Q-Network model (DRQN – Hausknecht and Stone (2015)) with fully-connected layers to predict game features in the context of learning to play a First-person shooter (FPS) game, improving significantly the downstream performance. A potential drawback of such supervised auxiliary supervision is the need for privileged information. However, RL training is generally performed in simulated environments (Kempka et al., 2016; Beattie et al., 2016), where basic information about the position of the goals for instance is simple to access as it is necessary to reward the agent. As a result, required annotations are obtained with very few supplementary work.

Unsupervised auxiliary supervision – Unlike these works, Jaderberg et al. (2017) propose unsupervised tasks such as pixel or feature-based control and reward prediction. In the same direction, Ye et al. (2020) (Figure 3.11) introduce self-supervised auxiliary tasks to speed up the training of agents in the *PointNav* task. They augment the base agent from Wijmans et al. (2019) with an inverse dynamics estimator as in Pathak et al. (2017), a temporal distance predictor that outputs the normalized number of steps between two observations, as well as an action-conditional contrastive module. In the latter, a dedicated GRU network takes a sequence of future actions as input and produces a hidden state that is concatenated to either a positive, i.e. the real observation that occurs after the given sequence of actions, or a negative sample, i.e. an observation sampled from other timesteps. The agent must predict if the sequence of actions will lead to the given observation, forcing to build long-horizon representations. In this work, authors also propose a new scheme to fuse multiple auxiliary tasks with several dedicated belief modules followed by a fusion module. Ye et al. (2021) introduce auxiliary tasks for the *ObjNav* task to mitigate overfitting and sample inefficiency of current agents. They build on top of Ye et al. (2020) and introduce the *Action Distribution Prediction* and *Generalized Inverse Dynamics* tasks, that both consist in predicting the actions that lead from one observation to another one, respectively with a 2-layer MLP and a GRU updated with actions taken. Finally, they have a *Coverage Prediction* task as, unlike with *PointNav*, exploration is important in *ObjNav*.

> **Context**
> In this manuscript, we also show the impact of auxiliary supervision when learning to navigate. More specifically, in chapter 4, we show that auxiliary tasks related to the mapping of semantic objects of interest improve the final navigation abilities towards goals.

3.3.5 *Multi-task robotic policies*

Having a single policy performing a wide range of tasks is a long-standing problem in robotics. With Deep Learning-based solutions becoming more popular, some prior work focuses on training multi-task neural agents. Approaches like BC-Z (Jang et al., 2022), RT-1 (Brohan et al., 2022), RT-2 (Zitkovich et al., 2023) or Gato (Reed et al., 2022) study the scaling abilities of neural models to large-scale datasets. Trained generalist agents show strong performance on a wide set of tasks, and can generalize to some extent to novel tasks. Another recent work is TD-MPC2 (Hansen et al., 2024), which introduces a model-based RL algorithm to learn general world models and studies task embeddings to condition a multi-task policy. Data-driven models can also be replaced by physics-informed models in model-based RL (Asri et al., 2024). Some work also learns to master multiple tasks by maximizing some intrinsic motivation objective (D'Eramo and Chalvatzaki, 2022). Finally, another direction is about learning multi-task policies from videos of humans interacting with the world (Soucek et al., 2024; Chane-Sane et al., 2023). This allows leveraging available large-scale datasets without requiring any task-specific labeling.

> **Context**
> More than only tackling visual features adaptation, chapter 7 will consider it in the context of multi-task policy learning.

3.3.6 *Neural agents with no task-specific training*

So far we have reviewed ways to train neural agents to solve one or several tasks of interest, while keeping in mind that some modular approaches will only train a specific global policy that will play the role of a single block in the final autonomous system. We have also considered pre-training approaches to leverage diverse external data in a first step. However, in all these cases, some training was happening with the final task in mind, either with a sparse RL training signal or more informative supervised learning methods. An interesting question is then the following: could the final neural agent be a combination of modules, some of them being non-neural and others being pre-trained general neural networks, that would just need to be assembled to solve a new task without performing any task-specific training?

The recent *GO to Any Thing* (*GOAT* – Chang et al. (2023)), illustrated in Figure 3.12, follows this path. It has the same modular form as previous work with a map-based representation,

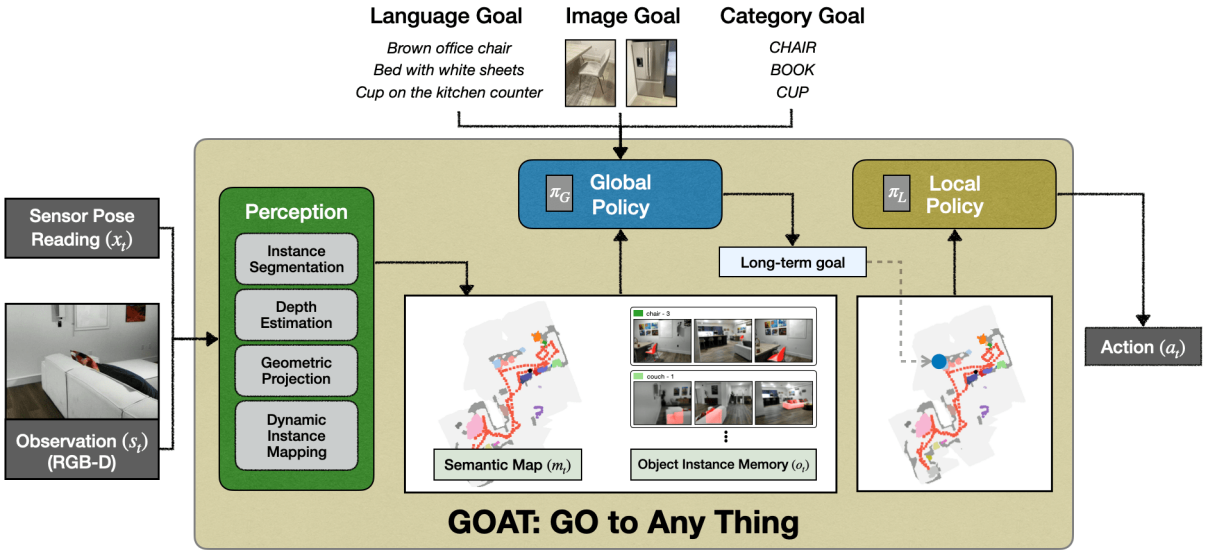


Figure 3.12: **GO to Any Thing** – reproduced from Chang et al. (2023)

a global policy and a local policy, but does not require any task-specific training. The method can reach goals specified by images (*ImageNav*), language (*VLN*) and object category (*ObjNav*), but the global policy is not trained with RL: when a new goal is specified, it will query (differently depending on the modality) a semantic memory built from the map, and will output an exploration goal using frontier-based exploration (Yamauchi, 1997) if the specified goal was not yet stored in memory. This work shows that neural-based modules are necessary to navigate as they are used to perceive, build a semantic map and embed goal queries, but we can reach satisfying performance on different tasks without a trained neural-based policy.

Another work (Krantz et al., 2023) focusing on *InstanceImageNav* also follows this direction. It is a modular method where exploration and local navigation are respectively performed by a Frontier-based exploration algorithm and the Fast Marching Method. The neural part is responsible for the identification of the target in the current observation. This is done by first extracting features with the neural-based SuperGlue method (Sarlin et al., 2020) in both the current observation and the target image. Features are then matched to check whether the goal is in sight. Once the goal has been identified, the agent can plan a path towards it with the Fast Marching method.

3.4 Neural scene representations

We have presented different approaches to encode past experience and, thus, represent 3D scenes while navigating in section 3.2. We have not yet mentioned a family of representations based on neural networks as we will dedicate this section to review the use of neural fields in robotics. These representations have originally been proposed in computer vision and computer graphics (see Chapter 2), in particular for novel view synthesis, but have been recently discovered for robotics, where they are starting to get used for representing scenes

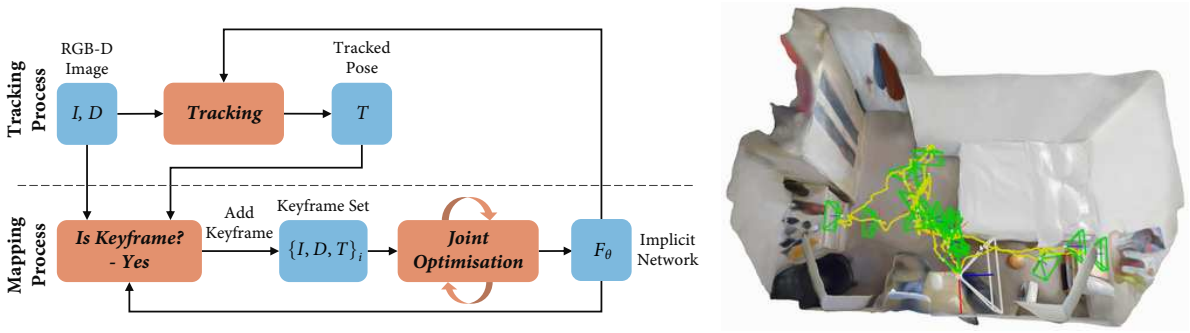


Figure 3.13: **iMap** – reproduced from Sucar et al. (2021)

during navigation or object manipulation. We dedicate a subsection to the use of NeRF-like representations in Embodied AI as it is a quite recent phenomenon that has been gaining interest, and that will play an important role in this manuscript (see chapters 5 and 6).

We can isolate three important research directions when it comes to using neural fields in robotics:

- NeRF-based SLAM: neural fields are updated in real-time to map a new 3D environment while localizing in the scene (3D online SLAM).
- NeRF for planning and control: this line of work studies the use of neural fields to perform planning or control, sometimes starting from a trained NeRF representing the scene of interest (the NeRF is not learned as in NeRF-based SLAM, but instead considered as known a priori).
- Active learning for neural fields: the goal here is to autonomously navigate a scene to collect NeRF training data to obtain the highest quality possible neural scene representation.

3.4.1 *NeRF-based SLAM*

Neural SLAM targets real-time SLAM with neural implicit representations. A first attempt at this was the iMap method (Sucar et al., 2021) illustrated in Figure 3.13. As presented by Sucar et al. (2021), a SLAM system must be *memory-efficient*, *data-efficient*, and *predictive*, i.e. able to reasonably predict the geometry of unobserved regions. An MLP-based representation thus appears as an interesting alternative as its continuous nature allows to automatically adapt the resolution (*memory-efficient*) to different parts of the scenes depending on the requirements (e.g. level of details of objects in a given region). Moreover, the interpolation abilities of neural networks can be useful in providing reasonable reconstructions of unobserved object parts (*predictive*). These advantages can only be leveraged if such models can be updated in real-time from a limited amount of data (*data-efficient*). The latter has been the main focus of the iMap paper, i.e. *how to efficiently train an MLP-based SLAM system from sparse incoming data?*

Unlike the original NeRF model, iMap does not require provided camera poses as localization is done in parallel with mapping. However, another difference is that iMap is provided with depth frames along with RGB images. Figure 3.13 shows the joint optimization of the camera poses and NeRF weights while new data is continually added when visiting a scene, with a supervision signal based on color and depth rendering. The implicit network is identical to the original NeRF implementation and the training process is very similar, with an additional depth rendering loss. We will rather focus on the challenges associated with solving the targeted problem: new data is continually arriving when the camera is moving and the network weights must be updated accordingly in real-time. This means it is important to focus on important new information to efficiently allocate the available computation, but also to overcome catastrophic forgetting problems, i.e. forgetting about parts of the scene that were seen a long time ago. In order to avoid the latter, they propose a replay-based approach where a replay buffer is populated with keyframes that will be continuously sampled to train the model. However, to keep computations efficient, not all keyframes are added: a frame is only selected if it shows a region of 3D space that is novel enough. A chosen proxy to measure novelty is the magnitude of the model training loss, i.e. keyframes associated with higher loss are likely to contain newer information that should be stored to be replayed later. Similarly, at optimization time, predicting color and depth for all pixels in all replay buffer frames is not computationally possible. Frames and a subset of their pixels are thus also actively sampled to allow the optimization process to run in real-time, combining uniform sampling and focus on regions where the neural network makes more mistakes.

Follow-up work adds semantic predictions to iMap (Zhi et al., 2021a) by augmenting the implicit model with a semantic head. An interesting specificity of this work is that scene-scale semantic labeling is learned from sparse semantic annotations of the scene, highlighting the interpolation abilities of NeRF-like models, and showing that they could realistically be deployed in a robotic system. Similarly, Zhi et al. (2021b) is also built on top of iMap and allows a user to interactively provide semantic annotation for the implicit representation to be trained in real-time.

Neural SLAM systems have then tried to overcome two limitations of iMap: the need to be provided with depth frames in real-time and the relatively small scale of scenes it can reconstruct robustly. NICE-SLAM (Zhu et al., 2022) propose a hierarchical grid-based representation to better handle the representation of large scenes: the scene is divided into 3 different voxel grids with different levels of detail (coarse, mid, fine). Pre-trained MLP decoders (one for each level of detail) fed with voxel locations and scene-specific features are used to render RGB and depth. Pre-training of the decoders is done following the setup from previous work (Peng et al., 2020), and only scene-specific features are optimized at NeRF training time. This leads to faster optimization but also the ability for decoders to encode resolution-specific priors. Follow-up work called NICER-SLAM (Zhu et al., 2024) builds on

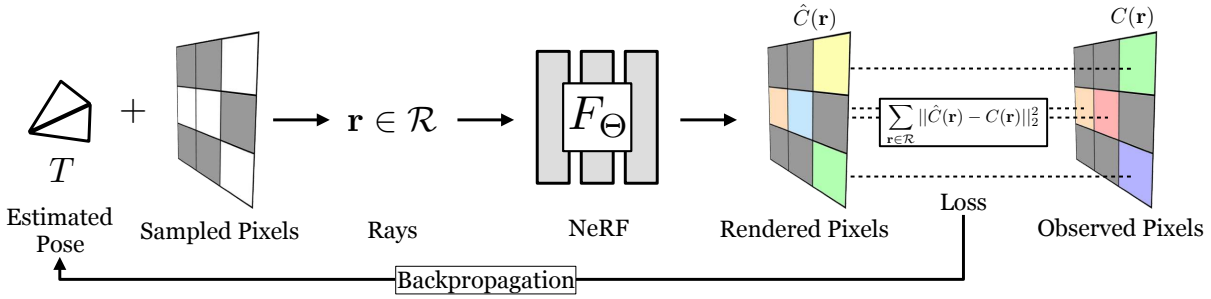


Figure 3.14: **iNeRF** – reproduced from Yen-Chen et al. (2021)

top of NICE-SLAM but removes the need for depth frame inputs. They achieve real-time RGB SLAM with NeRF models by augmenting the vanilla supervision signal with additional loss terms favoring the quality of the geometric representation. They still have a depth rendering loss term, but instead of requiring ground-truth depth, they use an off-the-shelf monocular depth predictor. They also predict normal maps with the same approach, add a warping loss, eikonal loss, and an optical flow loss. More recent work (Chung et al., 2023) combines NeRF and ORB-SLAM2 (Mur-Artal and Tardós, 2017) to provide a pre-training-free RGB SLAM method reaching better reconstruction performance than iMap and NICE-SLAM while being faster to optimize.

Similarly, more recent works start to tackle SLAM with Gaussian Splatting scene reconstruction (Yan et al., 2024; Keetha et al., 2024; Matsuki et al., 2024; Huang et al., 2024). The main advantage of Gaussian Splatting over NeRF is the higher training and rendering speeds, which are particularly important when performing real-time SLAM.

3.4.2 *NeRF for planning and control*

Instead of focusing on how to optimize NeRF-like models to map scenes, other works have studied how one could use an already-optimized NeRF model of a scene. A first interesting application of a trained NeRF model is to perform camera pose refinement, as in iNeRF (Yen-Chen et al., 2021) illustrated in Figure 3.14. The setup is as follows: we are provided with a noisy camera pose along with an image from this camera, and the goal will be to refine the pose. The idea in iNeRF is rather simple yet effective: we can use the same optimization strategy as when training a NeRF model, but instead of updating weight values, we will freeze them and only optimize the input camera pose to minimize the difference between the rendered frame and the camera frame. This is done with SGD directly in pixel space as when training a NeRF. Follow-up has looked at how to use a NeRF representation of a scene to perform visuomotor control. Adamkiewicz et al. (2022) propose to use NeRF density prediction to control a drone to avoid obstacles. Everything from state estimation to trajectory planning is based on the NeRF predictions. Adamkiewicz et al. (2022) improve upon iNeRF to estimate the state of the drone from visual inputs: at each timestep, the minimized loss to estimate the current pose is not only composed of the NeRF photometric loss as in iNeRF but

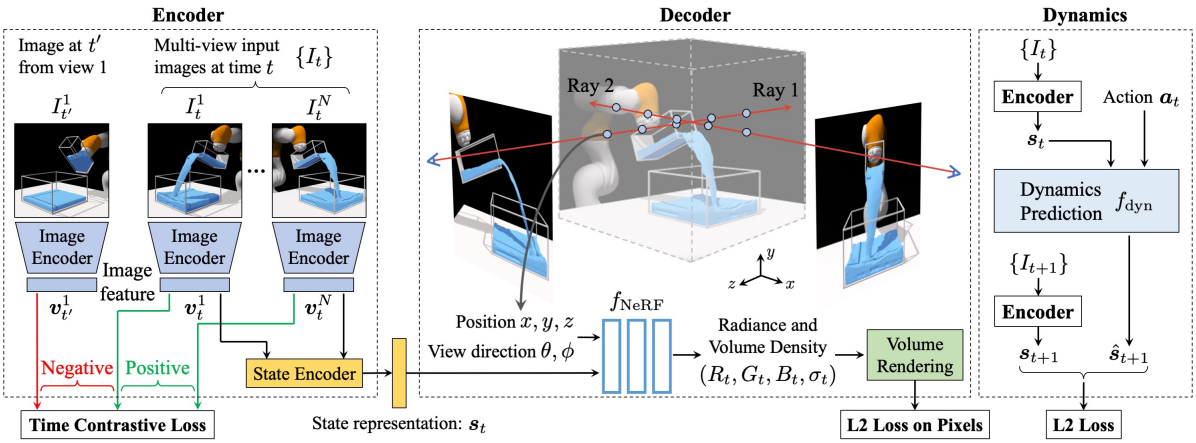


Figure 3.15: **Learning 3D-aware scene representations with NeRF guidance** – reproduced from Li et al. (2022b)

also an additional loss term minimizing the difference between the estimated state and the expected state according to the previously estimated one and a dynamics model of the drone given the previous taken action. This allows a more robust estimation, particularly when the drone passes through areas where the photometric loss brings little information (e.g. poor texture or no feature). Regarding planning a path while avoiding obstacles, they consider the probability of a light ray to be stopped at a given location, i.e. estimated by the NeRF density, to be a good approximation of the probability of collision at this location. Provided an estimated state, a path is thus updated to avoid high-density NeRF regions.

Li et al. (2022b) also propose to use a NeRF in the context of state estimation for visuomotor control, more specifically robotic manipulation. Figure 3.15 shows an overview of their approach. Compared with Adamkiewicz et al. (2022), the NeRF model is not used at state estimation time, but is used to supervise the training of an image encoder to extract a global 3D-aware representation of a full scene. More specifically, their introduced setup can be viewed as a form of auto-encoding where the NeRF model acts as a decoder only used at training time (discarded at inference time). The original NeRF model considers the scene to be static, while the scene is often dynamic when interacting with it. As a result, they introduce an additional input to the NeRF model which is a state vector encoding information about the 3D scene at a given timestep. At training time, the image encoder extracts features from different views of the same scene. Multi-view features are aggregated to form a scene state that is fed to the NeRF model which itself acts as a decoder trained to render back the different views from their associated camera poses. This auto-encoding approach guides the image encoder to extract information-rich features describing the whole 3D scene. The training signal is also augmented with a time contrastive loss forcing the image encoder to extract similar embeddings for views from the same timesteps and different representations for views from different timesteps. Once trained, the encoder can be used to extract the state of the scene at each timestep, from which a dynamics prediction model will be trained to predict next states

conditioned on actions. The encoder and dynamics prediction model can finally be used to perform visuomotor control.

Finally, Kwon et al. (2023) introduce a method with a similar spirit as it will use a neural field to guide the training of an image encoder, but this time, in the context of visual navigation, more specifically *ImageNav*. Reminiscent of 2D maps with implicit features presented earlier (Beeching et al., 2020b; Wani et al., 2020), they build a map, called *RNR-Map*, with neural features registered at each cell. The contribution in this work is about how the encoder is trained to extract map features: from a given image, features are extracted and projected to a 2D map. The latent 2D map is fed to a decoder based on generative scene networks (GSN – DeVries et al. (2021)) trained to reconstruct the input image. Enforcing the map to store the right features at different locations to render the 3D scene is a strong prior to solve *ImageNav* as the goal to reach is specified as an RGB image. The encoder is trained by sampling images in diverse 3D scenes with this auto-encoding setup. Once trained, the decoders are discarded, and only the encoder will be used to build latent maps while navigating, used to explore scenes, and identify goal locations to reach specified as RGB frames.

> **Context**
>
> Chapter 5 will present neural implicit representations keeping track of semantic objects, explored area, obstacles, and free space, and trained in real-time while navigating a scene. More importantly, we study how to train an RL-based policy to use such representations to navigate more efficiently.

3.4.3 *Active learning for neural fields*

Neural field active learning studies the selection of frames to efficiently train neural fields, i.e. reach the highest representation quality with the fewest number of samples. Unlike in NeRF-based SLAM where the collection of training frames is often externalized (e.g. done by a human holding a camera or in simulation by a given navigation method) and the focus is only made on the localization and mapping conditioned on the collected data, the story is different here: what we care about is the efficient data collection itself to optimize the quality of the underlying representation.

Active Learning for NeRF originally tackles the active selection of training data in fixed datasets of 2D frames: ActiveNeRF (Pan et al., 2022) estimates NeRF uncertainty by expressing radiance values as Gaussian distributions, and ActiveRMAP (Zhan et al., 2022) maximizes an entropy-based information gain metric.

More recent work (Zeng et al., 2023; Yan et al., 2023) studies robot navigation to actively collect data to train a 3D neural representation. Such approaches are very related to the task of *scene exploration* introduced earlier along with other Embodied AI tasks. A well-known baseline in this space is Frontier Based Exploration (FBE) (Yamauchi, 1997). Different variants

exist (Dornhege and Kleiner, 2013; Xu et al., 2017), but the core principle is to maintain a frontier between explored and unexplored space and to sample points on it. Learning-based approaches have also been proposed in recent work (Chaplot et al., 2020b; Chen et al., 2018; Ramakrishnan et al., 2021b). Similarly, active exploration of environment dynamics has also been studied in robotic manipulation (Schneider et al., 2022).

Going back to active exploration to collect neural field training data, Zeng et al. (2023) focus on improving planning efficiency. They introduce an implicit representation of the information gain brought by viewpoints in the form of a trainable MLP that will be queried to select parts of the environment to explore in priority. Unlike previous work (Ran et al., 2023) that was querying the scene representation itself to compute viewpoint information gain, the use of a specific information gain network is presented as more efficient. Indeed, querying the scene implicit representation itself to estimate information gain requires shooting a significant amount of rays through the 3D space. While still using a neural field to perform a fine-grained reconstruction of the scene, they combine the latter with a coarse explicit Truncated Signed Distance Field (TSDF) volumetric representation to allow fast collision checking when planning a path.

Yan et al. (2023) also study how to best explore a new scene to collect training data for a neural field, this time using the implicit representation itself to select waypoints. They specifically tackle the autonomous collection of data to learn an implicit neural signed distance field network provided depth inputs. To this end, they tend to favor regions of space where the neural field has the highest uncertainty. Their main contribution is a method to estimate such uncertainty: they show that unexplored regions will have a higher prediction variance when querying the neural field with the same input but randomly perturbing its weights. The 10% sampled 3D points with the highest prediction variance are selected and spatially clustered. The next waypoint to explore is then chosen by balancing maximum variance in the cluster, maximum number of cluster points, and minimal distance between the cluster and the current agent's location. Local navigation to the waypoint is carried out by a DD-PPO *PointNav* baseline (Wijmans et al., 2019).

> **Context**
>
> Chapter 6 studies how to autonomously explore a 3D scene to collect NeRF training data with a modular RL-based policy. We study data collection for RGB-only NeRF models on large realistic 3D scenes and present metrics to evaluate the quality of NeRF reconstructions in the context of robotics along with a use-case for our method.

Now that we have introduced necessary concepts and reviewed the recent progress in Embodied AI, we can begin with the next chapter studying the implicit learning of mapping abilities with auxiliary losses.

Chapter

Mapping Abilities Emerge in Multi-Object Navigation through Auxiliary Spatial Reasoning

Abstract

In the context of visual navigation, the capacity to map a novel environment is necessary for an agent to exploit its observation history in the considered place and efficiently reach known goals. This ability can be associated with spatial reasoning, where an agent is able to perceive spatial relationships and regularities, and discover object characteristics. Previous work introduces learnable policies parametrized by deep neural networks and trained with RL. In classical RL setups, the capacity to map and reason spatially is learned end-to-end, from reward alone. In this setting, we introduce supplementary supervision in the form of auxiliary tasks designed to favor the emergence of spatial perception capabilities in agents trained for a goal-reaching downstream objective. We show that learning to estimate metrics quantifying the spatial relationships between an agent at a given location and a goal to reach has a high positive impact in *Multi-Object Navigation*. Our method improves the performance of different baseline agents, that either build an explicit or implicit representation of the environment, even matching the performance of incomparable oracle agents taking ground-truth maps as input. A learning-based agent from the literature trained with the proposed auxiliary losses was the winning entry to the *Multi-Object Navigation Challenge*, part of the *CVPR 2021 Embodied AI Workshop*.

4.1 Context

Navigating in a previously unseen environment requires different abilities, among which is mapping, i.e. the capacity to build a representation of the environment. The agent can then reason on this map and act efficiently towards its goal. How biological species map their environment is still an open area of research (Peer et al., 2020; Warren et al., 2017). As presented in chapter 3, spatial representations in robotics have taken diverse forms, for instance metric (Elfes, 1989; Bresson et al., 2017) or topological maps (Shatkay and Kaelbling, 1997; Thrun, 1998). Most of these variants have lately been presented in neural counterparts, i.e. involving artificial neural networks — metric neural maps (Parisotto and Salakhutdinov, 2018; Beeching et al., 2020b; Henriques and Vedaldi, 2018) or neural topological maps (Beeching et al., 2020c; Chaplot et al., 2020d) learned from RL or with supervision.

This chapter focuses on improving the RL-based training strategy of autonomous agents parametrized by deep neural networks. We study whether the emergence of mapping and spatial reasoning capabilities can be favored by the use of spatial auxiliary tasks that are related to a downstream objective. We target the *Multi-ON* task (Wani et al., 2020), presented in chapter 3, where an agent must reach a sequence of specified objects in a particular order within a previously unknown environment. Such a task is interesting because it requires an agent to recall the position of previously encountered objects it will have to reach later in the sequence. The objective is not to introduce a new agent architecture, but rather to showcase the impact of augmenting the vanilla RL training of state-of-the-art (SOTA) agents selected in Wani et al. (2020) to solve the *Multi-ON* task. As already mentioned in chapter 3, augmenting the RL training of agents with auxiliary tasks has shown promise in many recent works introducing several variants (Mirowski et al., 2017; Jaderberg et al., 2017; Lample and Chaplot, 2017; Ye et al., 2021, 2020). The study performed in this chapter belongs to the group of supervised auxiliary tasks, with an application to 3D complex and photo-realistic environments, and specifically targets the learning of mapping and spatial reasoning, which has not been the scope of previous work.

We take inspiration from behavioral studies of human spatial navigation (Ekstrom et al., 2018). Experiments with human subjects aim at evaluating the spatial knowledge they acquire when navigating a given environment. Ekstrom et al. (2018) consider two important measures referred to as the *sense of direction* and *judgement of relative distance*. Regarding knowledge of direction, a well-known task is *scene- and orientation- dependent pointing* (SOP), where participants must point to a specified location that is not currently within their field of view. Being able to assess its relative position compared to other objects in the world is critical to navigate properly, and disorientation is considered a main issue. In addition to direction, evaluating the distance to landmarks is also of high importance.

We conjecture that an agent able to estimate the location of target objects relative to its

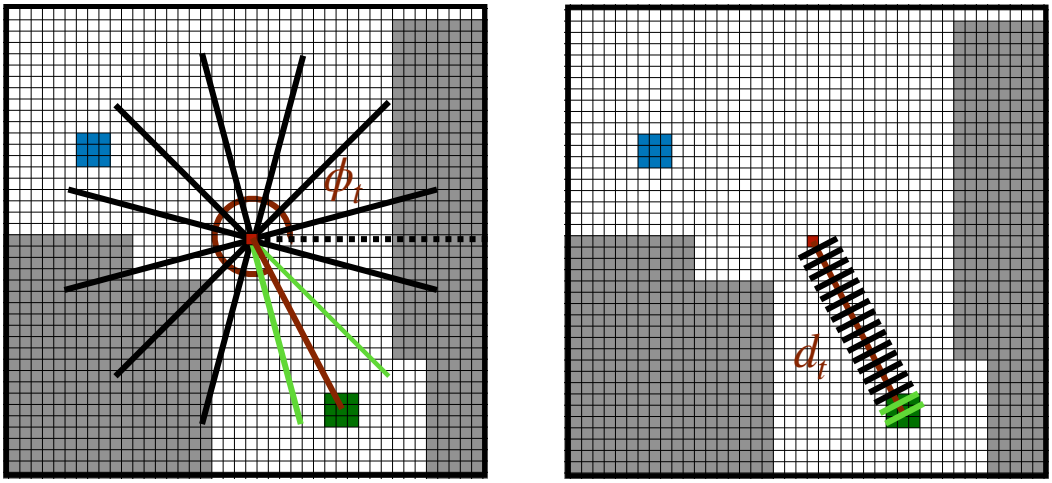


Figure 4.1: **Overview of the introduced auxiliary tasks**: In the context of DRL for *Multi-ON*, two auxiliary tasks predict the direction (*left*) and the distance (*right*) to the next object to retrieve *if it has been observed during the episode*. The **green object** (square) is the current target, and other targets exist and have already been found or might be required to be found later (ex: blue object / square). Red dot: position of the agent. White and gray cells respectively indicate free space and obstacles. Both the angle ϕ_t and the distance d_t between the center of the map (i.e. the agent) and the target at time t are discretized and associated with a class label. A third sub-task is to predict if the current target has already been within the agent's field of view during the episode.

current pose will implicitly extract more useful representations of the environment and navigate more efficiently. A fundamental skill for such an agent is thus to remember previously encountered objects. Our auxiliary supervision targets exactly this ability. Classical methods based on RL rely on the capacity of the learning algorithm to develop mapping strategies from reward alone. While this has been shown to be possible in principle (Beeching et al., 2020b), we will show that the emergence of a spatial mapping strategy is boosted through auxiliary tasks, which require the agent to continuously reason on the presence of targets with respect to its viewpoint — see Figure 4.1.

We introduce three auxiliary tasks, namely estimating if a target object has already been observed since the beginning of the episode, and if it is the case, the relative direction and the Euclidean distance to this object. If an object is visible in the current observation, it will be helpful for training the agent to recognize it (discover its existence and relevance to the task) and estimate its relative position. More importantly, if the target object was seen in the past, the auxiliary supervision will encourage the learning of representations of the environment, either implicitly or explicitly predicted by the agent, that are better spatially-structured and populated with more relevant semantic information, leading to an update of the neural memory of the agent.

The content of this chapter can be summarized as follows:

- We show that the auxiliary tasks improve the performance of previous neural baselines,

which even allows to reach the performance of (incomparable) agents using ground-truth oracle maps as input.

- We study the consistency of the gains over different inductive biases, i.e. different ways to structure neural networks, reaching from simple recurrent models to agents structured with projective geometry. This raises the question whether spatial inductive biases are required or whether spatial organization can be learned.
- The proposed method reaches SOTA performance on the *Multi-ON* task, and corresponds to the winning entry of the *CVPR 2021 Multi-ON challenge* [1] (*Test-Standard* leaderboard [2]).

4.2 Learning to map

We target the *Multi-ON* task (Wani et al., 2020), where an agent is required to reach a sequence of target objects, more precisely colored cylinders, in a certain order, and which was used for a recent challenge organized in the context of the CVPR 2021 Embodied AI Workshop. Compared to other tasks like *PointNav* or (Single) *ObjNav*, *Multi-ON* requires more difficult spatial reasoning capacities, in particular mapping the position of an object once it has been seen. The following capacities are necessary to ensure optimal performance: (i) mapping the object, i.e. storing it in a suitable latent memory representation; (ii) retrieving this location on request and using it for navigation and planning, including deciding when to retrieve this information, i.e. solving a correspondence problem between sub-goals and memory representation.

4.2.1 *SOTA agents in Multi-ON*

The proposed method is independent of the actual implementation choices in agents solving the *Multi-ON* task as we rather target an improvement of the learning objective. We therefore explored several neural baselines with different architectures, as selected in Wani et al. (2020). The considered agents share a common base shown in Figure 4.2, which extracts information from the current RGB-D observation with a convolutional neural network (CNN) c, and computes embeddings of the target object class and the previous action taken by the agent. Differences between the considered baselines is in their representation of the environment. The simplest recurrent baseline *NoMap* does not construct a map of its environment. *OracleMap* and *OracleEgoMap* baselines do not build a global map, but rather have access to oracle global maps of the environment containing channels for occupancy information and location of goal objects. Finally, *ProjNeuralMap* builds a map of the environment in real time, associating feature vectors from c with discrete cells in the spatial 2D representation using projective geometry. In variants that keep a global map, i.e. all except *NoMap*, it is first transformed into an egocentric representation centered around the agent's position (explained further below).

[1] http://multion-challenge.cs.sfu.ca/2021.html
[2] https://eval.ai/web/challenges/challenge-page/805/leaderboard/2202

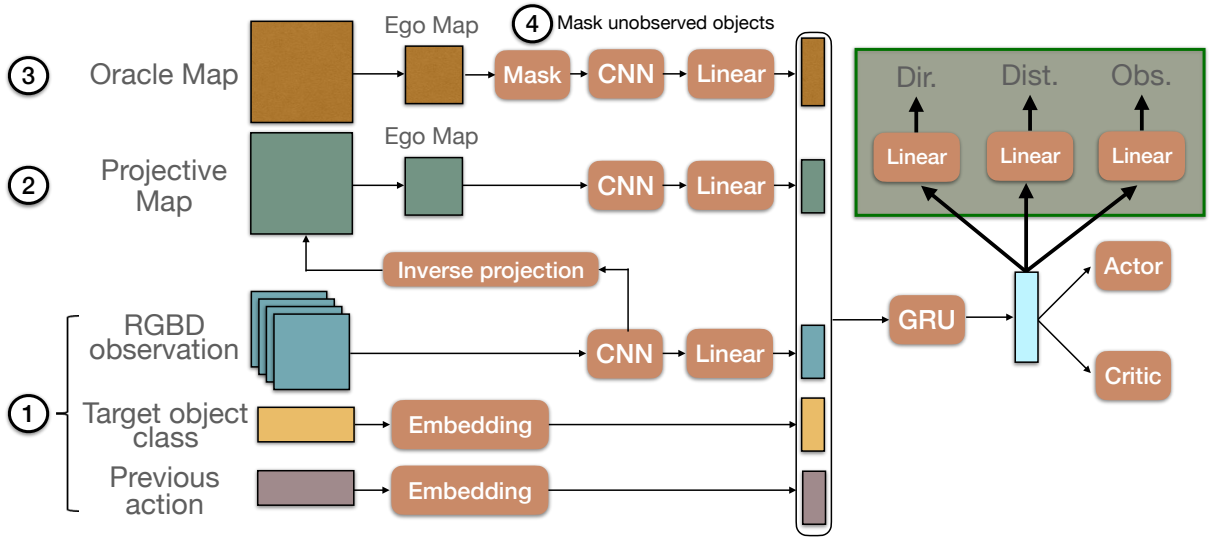


Figure 4.2: **Base *Multi-ON* agent architecture**: To study the impact of our auxiliary losses on different agents (Wani et al., 2020), we explore several input and inductive biases. All variants share basic observations ① (RGB-D image, target class, previous action). Variants also use a map ② produced with inverse projective mapping. Oracle variants receive ground truth maps ③, where in one further variant unseen objects are removed ④. These architectures have been augmented with classification heads implementing the proposed auxiliary tasks (**green rectangle**).

A vector representation of the map is then extracted using another CNN $\tilde{c}$. Such operation can be considered as a global read of the map. The vector representations are concatenated and fed to a GRU (Cho et al., 2014) unit that integrates temporal information, and whose output serves as input to an actor and a critic heads, that respectively output a distribution over the set of actions to take and an estimation of the value of the state the agent is currently in. All agents are trained with the same RL algorithm (and same training hyperparameter values) detailed in subsection 4.2.3, as well as the actor-critic formulation.

We present here in more details the considered variants which have been explored by Wani et al. (2020), but have been introduced in prior work (numbers ①②③④ correspond to choices in Figure 4.2):

NoMap ① – is a recurrent GRU baseline that does not explicitly build nor read a spatial map. The only memory available for storing mapping information is the flat vectorial hidden state of the GRU, a variant of a recurrent neural network. While the agent could in principle still learn (through RL) to use this vectorial memory like a spatial map, this is in no way enforced through any design choice.

ProjNeuralMap ①② **(Henriques and Vedaldi, 2018; Beeching et al., 2020b)** – is a neural network structured with spatial information and projective geometry. Or, stated in different terms, in this work the map is *not* pre-computed by a handcrafted and engineered function (e.g. with estimated occupancy) and fed to an agent, as done in classical robotics; rather,

the map is an internal activation of a neural network layer without trainable parameters. As such the content of the map is not predefined and interpretable through a handcrafted definition, the content is trained through machine learning, in our case RL. This layer is a map in the sense that (i) it is spatially organized and corresponds to an allocentric birds-eye representation, which is shifted and rotated with each agent motion through estimated odometry; (ii) using calibrated cameras, pixels are mapped to corresponding points on the map. However, the actual values stored at each position are determined through training and can, according to the learning signal, correspond to a latent representation of anything ranging from occupancy to more semantic information like object positions.

More specifically, ProjNeuralMap maintains a global allocentric map of the environment $M_t \in \mathbb{R}^{H \times W \times n}$ composed of n-channel vector representations, n being a hyperparameter, at each position within the full $H \times W$ environment. Similar to Bayesian occupancy grids (BOG), which have been used in mobile robotics for many years (Moravec, 1988b; Rummelhard et al., 2015), the map is updated in the two-step process already mentioned above: (1) resampling taking into account estimated agent motion, and (2) integration of the representation of the current observation produced by a CNN c.

Writing to the map — Given the current RGB observation $o_t \in \mathbb{R}^{h \times w \times 3}$, c extracts an n-channel feature map o'_t, which is then projected onto the 2D ground plane following the procedure in MapNet (Henriques and Vedaldi, 2018) to obtain an egocentric map of the agent's spatial neighborhood $m_t \in \mathbb{R}^{h' \times w' \times n}$. The ground projection module assigns a discrete location on the ground plane to each element within o'_t conditioned on the input depth map $d_t \in \mathbb{R}^{h \times w}$ and known camera intrinsics. Registration of the observation m_t to the global map is based on the assumption that the agent has access to odometry, as in Wani et al. (2020). The update to M_t is performed through an element-wise max-pooling between m_t and M_{t-1}.

Reading the map — The global map is first cropped around the agent and oriented towards its current heading to form an egocentric map of its neighborhood at time t, which is then fed to $\tilde{c}$ producing a context feature vector. The latter is then concatenated to the rest of the input, i.e. representations of the current observation, target object and previous action, producing the input to the recurrent memory (GRU) unit. The full model is trained end-to-end with RL, including networks involved in map writing and reading operations.

OracleMap ①③ – has access to a ground-truth grid map of the environment with 2 channels. The first channel is dedicated to occupancy information with a binary value per cell indicating the presence of free space or an obstacle. The second channel encodes the presence of objects and their classes with thus $n_{target} + 1$ possible values per cell, i.e. 1 to n_{target} for one of the n_{target} object classes, or 0 for no object. Each channel information is passed through a learned embedding layer to output a m-dim vector, m being a hyperparameter, as it is common practice to represent categorical data fed to a neural network. This leads to a map with $2 \times m$ channels.

Agent	GRU state	Map	Map update	Map reading	Full visibility	Oracle occupancy	Oracle goals	Neural features
NoMap	✓	–	–	–	–	–	–	–
OracleMap	✓	✓	–	✓	✓	✓	✓	–
OracleEgoMap	✓	✓	–	✓	–	–	✓	–
ProjNeuralMap	✓	✓	✓	✓	–	–	–	✓

Table 4.1: **Summary of environment representations in *Multi-ON* baseline agents**

The map is cropped and centered around the agent to produce an egocentric map as input to the model.

OracleEgoMap ①③④ – gets the same egocentric map as OracleMap with only object channels, and revealed in regions that have already been within its field of view during the episode. This variant corresponds to an agent capable of perfect mapping — no information gets lost, but only observed information is used.

Table 4.1 summarizes the environment representation strategies used by the different baselines.

4.2.2 *Learning to map objects with auxiliary tasks*

We introduce auxiliary tasks, additional to the classical RL objectives, and formulated as classification problems, which require the agent to predict information on object appearances, which were in its observation history in the current episode. To this end, the base model is augmented with three classification heads (Figure 4.2) taking as input the contextual representation produced by the GRU unit. It is important to note that these additional classifiers are only used at training time to encourage the learning of spatial reasoning. At inference time, i.e. when deploying the agent on new episodes and/or environments, predictions about already seen targets, their relative direction are distance are not considered. Only the output of the actor is taken into account to select actions to execute.

Direction – the agent predicts the relative direction of the target object, only if it has been within its field of view in the observation history of the episode (Figure 4.1 left). The ground-truth direction towards the goal is computed as,

$$\phi_t = \sphericalangle(\mathbf{o}_t, \mathbf{e}) = -\operatorname{atan2}(\mathbf{o}_{t,x} - \mathbf{e}_x, \mathbf{o}_{t,y} - \mathbf{e}_y) \tag{4-1}$$

where $\mathbf{e} = [\mathbf{e}_x \ \mathbf{e}_y]$ (“*ego*”) are the coordinates of the agent on the grid and $\mathbf{o} = [\mathbf{o}_{t,x} \ \mathbf{o}_{t,y}]$ are the coordinates of the center of the target object at time t. As the ground-truth grid is egocentric, the position of the agent is fixed, i.e. at the center of the grid, while the target object gets different coordinates with time. The angles are kept in the interval $[0, 2\pi]$ and then discretized into K bins, giving the angle class. The ground-truth one-hot vector is denoted

ϕ_t^*. At time instant t, the probability distribution over classes $\hat{\phi}_t$ is predicted from the GRU hidden state $\mathbf{h}_t$ through an MLP as $p(\hat{\phi}_t) = f_\phi(\mathbf{h_t}; \theta_\phi)$ with parameters θ_ϕ.

Distance – The second task requires the prediction of the Euclidean distance in the egocentric map between the center box, i.e. position of the agent, and the mean of the grid boxes containing the target object (Figure 4.1 right) that was observed during the episode, $d_t = ||\mathbf{o}_t - \mathbf{e}||_2$. Again, distances are discretized into L bins, with d_t^* as the ground-truth one-hot vector, and at time instant t, the probability distribution over classes $\hat{d}_t$ is predicted from the hidden state $\mathbf{h}_t$ through an MLP as $p(\hat{d}_t) = f_d(\mathbf{h_t}; \theta_d)$ with parameters θ_d.

Observed target – This third loss favors learning whether the agent has previously encountered the target object. The model is required to predict the binary value $\mathbb{1}_t^{\text{obs}}$, defined as 1 if the target object at time t has been within the agent's field of view at least once in the episode, and 0 otherwise. The model predicts the probability distribution over classes $\hat{obs}_t$ given the hidden GRU state $\mathbf{h}_t$ through an MLP as $p(\hat{obs}_t) = f_{obs}(\mathbf{h_t}; \theta_{obs})$ with parameters θ_{obs}.

4.2.3 *Training agents with Deep RL*

Following Wani et al. (2020), all agents are trained with PPO (Schulman et al., 2017) and a reward composed of three terms,

$$R_t = \mathbb{1}_t^{\text{reached}} \cdot R_{\text{goal}} + R_{\text{closer}} + R_{\text{time-penalty}} \tag{4-2}$$

where $\mathbb{1}_t^{\text{reached}}$ is the indicator function whose value is 1 if the *Found* action was called at time t while being close enough to the target, and 0 otherwise. R_{closer} is a reward shaping term equal to the decrease in geodesic distance to the next goal compared to previous timestep. Finally, $R_{\text{time-penalty}}$ is a negative slack reward to force the agent to take short paths.

The PPO loss denoted $\mathcal{L}_{PPO}$ can then be computed as presented in equation 2-29 from chapter 2.

4.2.4 *Modification of the training objective with auxiliary tasks*

We now present the additional terms to the base PPO loss introduced to encourage spatial reasoning in trained agents. Let us recall that, as stated in chapter 2, at sampling time k in the PPO algorithm, a set $\mathcal{U}_k$ of trajectories τ with length T are collected using the latest policy. $\mathcal{L}_{PPO}$ is computed from such data and our auxiliary supervision too. Direction, distance and observed target predictions are supervised with cross-entropy losses from ground truth values

ϕ_t^*, d_t^* and $\mathbb{1}_t^{\text{obs}}$, respectively, as

$$\mathcal{L}_\phi = \frac{1}{|\mathcal{U}_k|\,T} \sum_{\tau \in \mathcal{U}_k} \sum_{t=0}^{T-1} \left[-\mathbb{1}_t^{\text{obs}} \sum_{c=1}^{K} \phi_{t,c}^* \log p(\hat{\phi}_{t,c}) \right] \tag{4-3}$$

$$\mathcal{L}_d = \frac{1}{|\mathcal{U}_k|\,T} \sum_{\tau \in \mathcal{U}_k} \sum_{t=0}^{T-1} \left[-\mathbb{1}_t^{\text{obs}} \sum_{c=1}^{L} d_{t,c}^* \log p(\hat{d}_{t,c}) \right] \tag{4-4}$$

$$\mathcal{L}_{obs} = \frac{1}{|\mathcal{U}_k|\,T} \sum_{\tau \in \mathcal{U}_k} \sum_{t=0}^{T-1} -(\mathbb{1}_t^{\text{obs}} \log p(\hat{obs}_t) + (1 - \mathbb{1}_t^{\text{obs}}) \log(1 - p(\hat{obs}_t))) \tag{4-5}$$

where $\mathbb{1}_t^{\text{obs}}$ is the binary indicator function specifying whether the current target object has already been seen in the current episode ($\mathbb{1}_t^{\text{obs}}$=1), or not ($\mathbb{1}_t^{\text{obs}}$=0).

The auxiliary losses $\mathcal{L}_\phi$, $\mathcal{L}_d$ and $\mathcal{L}_{obs}$ are added as follows,

$$\mathcal{L}_{tot} = \mathcal{L}_{PPO} + \lambda_\phi \mathcal{L}_\phi + \lambda_d \mathcal{L}_d + \lambda_{obs} \mathcal{L}_{obs} \tag{4-6}$$

where λ_ϕ, λ_d and λ_{obs} weight the relative importance of auxiliary losses.

4.3 Experimental results

We focus on the *3-ON* version of the *Multi-ON* task, where the agent deals with sequences of 3 objects. The time limit is fixed to 2500 environment steps, and there are 8 object classes. The agent receives a $(256\times256\times4)$ RGB-D observation and the one-in-K encoded class of the current target object within the sequence. We recall the discrete action space introduced in chapter 3: *Move Forward 0.25m*, *Rotate Right* 30°, *Rotate Left* 30°, and *Found*, which signals that the agent considers the current target object to be reached. As the aim of the task is to focus on evaluating the importance of mapping, a perfect localization of the agent was assumed as in the protocol proposed by Wani et al. (2020).

Dataset and metrics – we used the standard train/val/test split over scenes from the Matterport3D dataset (Chang et al., 2018), ensuring no scene overlap between splits, as done by Wani et al. (2020). There are 61 training scenes, 11 validation scenes, and 18 test scenes. The train split consists of 50,000 episodes per scene, while there are 12,500 episodes per scene in the val and test splits. Reported results on the val and test sets (Tables 4.3 and 4.4) were computed on a subset of 1,000 randomly sampled episodes. Figure 4.3 shows an example of episode (from the Mini-val set of the *CVPR 2021 Multi-ON Challenge*) with RGB-D inputs.

We consider standard metrics of the field as given by Wani et al. (2020) and presented in chapter 3: *Success Rate*, *Progress Rate*, *SPL*, *PPL*. Note that for an object to be considered found, the agent must take the *Found* action while being within 1.5m of the current goal. The episode ends immediately if the agent calls *Found* in an incorrect location.

Agent	Dir.	Dist.	Obs.	Success	Progress	SPL	PPL	†
OracleMap	–	–	–	44.9± 1.7	55.7± 2.4	35.4± 1.4	43.7± 2.2	–
OracleEgoMap	–	–	–	27.5± 2.7	42.8± 2.8	21.3± 2.5	32.7± 2.9	–
ProjNeuralMap	–	–	–	21.8± 1.7	38.6± 1.3	15.4± 0.7	27.0± 0.7	✓
	–	–	✓	22.4± 2.9	40.2± 2.2	16.2± 2.7	28.9± 2.3	✓
	–	✓	–	27.3± 3.3	43.0± 3.6	19.2± 2.1	30.6± 2.4	✓
	✓	–	–	40.2± 4.2	55.9± 3.5	26.1± 2.2	36.4± 2.0	✓
	✓	✓	–	44.3± 6.6	58.9± 4.9	29.0± 3.7	39.0± 2.2	✓
	✓	✓	✓	**49.2 ± 7.1**	**62.8 ± 5.2**	**32.0 ± 2.7**	**41.1 ± 1.1**	✓

Table 4.3: **Impact of different auxiliary tasks** (validation performance) – The † column specifies comparable agents. Performance is reported as *mean* ± *std* over training runs (seeds).

Optimizer	Adam
Adam eps	1e-5
Learning rate	2.5e-4
Linear learning rate decay	✓
Number of epochs	2
Env. steps per update	128
Clipping ratio	0.2
Linear clip decay	✓
Value loss coefficient	0.5
Entropy coefficient	0.01
Max Grad Norm	0.2
GAE	✓
GAE-λ	0.95
Discount factor	0.99
Reward window size	50

Table 4.2: **PPO hyperparameters:** Values for PPO hyperparameters used when training agents (same in chapter 5).

Implementation details – training and evaluation hyperparameters, as well as architecture details have been taken from Wani et al. (2020). All reported quantitative results are obtained after 4 training runs (6 runs were computed for *ProjNeuralMap* with the three auxiliary losses for job scheduling reasons) for each model, during $70M$ steps (increased from $40M$ in Wani et al. (2020)). This amount of training time is standard when considering previous work targeting visual navigation with learning-based agents trained with RL. We report the average performance and standard deviation among training runs (random seeds) for each variant as *mean* ± *std*. Ground-truth direction and distance measures are respectively split into K=12 and L=36 classes. Indeed, angle bins span 30°, and distance bins span a unit distance on the egocentric map which is 50×50 (the maximum distance between center and a grid corner is thus 35). The map used to compute ground-truth labels for auxiliary losses is the one fed to the *OracleEgoMap* agent. Training weights λ_ϕ, λ_d and λ_{obs} are all fixed to 0.25. Each classification head is a single linear layer followed by a softmax activation function. Table 4.2 presents PPO training hyperparameters that will be identical in chapter 5.

Do the auxiliary tasks improve the downstream objective? – in Table 4.3, we study the impact of the different auxiliary tasks on the *3-ON* benchmark when added to the training objective of *ProjNeuralMap*, and their complementarity. Direction prediction significantly improves performance, adding distance prediction further increases all metrics, outperforming the performance of (incomparable) *OracleEgoMap*. Both losses have thus a strong impact and

Agent	Aux. Sup.	Success	Progress	SPL	PPL	†
OracleMap	–	50.4± 3.5	60.5± 3.1	40.7± 2.2	48.8± 1.9	–
OracleEgoMap	–	32.8± 5.2	47.7± 5.2	26.1± 4.5	37.6± 4.7	–
	✓	44.0± 7.1	55.1± 7.0	35.0± 5.2	43.8± 5.0	–
ProjNeuralMap	–	25.9± 1.1	43.4± 1.0	18.3± 0.6	30.9± 0.7	✓
	✓	**57.7 ± 3.7**	**70.2 ± 2.7**	**37.5 ± 2.0**	**45.9 ± 1.9**	✓
NoMap	–	16.7± 3.6	33.7± 3.3	13.1± 2.4	26.0± 1.7	✓
	✓	43.0± 4.7	58.2± 4.0	29.5± 1.8	39.9± 1.3	✓

Table 4.4: **Consistency over multiple models** (test set) – The † column specifies comparable agents. Performance is reported as *mean* ± *std* over training runs (seeds).

Agent/Method	— Test Challenge —				— Test Standard —			
	Success	Progress	SPL	PPL	Success	Progress	SPL	PPL
Ours (Aux. losses)	**55**	**67**	**35**	**44**	57	70	36	45
SGoLAM	52	64	32	38	62	71	34	39
VIMP	41	57	26	36	43	57	27	36
ProjNeuralMap*	–	–	–	–	12	29	6	16
NoMap*	–	–	–	–	5	19	3	13

Table 4.5: **CVPR 2021 *Multi-ON* Challenge Leaderboard** – *Test Challenge* are the official challenge results. *Test Standard* contains pre- and post-challenge results. Ranking is done with **PPL**. The * symbol denotes Challenge baselines.

are complementary, confirming the assumption that *sense of direction* and *judgement of relative distance* are two key skills for spatially navigating agents. The third loss about observed target objects brings a supplementary non-negligible boost in performance, showcasing the effectiveness of explicitly learning to remember whether objects have already been seen, and is complementarity with distance and direction prediction.

Table 4.4 presents results on the test set, confirming the impact on each of the considered metrics. *ProjNeuralMap* with auxiliary losses matches the performance of (incomparable) *OracleMap* on *Progress Rate* and *Success Rate*, again outperforming *OracleEgoMap* when considering all metrics. *OracleMap* has higher PPL and SPL, but has also access to very strong privileged information.

Interestingly, *OracleEgoMap* also benefits from the use of the auxiliary tasks at training time. As such agent already has access to priviledged information about the position of seen objects, this might suggest the auxiliary losses improve its spatial reasoning capabilities.

Can an unstructured recurrent agent learn to map? – we explore whether an agent without spatial inductive bias, i.e. the assumption that the representation of the environment must be a 2D map, can be trained to learn a mapping strategy, to encode spatial properties of the

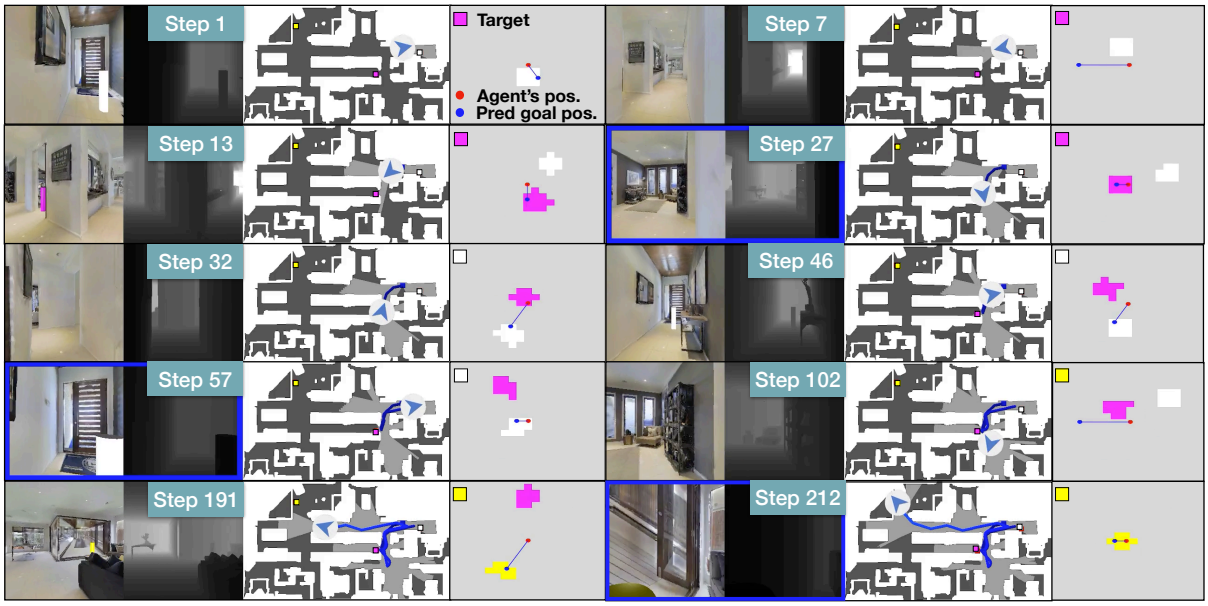


Figure 4.3: **Example agent trajectory** (sample from competition Mini-val set): The agent properly explores the environment to find the pink object. It then successfully backtracks to reach the white cylinder, and finally goes to the yellow one after another exploration phase (see text for a detailed description). In columns 3 and 6, the relative direction and distance predictions are combined into a visualised blue point on top of the oracle egocentric map (Ground-truth object positions). The red point corresponds to the position of the agent. Note that these predictions are not used by the agent at inference time, and are only shown for visualisation purposes. The top down view and oracle egocentric map are also provided for visualisation only.

environment into its unstructured hidden representation. As shown in Table 4.4, *NoMap* indeed strongly benefits from the auxiliary supervision (*Success Rate* for instance jumping from 16.7% to 43.0%). The improvement is important, as *NoMap* trained with auxiliary lossess outperforms *ProjNeuralMap* trained without auxiliary supervision, and closes the gap with *OracleEgoMap*. The quality of extra supervision can thus help to guide the learned representation, mitigating the need for incorporating inductive biases into neural networks. When both are trained with our auxiliary losses, *ProjNeuralMap* still outperforms *NoMap*, indicating that spatial inductive bias still provides an edge.

Comparison with the state-of-the-art – using our auxiliary losses to train *ProjNeuralMap* lead to the winning entry of the *CVPR 2021 Multi-On Challenge* organized with the *Embodied AI Workshop*, shown in Table 4.5. Test-standard is composed of 500 episodes and Test-challenge of 1000 episodes. In the context of the Challenge, the *ProjNeuralMap* agent was trained for $80M$ steps with the auxiliary objectives, and then finetuned for $20M$ more steps with only the vanilla RL objective. The official challenge ranking is done with **PPL**, which evaluates correct mapping (quicker and more direct finding of objects), while mapping does not necessarily have an impact on *Success Rate*, which can be obtained by pure exploration.

Visualization – Figure 4.3 illustrates an example trajectory from the agent trained with the

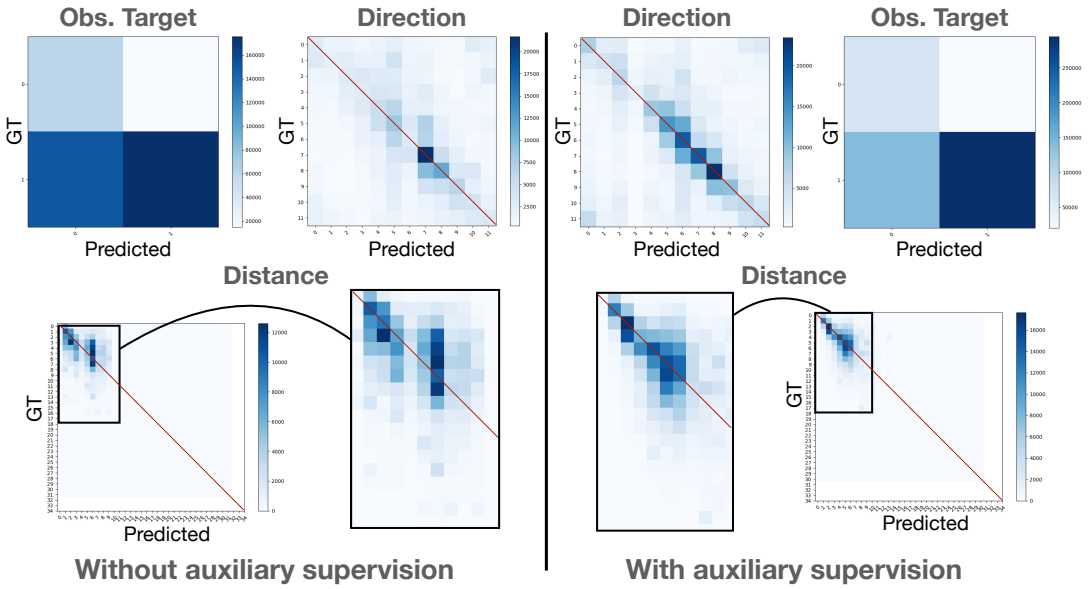


Figure 4.4: **Linear probes of representations optimized with and without auxiliary supervision**: Confusion matrices (validation set) of linear probes trained on representations from both *ProjNeuralMap* initially optimized with and without auxiliary supervision. Red lines indicate matrix diagonals.

auxiliary supervision in the context of the *CVPR 2021 Multi-ON Challenge*. The agent starts the episode (Step 1) seeing the white object, which is not the first target to reach. It thus starts exploring the environment (Step 7), until seeing the pink target object (Step 13). Its prediction of the goal distance immediately improves, showing it is able to recognize the object within the RGB-D input. The agent then reaches the target (Step 27). The new target is now the white object (that was seen in Step 1). While it is still not within its current field of view, the agent can localize it quite precisely (Step 32), and go towards the goal (Step 46) to call the *Found* action (Step 57). The agent must then explore again to find the last object (Step 102). When the yellow cylinder is seen, the agent can estimate its relative position (Step 191) before reaching it (Step 212) and ending the episode.

Information about observed targets, their relative distance and direction – Is such knowledge extracted by *ProjNeuralMap* without auxiliary supervision? We perform a probing experiment by training three linear classifiers to predict this information from the contextual representation of the GRU unit, both for *ProjNeuralMap* agent initially trained with and without auxiliary losses. We generate rollout trajectories on 1000 training and validation episodes. It is important to note that, as both agents behave differently, linear probes are not trained and evaluated on the same data. Fig. 4.4 shows that linear probes trained on representations from our method perform better, and more consistently, suggesting the presence of more related spatial information.

4.4 Conclusion

This chapter presented a method to guide the learning of mapping and spatial reasoning capabilities by augmenting vanilla RL training objectives with auxiliary tasks. We showed that learning to predict the relative direction and distance of already seen target objects, as well as to keep track of those observed objects, improved the performance on various metrics and that these gains were consistent over agents with or without spatial inductive bias.

The next chapter will also focus on improving the mapping abilities of neural agents in the context of the *Multi-ON* task, but instead of augmenting the supervision strategy, we will present an implicit mapping of semantic objects and occupancy, and more importantly will study how to train agents with RL to use such scene representations to navigate efficiently.

Chapter

Neural implicit representations as a means to better navigate to multiple objects

Abstract

As already motivated in the previous chapter 4, understanding and mapping a new environment are core abilities of any autonomously navigating agent. While classical robotics usually estimates maps in a stand-alone manner with SLAM variants, which maintain a topological or metric representation, end-to-end learning of navigation keeps some form of memory in a neural network. Networks are typically imbued with inductive biases, which can range from vectorial representations to birds-eye metric tensors or topological structures. In this work, we propose to structure neural networks with two neural implicit representations, which are learned dynamically during each episode and map the content of the scene: (i) the *Semantic Finder* predicts the position of a previously seen queried object; (ii) the *Occupancy and Exploration Implicit Representation* encapsulates information about explored area and obstacles, and is queried with a global read mechanism which directly maps from function space to a usable embedding space. Both representations are leveraged by an agent trained with RL and learned online during each episode. We evaluate the agent on *Multi-Object Navigation* and show the impact of using neural implicit representations as a memory source.

5.1 Context

Autonomous navigation in complex unknown 3D environments from visual observations requires building a suitable representation of the environment, in particular when the targeted navigation task requires high-level reasoning. Whereas classical robotics builds these representations explicitly through reconstructions, possibly supported through machine learning, end-to-end training learns them automatically either from reward, by imitation learning or through self-supervised objectives.

While spatial representations can emerge even in unstructured agents, as shown in the form of grid-cells in artificial (Cueva and Wei, 2018; Banino et al., 2018) and biological agents (Hafting et al., 2005), spatial inductive biases can support learning actionable spatial representations and decrease sample complexity. As already presented in previous chapters, popular inductive biases are metric maps (Parisotto and Salakhutdinov, 2018; Beeching et al., 2020b; Henriques and Vedaldi, 2018), topological maps (Beeching et al., 2020c; Chaplot et al., 2020d) and more recently, self-attention, adapting Transformers (Vaswani et al., 2017) to sequential decision-making and navigation (Fang et al., 2019; Du et al., 2021; Chen et al., 2022b; Reed et al., 2022). The chosen representation should support robust estimation of navigable space even in difficult conditions, mapping features and objects of interest, as well as querying and reusing this information at a later time. The representation should be as detailed as required, span the full (observed) scene, easy to query, and efficient to read and write to, in particular when training is done in large-scale simulations.

Our work builds on neural fields and implicit representations, a category of models which represent the scene geometry, and eventually the semantics, by the weights of a trained neural network (Xie et al., 2022). They have the advantage of avoiding the explicit choice of scene representation (e.g. volume, surface, point cloud etc.) and inherently benefit from the generalization abilities of deep networks to interpolate and complete unobserved information. As presented in chapters 2 and 3, neural implicit representations have demonstrated impressive capabilities in novel view synthesis (Mildenhall et al., 2020; Sitzmann et al., 2020), and have potential as a competitive representation for robotics (Ortiz et al., 2022; Li et al., 2022a; Sucar et al., 2021; Adamkiewicz et al., 2022). Their continuous nature allows them to handle level of detail efficiently through a budget given as the amount of trainable weights. This allows to span large environments without the need of discretizing the environment and handling growing maps.

We explore and study the potential of implicit representations as inductive biases for visual navigation. Similar to recent work in implicit SLAM (Sucar et al., 2021), our representations are dynamically learned in each episode. Going beyond, we exploit the representation dynamically in *MultiON*. We introduce two complementary representations, namely a queryable *Semantic Finder* trained to predict the scene coordinates of an object of interest specified

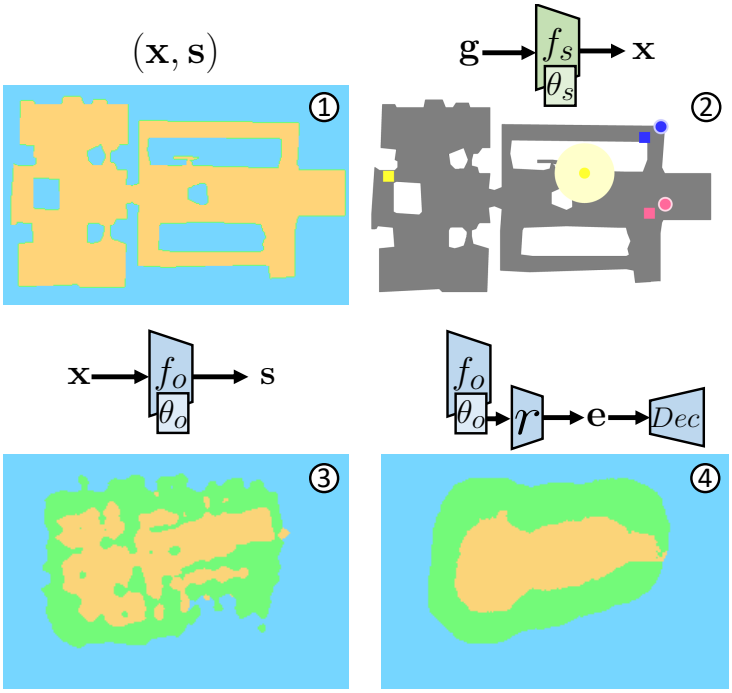


Figure 5.1: **Overview of the proposed neural implicit representations**: We propose **two implicit representations** as inductive biases for autonomous agents — both are learned online during each episode. ② a semantic representation f_s predicts positions $\mathbf{x}$ from goals $\mathbf{g}$ given as semantic codes. We show Ground Truth object positions (rectangles) and predictions (round; radius shows uncertainty, unit-less, as an illustration). Blue and pink objects have been observed, but not the yellow target. ③ A structural representation f_o predicts occupancy and exploration $\mathbf{s}$ from positions $\mathbf{x}$; we provide a global read which directly maps from function space f_o (represented by trainable weights θ_o) to a context embedding $\mathbf{e}$ used by the agent. ④ shows the reconstruction produced by a decoder *Dec* during training. Orange=navigable, Green=Obstacles, Blue=Unexplored. ① a ground-truth map is shown for reference, simulating a fully explored scene.

as input, and an *Occupancy and Exploration Implicit Representation*, which maps 2D coordinates to occupancy information, see Figure 5.1. We address the issue of the efficiency of querying an implicit representation globally by introducing a global read mechanism, which directly maps from function space, represented through its trainable parameters, to an embedding summarizing the current status of occupancy and exploration information, useful for navigation. Invariance with respect to reparametrization of the queried network is favored (but not enforced) through a Transformer-based solution. Our method does *not* require previous rollouts on the scene for pre-training or building a representation.

This chapter targets a fundamental aspect of visual and semantic navigation, the mapping of space and key objects of interest. Its content can be summarized as follows:

- We propose two implicit representations for semantic, occupancy and exploration information, which are trained online during each episode.
- We introduce a global read procedure which can extract summarizing context informa-

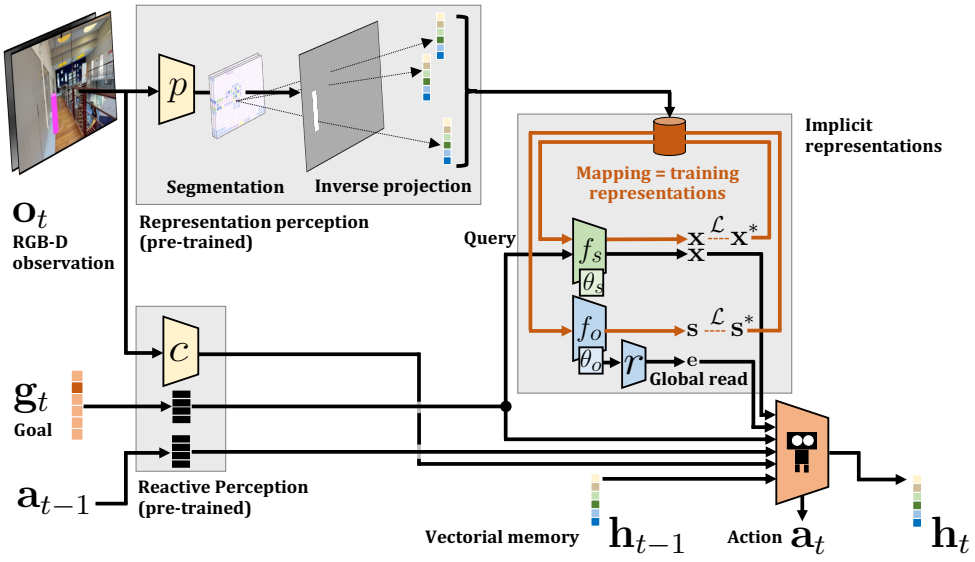


Figure 5.2: **Navigating with implicit representations**: Red connections ⟶ indicate the training process of the two implicit representations (=mapping), which is also done during agent deployment. Black connections ⟶ show the forward pass of the agent. ▮ are discrete learned embeddings (LUT). Policy training is not shown in this figure.

tion directly from the neural representation itself.

- We show that the representations lead to performance gains compared to classical neural agents.
- We evaluate and analyze key design choices, the representation's scaling laws and its capabilities of lifelong learning.

5.2 Navigating with implicit representations

We follow and augment the base end-to-end architecture presented in chapter 4, with the RGB-D observation, class of the current target and previous action as input to the agent. Temporal information is aggregated with a GRU unit whose output is fed to actor and critic heads. We equip this agent with two implicit representations, trained to hold and map essential information necessary for navigation: the positions of different objects of interest, and occupancy / exploration information, as shown in Figure 5.1,

- The goal of the *Semantic Finder* $f_s(.;\theta_s)$ parametrized by trainable weights θ_s is to predict the absolute position of an object as $\mathbf{x} = [\mathbf{x}_x\ \mathbf{x}_y\ \mathbf{x}_z] = f_s(\mathbf{q};\theta_s)$ specified through an input query vector $\mathbf{q}$. Uncertainty u is also estimated — see Section 5.2.1 for details. $\mathbf{x}$ is then converted into coordinates relative to the agent to be fed to the GRU. Compared to classical metric representations (Henriques and Vedaldi, 2018; Parisotto and Salakhutdinov, 2018; Chaplot et al., 2020d; Beeching et al., 2020b), querying the location of an object can be done through a single forward pass.

- The *Occupancy and Exploration Representation* $f_o(.;\theta_o)$ parametrized by trainable weights θ_o encodes information about free navigable space and obstacles. It predicts occupancy $\mathbf{s}$ as a classification problem with three classes *{Obstacle, Navigable, Unexplored}*, with $\mathbf{s} = f_o(\phi;\theta_o)$, ϕ being a position feature vector encoded from coordinates $\mathbf{x}$ — see Section 5.2.2 for details.

The *Occupancy and Exploration Representation* can in principle be queried directly for a single position, but reading out information over a large area directly this way would require multiple reads. We propose to compress this procedure by providing a trainable global read operation $r(.;\theta_g)$, which predicts an embedding $\mathbf{e}$ containing a global context about what has already been explored, and positions of navigable space. The prediction is done directly from the trainable parameters of the implicit representation, as $\mathbf{e} = r(\theta_o;\theta_r)$. Here θ_o is input to r, whereas θ_r are its parameters.

Given representations f_s and f_o, a single forward pass of the agent at timestep t and for a goal g_t involves reading the representations and providing the input to the policy. The current RGB-D observation o_t is also encoded by the convolutional network c (different from the projection module p used to generate samples for training the *Semantic Finder*). Previous action a_{t-1} and current goal g_t are passed through embedding layers, named $L(.)$ in the following equations. These different outputs are fed to the policy,

$$\mathbf{x}_t = f_s(g_t;\theta_{s,t}),\ \mathbf{e}_t = r(\theta_{o,t};\theta_r),\ \mathbf{c}_t = c(o_t;\theta_c), \tag{5-1}$$

$$\mathbf{h}_t = \text{GRU}(\mathbf{h}_{t-1}, \mathbf{x}_t, u_t, \mathbf{e}_t, L(a_{t-1}), L(g_t), \mathbf{c}_t;\theta_G), \tag{5-2}$$

$$a_t = \pi(\mathbf{h}_t;\theta_\pi), \tag{5-3}$$

where we added indices $\cdot_t$ to relevant variables to indicate time. Please note that the trainable parameters $\theta_{s,t}$ and $\theta_{o,t}$ of the two implicit representations are time dependent, as they depend on the observed scene and are updated dynamically, whereas the parameters of the policy π and the global reader r are not. Here, GRU corresponds to the update equations of a GRU network, where we omitted gates for ease of notation. The notation $a_t = \pi(.)$ predicting action a_t is also a simplification, as we train the agent with PPO — see Section 5.2.5.

Mapping means training! – The implicit representations f_s and f_o maintain a compact and actionable representation of the observed scene, and as such need to be updated at each timestep from the current observation o_t. Given their implicit nature and implementation as neural networks, updates are gradient-based and done with SGD. The implicit representations are therefore *trained from scratch at each episode even after deployment*.

Training a representation from observations obtained sequentially during an episode also raises a serious issue of catastrophic forgetting (Goodfellow et al., 2014), as places of the scene observed early might be forgotten later in the episode (Sucar et al., 2021; Zhi et al., 2021a). We solve this by maintaining two replay buffers throughout the episode, one for each representation. Training samples are generated from each new observation and added to the replay

buffers at each timestep. Both representations are then trained for a number of gradient steps (n_s for the *Semantic Finder* and n_o for the *Exploration and Occupancy Representation*). Details on the two representations and their training are given in Sections 5.2.1 and 5.2.2. The global reader r is not trained or fine-tuned online but rather trained once offline.

5.2.1 *The Semantic Finder* f_s

While recent work on implicit representations for robotics focused on signed distance functions (Ortiz et al., 2022; Li et al., 2022a), occupancy (Sucar et al., 2021) or density, assuming light density approximates mass density (Adamkiewicz et al., 2022), the aim of our model is to localize an object of interest within the scene, which can be seen as inverse operation to classical work. From a query vector given as input, the *Semantic Finder* predicts the position of the object, which is particularly useful for an agent interacting with an environment in the context of a goal conditioned task. It is implemented as a 3-layer MLP with ReLu activations in the intermediate layers and a sigmoid activation for the output. Hidden layers have 512 neurons. The query vector $\mathbf{q}$ corresponds to the 1-in-K encoding of the target object class, which during navigation is directly determined by the object goal g_t provided by the task.

Mapping/Training – The implicit representation is updated minimizing the L1 loss between the prediction $\mathbf{x}_i = f_s(\mathbf{q}_i; \theta_s)$ and the supervised coordinates $\mathbf{x}_i^*$ (we avoided the term "*ground-truth*" here on purpose), $\mathcal{L}_s = \sum_i \|\mathbf{x}_i^* - \mathbf{x}_i\|_1$, where the sum goes over the batch sampled from the scene replay buffer. Coordinates $\mathbf{x}_i^*$ are normalized $\in [0, 1]$.

The data pairs (x_i^*, q_i) for training are created from each observation o_t at each timestep, every data point corresponding to an observed point. Pixels in o_t are inversely projected into 3D coordinates in the scene using the depth channel, the camera intrinsics, as well as agent's coordinates and heading that are assumed to be available, as introduced by Wani et al. (2020). The query vector $\mathbf{q}$ corresponds to a 9-dimensional vector encoding a distribution over object classes (8 target objects and the "*background*" class). Let us recall that while the training of the representation is supervised, this supervision cannot use "ground-truth" information available only during training. All supervision information is required to be predicted from the data available to the agent even after deployment. We predict object class information through a semantic segmentation model p, which is applied to each current RGB-D observation $o_t \in \mathbb{R}^{h \times w \times 4}$, recovering the output segmentation map $\nu_t \in \mathbb{R}^{k \times l \times 9}$. The model has been pre-trained on the segmentation of the different target objects, i.e. colored cylinders, and is not fine-tuned during training of the agent itself.

Training data pairs (x_i^*, q_i) are sampled from this output. The supervised coordinates x_i^* are simply the mean 3D coordinates of each feature map cell, after inverse projection. The query vector $\mathbf{q}_i$ is the distribution over semantic classes. After the replay buffer is updated, a training batch must be sampled to update the neural field. One fourth of the samples in the batch of size b correspond to the $b/4$ last steps. The rest are sampled from the previous steps

in the replay buffer. Uniform sampling is also performed among pairs collected at a given timestep.

Estimating uncertainty — is an essential component, as querying yet unseen objects will lead to wrong predictions, which the agent needs to recognize as such, and discard. The estimation of uncertainty in neural networks is an open problem, which has been previously addressed through different means, including drop out as a Bayesian approximation (Gal and Ghahramani, 2014), variational information bottlenecks (Shah et al., 2021), density estimation (Kobyzev et al., 2021), and others. In this work, we approximate a density estimate in the scene replay buffer by calculating the minimum Euclidean distance between the input query and all embeddings in the replay buffer at the current timestep. The method is simple and efficient and does not require explicitly fitting a model to estimate the marginal distribution $p(\mathbf{q})$, in particular as the uncertainty representation is latent, can be un-normalized as not required to be a probability.

5.2.2 *Occupancy and Exploration Implicit Representation f_o*

Unlike f_s, the occupancy representation f_o is closer to classical implicit representations in robotics (Sucar et al., 2021; Ortiz et al., 2022; Li et al., 2022a; Adamkiewicz et al., 2022), which map spatial coordinates to variables encoding information on navigable area like occupancy or signed distances. Different to previous work, our representation also includes exploration information, which changes over time during the episode. Once explored, a position changes its class, which makes our neural field *dynamic*. Another difference with f_s is that the latter deals with 3D coordinates while f_o is a top-down 2D representation. Inspired by Xie et al. (2022) or Tancik et al. (2020), the model uses Fourier features ϕ extracted from the 2D coordinates $\mathbf{x}$ previously normalized $\in [0, 1]$,

$$\phi = (\cos(\mathbf{x}2^0), \sin(\mathbf{x}2^0), ..., \cos(\mathbf{x}2^{\frac{\kappa}{4}}), \sin(\mathbf{x}2^{\frac{\kappa}{4}})). \tag{5-4}$$

The network f_o is a 3-layer MLP with ReLu intermediate activations and a softmax function at the output layer. Hidden layers have 512 neurons, and $\kappa = 40$.

Mapping/Training – The implicit representation is updated minimizing the Cross Entropy loss between the prediction $\mathbf{s}$ of the neural field and the supervised label $\mathbf{s}^*$ of three classes *{Obstacle, Navigable, Unexplored}*, as $\mathcal{L}_o = -\sum_{c=1}^{3} \mathbf{s}_\mathbf{c}^* \log \mathbf{s}_c$. As for the *Semantic Finder*, training data pairs $(\mathbf{s}^*, \mathbf{s})$ are created through inverse perspective projection of the pixels of the observation o_t into 3D scene coordinates. Thresholding the z (height) coordinate decides between *Navigable* and *Obstacle* classes. Points with a z coordinate higher than a certain threshold are discarded. The replay buffer is balanced between both classes, and only samples of the last 1000 steps are kept. Samples of the *Unexplored* class are not stored.

The replay buffer is sampled similarly to the one for the *Semantic Finder*. However, additional samples for the *Unexplored* class are created by sampling uniformly inside the scene, for

speed reasons simply ignoring conflicts with explored areas and treating them as noisy labels.

5.2.3 *Global Occupancy Read r — handling reparametrization invariance*

The global Occupancy reader r allows to query the occupancy information of the scene globally, beyond point-wise information, and as such is a trainable mapping from the space of functions $f_o(.;.)$ to an embedding space $\mathbf{e}$. In particular, two functions f_o and f'_o s.t. $f_o(\mathbf{x}) = f'_o(\mathbf{x})\ \forall \mathbf{x}$ should be mapped to identical or close embeddings. However, as the occupancy networks f_o are implemented as MLPs, any given instance $f_o(.;\theta_o)$ parametrized by trainable weights θ_o can be reparametrized by any permutation of hidden units, which leads to permutations of the rows and columns, respectively, of two weight matrices, its own and the one of the preceding layer. This reparametrization keeps the functions identical, although their representations as weight vectors are different.

To favor learning a global occupancy reader which is invariant with respect to these transformations, we implement it as a Transformer model with self-attention (Vaswani et al., 2017) — this, however, does not enforce full invariance. The model takes as input a sequence of tokens $(w_1, ..., w_{N_n})$, where $w_i \in \mathbb{R}^a$ is a learned linear embedding of the incoming weights of one neuron within the implicit representation f_o, and N_n is the number of neurons of f_o. Each token is summed with a positional encoding in the form of Fourier features. An additional "*CLS*" token with learned embedding is concatenated to the input sequence. The reader is composed of 4 self-attention layers, with 8 attention heads. The output representation of the "*CLS*" token is used as the global embedding of the implicit representation.

Training – The global reader r is trained with full supervision from a dataset of $25k$ trajectories composed of MLP weights $\theta_{o,i}$ and absolute maps $\mathbf{M}_i$, $i = 1..25k$. Each map is a metric tensor providing occupancy information extracted from the corresponding implicit representation, i.e. $\mathbf{M}_i(\mathbf{x}_y, \mathbf{x}_x) = f_o(\mathbf{x}, \theta_{o,i})$. The dataset also contains an egocentric version $\mathbf{M}'_i$ of each map, which is centered on the agent and oriented depending on its current heading. The reader r is trained in an Encoder-Decoder fashion, where r plays the role of the encoder,

$$\mathbf{e}_i = r(\theta_{o,i}), \quad \hat{\mathbf{M}}_i = Dec(e_i, p_i), \tag{5-5}$$

where p_i is the agent pose (position and heading), necessary to decode egocentric information. We minimize a cross entropy loss on the prediction of egocentric maps,

$$\mathcal{L}_g = -\sum_i \sum_k \sum_l \sum_{c=1}^{3} \mathbf{M}'^{*}_{i,c}(k,l) \log \mathbf{M}'_{i,c}(k,l) \tag{5-6}$$

Directly training this prediction proved to be difficult. We propose a procedure involving several steps, detailed in section 5.2.6. After the training phase, the reader g is used in the perception + mapping module of the agent as given in equation 5-1, and kept frozen during agent RL training.

5.2.4 *Algorithmic description of an agent forward pass*

Algorithm 4 gives a high-level overview of the different steps happening after receiving the current observation from the environment to take the most suitable action.

Lines $1-5$ **— Adding training samples to the semantic replay buffer**

The segmentation map v_t is obtained by passing the RGB-D observation o_t through the representation perception module, i.e. a segmentation model pre-trained to segment the target objects (Line 2). An inverse projection operation, denoted $invProj()$ is used to project pixels from o_t into their 3D coordinates n_t using the depth channel of o_t and the known camera intrinsics K (Line 3). A meanpooling operation, denoted $meanPooling()$ is then applied to n_t in order to obtain the mean 3D coordinates of all pixels in each cell of the segmentation map v_t (Line 4). Finally, pairs of softmax distribution over classes from v_t and mean 3D coordinates are added to the training replay buffer of the *Semantic Finder*. This is implemented as the $addSemSamples()$ in the algorithm (Line 5).

Lines $6-8$ **— Adding training samples to the occupancy replay buffer**

The 3D coordinates of projected pixels in n_t are compared with threshold values along their vertical coordinate to be either labelled as navigable space or obstacle. Only 3D points with a vertical coordinate below than another threshold value are kept. These comparisons are done in the $labelOccPos()$ function (Line 7). Pairs of label and 2D coordinates (discarding vertical coordinates) are then sampled in order to keep the balance between the two classes and added to the training replay buffer of the *Occupancy and Exploration Implicit Representation*. This is the $addOccSamples()$ function (Line 8).

Lines $9-13$ **— Updating the Semantic Finder**

Two operations are repeated n_s times. First a batch of training examples b_s is sampled ($getSemBatch()$, line 11). Then, the $SGD()$ function encapsulates the forward pass of f_s on the sampled batch, the L1 loss computation, gradient computation and finally backpropagation. In this work, we fixed $n_s = 1$.

Lines $14-22$ **— Updating the Occupancy and Exploration Implicit Representation**

The implicit representation is iteratively updated for a maximum of n_o steps while the error of the model ($loss$, initialized to 0 in line 15) is higher than a threshold. Same as for the *Semantic Finder*, a training batch b_o is first sampled ($getOccBatch()$, line 18). The model is then evaluated on samples from b_o (Line 19). The $SGD()$ function is then applied to update the implicit representation. In this work, we chose $n_o = 20$ and $thresh = 0.3$.

Algorithm 4: Different steps necessary to update implicit representations at each agent step.

Input : Observation o_t, camera intrinsics K, goal g_t, replay buffers r_s and r_o, weights $\theta_{s,t}$, $\theta_{o,t}$

// Adding training samples to the semantic replay buffer
$v_t = p(o_t)$
$n_t = \texttt{invProj}(o_t, K)$
$k_t = \texttt{meanPooling}(n_t)$
$r_s = \texttt{addSemSamples}(r_s, v_t, k_t)$
// Adding training samples to the occupancy replay buffer
$l_t = \texttt{labelOccPos}(n_t)$
$r_o = \texttt{addOccSamples}(r_o, n_t, l_t)$
// Updating the Semantic Finder
for $i \leftarrow 0$ **to** $n_s - 1$ **do**
    $b_s = \texttt{getSemBatch}(r_s)$
    $\theta_{s,t} = \texttt{SGD}(b_s, \theta_{s,t})$
end for
// Updating the Occupancy and Exploration Implicit Representation
$loss = 0$
$j = 0$
while $loss > thresh$ *and* $j < n_o$ **do**
    $b_o = \texttt{getOccBatch}(r_o)$
    $loss = \texttt{eval}(b_o, \theta_{o,t})$
    $\theta_{o,t} = \texttt{SGD}(b_o, \theta_{o,t})$
    $j = j + 1$
end while

5.2.5 *Training the agent*

The agent is trained with RL, more precisely PPO (Schulman et al., 2017). The inner training loops of the implicit representations are supervised (**red arrows** in Figure 5.2) and occur at each timestep in the forward pass, whereas the RL-based outer training loop of the agent occur after N_a acting steps (**black arrows** in Figure 5.2). As the perception module used to generate training data from RGB-D observations for the *Semantic Finder* f_s is independent of the visual encoder c in the agent (see Figure 5.2), and as its query $\mathbf{q}_t$ is fixed to the navigation goal $\mathbf{g}_t$, there is no need to track the weights θ_s at each timestep in order to backpropagate the PPO loss (outer training). This is a key design choice of our method.

Training assumptions – We do not rely on the existence of global ground-truth maps for occupancy, as $f_o()$ and $r()$ were trained on observation data from agent trajectories only. However, as done in the previous chapter 4, we exploit object positions in simulation, and require pixel-wise segmentation masks (similar to other work such as Pashevich et al. (2021)) during the pre-training of the segmentation head only (not for RL agent training, or at inference time).

5.2.6 *Training the global reader*

The proposed training procedure for the global reader r (performed *before* training the agent) can be split into 3 phases. The architecture for the convolutional decoder $Dec()$ is kept the same in all of them. The hyperparameters of its different layers are detailed in subsection 5.2.7. It is composed of 6 transpose convolution layers along with batch norm layers and ReLU activations, except for the last layer with a softmax activation. Figure 5.3 provides an overview of the 3 steps involved in the training of the global reader.

Fully convolutional autoencoder — The first step is to train a fully convolutional autoencoder on the set of absolute maps $\mathbf{M}_i$, $i = 1..25k$. Only the decoder weights are kept.

Global reader training on absolute maps — The second step consists in training the global reader to output embeddings fed to the frozen decoder from the previous step. The objective is to reconstruct absolute maps from the weights of the implicit representation. The global reader weights are kept after this training phase.

Global reader fine-tuning on egocentric maps — The global reader, whose weights are initialized from the weights obtained in the previous step, is now trained along with the same decoder from the first step (also used in the second step) on the set of egocentric maps $\mathbf{M}'_i$, $i = 1..25k$. Both the global reader and decoder are fine-tuned. The output of the global reader is not directly fed to the decoder, but is passed through linear layers in order to fuse information about first the position of the agent, and then its heading because this time the right operations of shift and rotation must be applied in order to reconstruct egocentric maps.

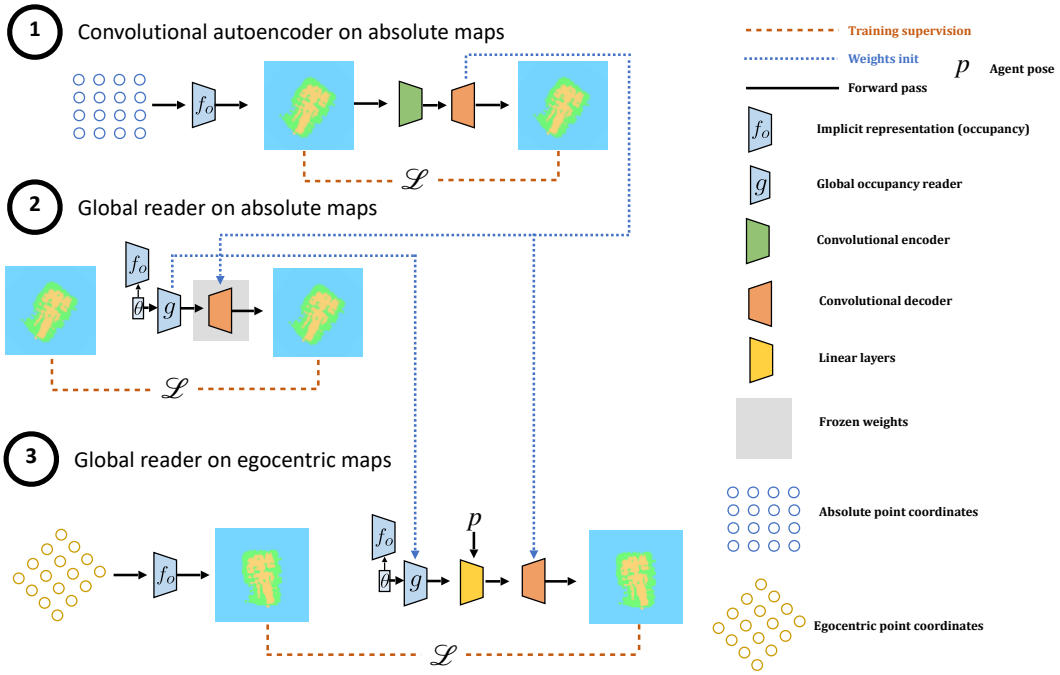


Figure 5.3: **Training the Global Reader** in 3 steps: ① Training a convolutional auto-encoder on absolute maps; ② extension to weight-based (θ_o) inputs; ③ extension to predicting egocentric maps, better suited to navigation problems.

Integrating the pre-trained global reader into the agent — After this pre-training in 3 steps, the weights of the global reader are frozen and not updated during the RL training. However, a linear layer is learned to project the 576-dim embedding from the global reader into a 256-dim representation fed to the GRU. This linear layer is trained from reward signals.

5.2.7 *Perception modules*

Two different perception modules are used in the presented method. The first one, responsible for *representation perception*, extracts representations from the RGB-D observation to populate the training replay buffer of the *Semantic Finder*. The second one, tackling *reactive perception*, encodes the observation into a vector fed to the GRU. This representation of the observation is thus more directly used in the decision-making process — the name *Reactive* is certainly not 100% accurate, since the output of this module is still used to update agent memory, but this concerns only the hidden GRU memory and not the main implicit representations.

Reactive perception We use the encoder module from Wani et al. (2020), which encodes visual observations at each step. Table 5.1 (*Enc*) presents the hyperparameters of the convolutional layers in this visual encoder. It is composed of 3 convolutional layers followed by a linear layer. ReLU activations are used. The embedding produced by this visual encoder is fed to the GRU module. In some of our experiments (presented later in this chapter), this reactive perception module was pre-trained with auxiliary losses

Model	layer id	type	in channels	out channels	kernel size	stride	in padding	out padding
Dec	0	TransposeConv2D	64	32	3	2	0	0
	1	TransposeConv2D	32	32	3	2	0	0
	2	TransposeConv2D	32	16	3	2	0	0
	3	TransposeConv2D	16	8	3	2	0	0
	4	TransposeConv2D	8	8	3	2	0	0
	5	TransposeConv2D	8	3	3	2	0	1
Enc	0	Conv2D	4	32	8	4	0	–
	1	Conv2D	32	64	4	2	0	–
	2	Conv2D	64	32	3	1	0	–
Seg	0	Conv2D	32	32	5	1	2	–
	1	Conv2D	32	32	5	1	2	–
	2	Conv2D	32	9	3	1	1	–

Table 5.1: **Architecture of involved convolutional layers**: Hyperparameter values in the different presented CNN architectures. *Dec* is the CNN decoder trained with the global reader, *Enc* is the visual encoder used in both the representation and reactive perceptions, *Seg* is the segmentation head combined with *Enc* in the representation perception module *c*.

presented in the previous chapter 4, and was then frozen and not updated during RL training. We also compare other variants where it is updated during the RL-based agent training.

Representation perception The goal of the representation perception module is to extract vectors to be added to the *Semantic Finder* training replay buffer. The backbone encoder is the same as the reactive perception module (see *Enc* in Table 5.1). This network is augmented with a segmentation head and is fine-tuned end-to-end on the task of segmenting *MultiON* target objects (before agent training). Table 5.1 (*Seg*) details the architecture of the segmentation head. It is composed of 3 convolutional layers with ReLU activations, except for the last layer where a softmax activation is applied. After this training phase, the weights of the representation perception module are frozen and not updated during RL training.

5.3 Experimental results

Task, dataset, episodes and metrics – The same task (*MultiON* – 3-ON), dataset (Matterport3d Chang et al. (2018)), training and evaluation episodes (train/val/test split), as well as metrics (*Success Rate*, *Progress Rate*, *SPL*, *PPL*) as in chapter 4 are used in this section.

***Global reader* dataset** – The *Global reader r* was trained on a dataset of 25*k* trajectories obtained from rollouts performed by the best agent from chapter 4 (*ProjNeuralMap* + auxiliary losses). 95% were used for training and the rest for validation. On these trajectories we first trained the

	0 – 30		30 – 50		50 – 70		— Val —				— Test —			
	S	O	S	O	S	O	**Success**	**Progress**	**SPL**	**PPL**	**Success**	**Progress**	**SPL**	**PPL**
	–	–	–	–	–	–	33.2± 1.2	49.0± 1.1	21.2± 0.5	31.6± 1.2	42.3± 1.5	56.7± 0.9	28.1± 1.0	37.8± 1.8
μ	–	–	✓	–	✓	–	37.8± 1.6	52.3± 0.9	26.35± 1.5	36.5± 0.8	**47.0**± 1.7	**60.5**± 1.6	34.5± 0.8	44.2± 1.0
	–	–	✓	–	✓	✓	**38.5**± 4.6	**52.5**± 4.8	**28.2**± 2.1	**38.3**± 1.7	46.7 ± 3.0	60.1 ± 3.1	**35.1** ± 1.4	**44.8** ± 1.0
	–	–	–	–	–	–	32.1	47.7	21.6	32.6	41.0	55.9	28.9	39.0
↑	–	–	✓	–	✓	–	38.1	51.9	27.3	37.1	48.6	61.5	35.2	44.2
	–	–	✓	–	✓	✓	**43.1**	**56.8**	**30.5**	**40.1**	**49.7**	**63.4**	**36.4**	**45.9**

Table 5.2: **Impact of the implicit representations**: Navigation performance on *MultiON val* and *MultiON test*. S=f_s activated, O=f_o activated in the corresponding training period (see text). Top/μ: means over 3 runs (Performance is reported as *mean* ± *std* over runs); Bottom/↑: best validation seeds over 3 runs.

Uncertainty	**Success**	**Progress**	**SPL**	**PPL**
–	35.4± 3.0	49.7± 3.3	29.4± 2.0	40.9± 2.4
✓	**43.4**± 3.1	**58.0**± 3.0	**35.1**± 0.8	**46.4**± 1.0

Table 5.3: **Importance of the uncertainty estimation**: Comparing training w/ semantic input only, no occupancy, from the beginning of training, w/ and w/o uncertainty. Performance is reported as *mean* ± *std* over training runs (seeds).

occupancy representation f_o "*in-situ*", i.e. as if it were deployed on the agent, and we recorded training samples i for training the reader r: pairs of network weights $\theta_{o,i}$ and associated maps M_i obtained by iteratively querying the implicit representation. Egocentric maps were generated from the absolute ones and both were cropped around their center.

Perception module dataset – The perception module p was trained to segment the different target objects. The generated dataset is composed of 132k pairs of RGB-D observations and segmentation masks. Samples from 4 scenes were kept as a validation set.

Training details – We use the reward function presented in the previous chapter 4 (equation 4-6), and train all agents for the same number of frames as well (70M steps). PPO hyperparameters are the same as in chapter 4 (Table 4.2). For all agents in Table 5.2 and some in Table 5.4 (*w/ pre-train*: ✓ in ρ column), the encoders (visual encoder c, as well as goal and previous action embedding layers, see Figure 5.2) are pre-trained with a baseline, which corresponds to the *ProjNeuralMap* agent trained with auxiliary losses from chapter 4. This is done to faster training, as it will be shown later (in Table 5.4) that the same final performance can be reached without this initial pre-training of encoders. All reported quantitative results for our method are obtained after 3 training runs for each model. We report the average performance and standard deviation among training runs (random seeds) for each variant as *mean* ± *std*.

Impact of the implicit representations – Table 5.2 shows the impact of the two implicit representations on navigation (top: means over 3 runs; bottom: best validation seeds over 3 runs). To keep compute requirements limited and decrease sample complexity, in these ablations

	Agent	ρ	α	γ	**Success**	**Progress**	**SPL**	**PPL**	AUX	ORC
(a)	OracleMap	–	✓		50.4± 3.5	60.5± 3.1	40.7± 2.2	48.8± 1.9	–	✓
(b)	OracleEgoMap	–	✓		32.8± 5.2	47.7± 5.2	26.1± 4.5	37.6± 4.7	–	✓
(c)	NoMap	–	✓		16.7± 3.6	33.7± 3.3	13.1± 2.4	26.0± 1.7	–	–
(d)	ProjNeuralMap	–	✓		25.9± 1.1	43.4± 1.0	18.3± 0.6	30.9± 0.7	–	–
(e)	NoMap	✓	–		42.3± 1.5	56.7± 0.9	28.1± 1.0	37.8± 1.8	–	–
(f)	ProjNeuralMap	✓	–		39.7± 2.3	55.4± 1.4	28.7± 1.1	40.1± 1.9	–	–
(g)	Ours *w/ curriculum w/ pre-train*	✓	–	–	46.7± 3.0	60.1± 3.1	35.1± 1.4	44.8± 1.0	–	–
(h)	ProjNeuralMap + *AUX*	N/A	✓	N/A	57.7 ± 3.7	70.2 ± 2.7	37.5 ± 2.0	45.9 ± 1.9	✓	–
(i)	Ours *w/o curriculum w/ pre-train + AUX*	✓	✓	✓	58.3± 0.8	69.4± 1.1	**43.8**± 1.0	**52.1**± 1.6	✓	–
(j)	Ours *w/o curriculum w/o pre-train*	–	✓	✓	54.8± 3.6	68.0± 3.4	41.7± 1.9	51.3± 1.6	–	–
(k)	Ours *w/o curriculum w/o pre-train + AUX*	–	✓	✓	57.9± 2.0	69.5± 0.6	**43.3**± 2.2	**51.9**± 3.7	✓	–

Table 5.4: **Comparison with state-of-the-art methods** on *MultiON test*:"*AUX*" = auxiliary losses using privileged information. "*ORC*"=non-comparable, uses oracle information. ρ = pre-training of input encoders from agent trained with auxiliary losses. α = fine-tuning of input encoders with RL. γ = implicit representations are accessible to the agent since the beginning of RL training (*w/o curriculum*). (h) is the agent from chapter 4. Performance is reported as *mean* $\pm$ *std* over training runs (seeds).

we do not train the full agent from scratch, in particular since the early stages of training are spent on learning basic interactions. We decompose training into three phases: $0-30M$ steps (no implicit representations, i.e. all entries to the agent related to f_s and f_o are set to 0); $30M-50M$ steps (training includes the *Semantic Finder* f_s) and finally $50M-70M$ steps (full model). This 3-steps approach is used to train all agents in Table 5.2, and will be denoted as *curriculum* (See Table 5.4, *w/ curriculum*: – in γ column). All metrics on both val and test sets are improved, with the biggest impact provided by the *Semantic Finder*, which was expected. We conjecture that mapping object positions is a more difficult task, which is less easily delegated to the vectorial GRU representation, than occupancy. We also see an impact of the occupancy representation, which not only confirms the choice of the implicit representation f_o itself, but also its global read through $r(\theta_o)$.

Uncertainty – has an impact on agent performance, as we show in the ablation in Table 5.3. Indeed, when training an agent with the semantic input since the beginning of training (*w/o curriculum*) and no occupancy input (as the uncertainty is only related to semantic information), feeding the agent with the computed uncertainty about the output of the *Semantic Finder* brings a boost in performance.

Comparison with previous SOTA methods – is done in Table 5.4. The performance entries of these baselines are taken from the previous chapter 4. The introduced method outperforms the different competing representations, even when they benefit from the same pre-training scheme and are thus completely comparable. *NoMap* with pre-training corresponds to the first row of Table 5.2. The difference between (g) and (i) is the use of the auxiliary tasks

presented in the previous chapter 4, but also that the implicit representations are available to the agent during the whole training period for (i), i.e. no decomposition into 3 phases as in the ablations in Table 5.2 (*w/o curriculum*). Moreover, compared to (g), in (i) the weights of the pre-trained encoders are fine-tuned. We see that the gains of our representations are complementary to the auxiliary losses. (j) and (k) confirm this finding, showing that most of the gain compared with (h) comes from the implicit representations, with the auxiliary losses bringing an additional boost. (j) and (k) also show that, even though pre-training can help speed up RL training, similar test performance is achieved without it.

Visualization of the agent behavior – Figure 5.4 provides an example for an episode rollout from the minival set of the *MultiON* CVPR 2021 Challenge. The agent is equipped with both implicit representations. For each line, from left to right, we first see the RGB-D egocentric view of the agent, then a topdown map with the three goals (white, pink and yellow squares) and their estimated location by the *Semantic Finder* (white, pink and yellow dots, with a shaded region to denote uncertainty). The radius illustrating uncertainty is unit-less and only given for visualization purposes - is not available to the agent in this particular form. The third illustration shows the implicit map obtained by querying the *Occupancy and Exploration Implicit Representation*, and on the right, there is the reconstructed output when feeding the embedding of the global reader to the convolutional decoder it was trained with. The last element is a curve showing the evolution of the uncertainty estimation of the *Semantic Finder* on the currently provided target object.

In this episode, the agent starts with the white object within its field of view, but the first target to reach is the pink cylinder (Row 1). As we can see, the estimation of goal positions from the *Semantic Finder* are wrong, which is expected as the episode has not yet started. However, the associated uncertainty is high, allowing the agent to discard this information. The agent then explores until it observes the pink object (Row 2). At that point, the uncertainty about the object to find drops. The estimate of the position of the pink object will be updated as training samples will be added to the semantic replay buffer. Also note that at that point the estimate of the position of the white object from the *Semantic Finder* is accurate as the object has already been seen previously. The agent then goes towards the pink target object and calls the *Found* action (Row 3). Estimation of the positions of pink and white objects are accurate. The next target to find is the white object. The uncertainty about the current target is low as the white object has already been observed. The agent backtracks (Row 4) and goes towards the white object to call the *Found* action (Row 5). The next goal is the yellow cylinder. At that point, the uncertainty about the current target increases as the yellow cylinder has not yet been within the agent's field of view. The agent explores (Row 6) and when the target is within its field of view (Row 7) the uncertainty related to the target to find drops. The agent goes towards the yellow object and calls the *Found* action (Row 8). At the end of the episode, the *Semantic Finder* is able to localize the 3 objects, and the associated uncertainties are low.

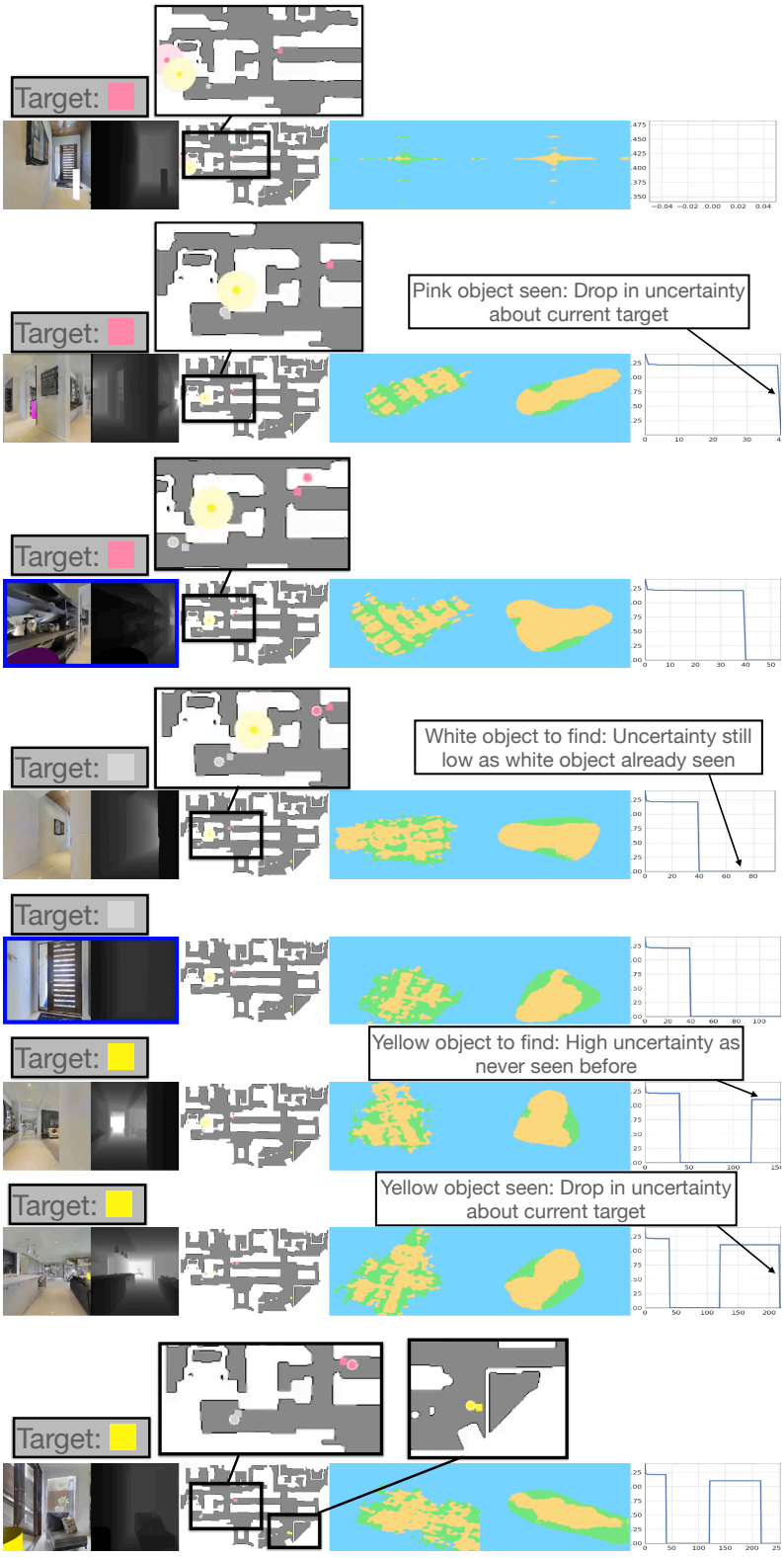


Figure 5.4: **Agent rollout on an example episode** from the *MultiON* CVPR 2021 Challenge minival set. From left to right: RGB-D ego view, topdown map (viz only) with targets (squares) and their estimated location by the *Semantic Finder* (dots, shaded region for uncertainty), map from the *Occupancy and Exploration Implicit Representation*, reconstructed egomap from the global reader and CNN Decoder trained end-to-end, uncertainty of the *Semantic Finder* on the currently selected target.

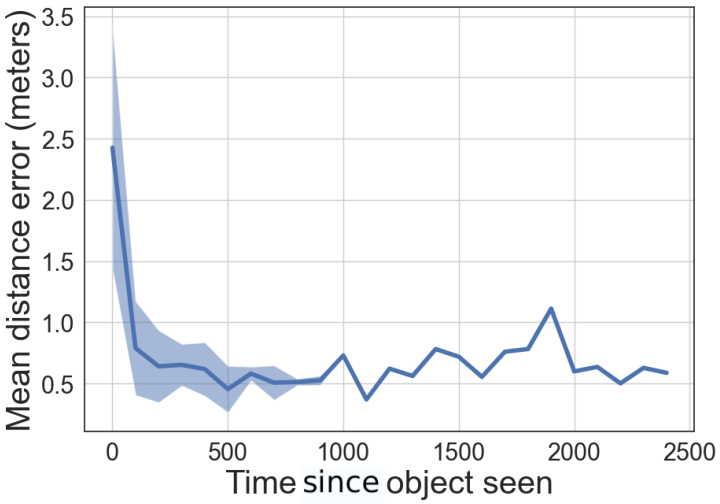


Figure 5.5: **Lifelong learning of the semantic representation** f_s: We report mean error in meters, test set, 600 episodes, as a function of the number of timesteps since the object was first seen in the episode (t=0). The error falls immediately and stays low over the episode.

All objects have been successfully found, so this episode is considered as a success.

Reconstruction performance of the Global Reader r – although the task of reconstructing egocentric maps from occupancy functions f_o (represented by θ_o) is only used as a proxy task to train the Global reader r, we see it is a reliable proxy for the quality of extracting the global latent vector **e** fed to the agent.

Accuracy	Jaccard Index
83.4	56.5

Table 5.5: **Performance of the global reader** r: We report accuracy and Jaccard index.

In Table 5.5 we report reconstruction performance measured as accuracy and mean Jaccard Index on the validation split of the dataset used to train the reader. We judge that an accuracy of 83.4% is satisfactory, given that the global reader needs to reconstruct the content of the representation directly from its parameters θ_o, that *each implicit representation has been initialized randomly*, and that the reader is required to be invariant with respect to reparametrization (see Section 5.5). The task is even made harder as neural weights can be considered as an absolute representation of the environment and the reader must combine it with information about the agent pose to reconstruct an egocentric map.

Evaluating catastrophic forgetting – we evaluate the capacity of the *Semantic Finder* f_s to hold the learned information over the full length of the sequence in spite of the fact that it is continuously trained. Figure 5.5 shows the evolution of the mean error in distance for the predicted position of queried target objects as a function of time. The error quickly goes below 1.5m once the object has been seen the first time (t=0 in the plot), which is the distance threshold required by the *MultiON* task, and stays there, providing evidence that the model does not suffer from catastrophic forgetting.

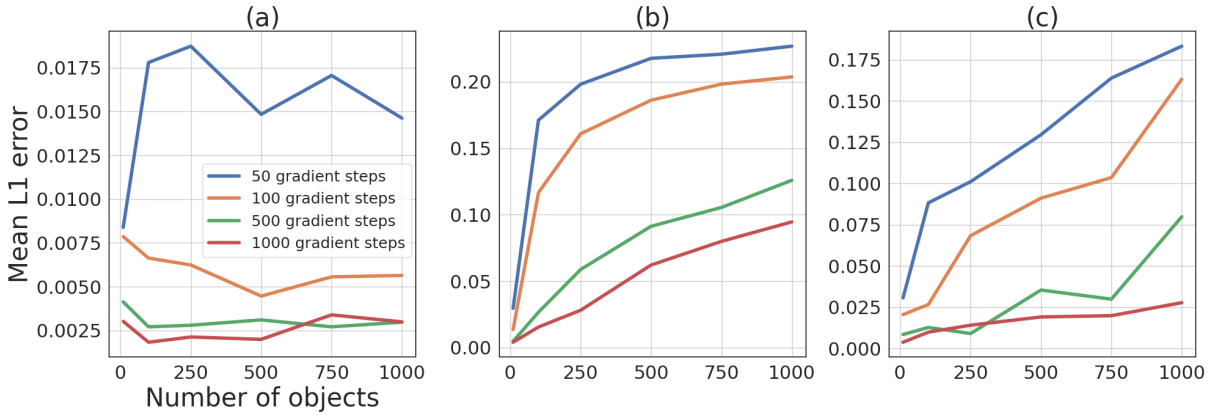


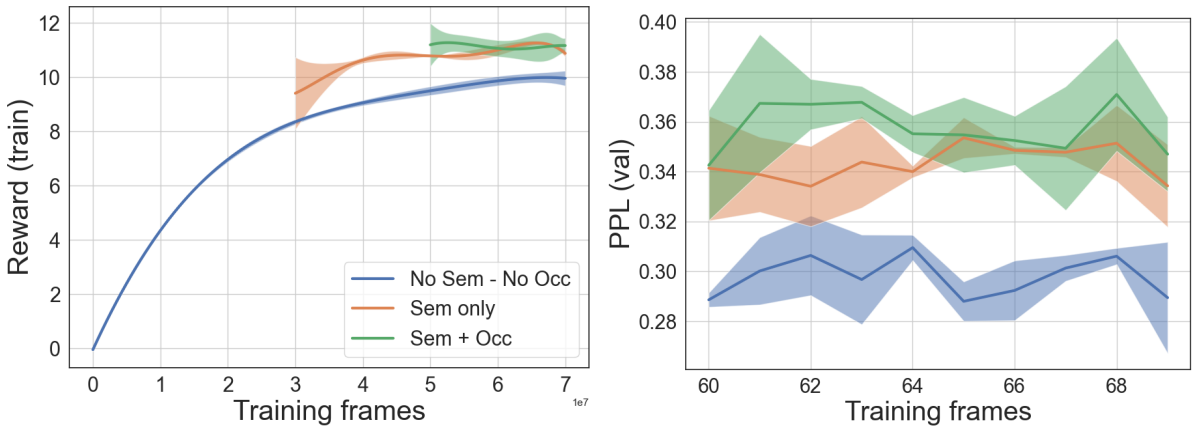
Figure 5.6: **Capacity of the semantic representation**: We report mean distance prediction error (normalized $\in [0, 1]$) as a function of the number of stored objects. Replay buffers are composed of dummy queries: (a) one-hot queries with same dimension as number of objects; (b) random query with dimension 9; (c) random query with same dimension as number of objects.

Capacity of the Semantic Finder – Unlike all other results presented in this chapter, this experiment is performed independently of the official *MultiON* benchmark. As the granularity of the implicit representations is handled through the budget in terms of trainable parameters, we construct a synthetic dataset to evaluate the capacity of the *Semantic Finder* f_s to store large numbers of objects. Results are presented in Figure 5.6.

We consider three new scenarios, and for each one, a dataset is generated and used to train the *Semantic Finder*. All datasets are made of (query, position) pairs with positions being uniformly sampled between arbitrary scene bounds (between 0 and 1 along each axis). For each dataset, we also create variants varying the sample size. To reduce the amount of hyperparameters (e.g. batch size), we resort to gradient descent as opposed to stochastic gradient descent, i.e. each gradient step is computed over the whole dataset. The considered metrics is the mean L_1 error on the prediction of positions as a function of the number of objects to memorize. The difference between the three scenarios is in the nature of queries associated with positions, and each scenario corresponds to a sub figure of Figure 5.6.

In Figure 5.6(a), for a given size of the dataset, i.e. for a given number of objects, each query is a 1-in-K encoded vector of the object category, which means that the query dimensions grow with increasing numbers of objects. This evaluates the representation in situations where objects are identified by a unique class index. Provided a sufficient number of gradient steps, we can see that the error stays low even with an increasing number of objects. We conjecture that the good performance of this setting is due to the growth in, both, query size and thus capacity of the model (as the query is the input to the model) as the number of objects grows.

In Figure 5.6(b), the query vectors have a fixed dimension of 9, equivalent to the dimension in the *MultiON* benchmark. Queries are not 1-in-K encoded, but composed of randomly



(a) (b)

Figure 5.7: **Training stability**: (a) Evolution of the collected reward on training episodes for the 3 models presented in Table 1 in the main paper. (b) Evolution of PPL, the official ranking metrics in the *MultiON* Challenge leaderboard, on val episodes for model checkpoints from the last 10M training frames.

sampled values. Even though more gradient steps are helpful, the conclusion here is that increasing the number of objects has a negative impact on the mean error of the model. Unlike in (a), the number of parameters does not increase here with number of objects as queries have a fixed size. This is thus an illustration of the challenge to memorize an increasingly high number of objects with a fixed model capacity.

Figure 5.6(c) is a combination of (a) and (b) with query size increasing with number of objects and queries composed of random values (no one-hot vectors). The increase in model capacity with more objects seems again to be beneficial provided enough gradient steps. However, its is clear that the positions associated with random queries are more difficult to memorize than for one-hot queries. This emphasizes the importance of the chosen query representation when building query-able semantic implicit representations.

Runtime performance – despite requiring to continuously train the representations, we achieve 45 fps during parallelized RL training, including the environment steps (simulator rendering), forward passes, representation training and RL training on a single V100 GPU. The average time of one agent forward pass, including updates of implicit representations is 20ms on a V100 GPU, which is equivalent to 50 fps, enough for real-time performance.

Training stability – Figure 5.7 shows the training curves of the 3 different agents we compared in Table 5.2: the recurrent baseline agent (blue), with the *Semantic Finder* (orange) and with both implicit representations (green). On the left, (a), we see the evolution of the training reward as a mean and standard deviation over 3 runs. The right part, (b), shows PPL, the main metric chosen for ranking the agents in the *MultiON* leaderboard, which we show for different checkpoints during training and evaluated on the **validation set**. As can be seen,

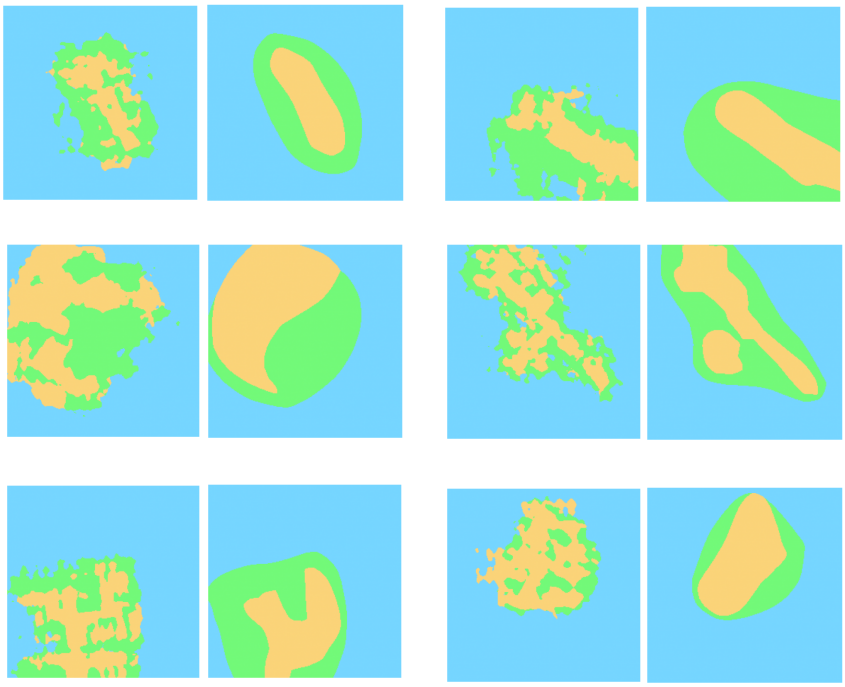


Figure 5.8: **Importance of Fourier features**: Comparison of top-down maps obtained by querying the *Occupancy and Exploration Implicit Representation* trained with (left) and without (right) Fourier features.

training is quite stable over runs, and adding the two representations provides a boost in performance (as already reported in Table 5.2).

Importance of Fourier features – Figure 5.8 compares the top-down map obtained after querying the *Occupancy and Exploration Implicit Representation* trained with and without Fourier features. On each plot, the left and right maps respectively show the impact of using and not using Fourier features. Without the latter, no detail about the environment layout can be reconstructed. This corroborates findings also reported in other literature on implicit representations and coordinate networks, e.g. Mildenhall et al. (2020).

5.4 Conclusion

This chapter introduced two implicit representations to map semantic, occupancy and exploration information. The first estimates the position of an object of interest from a vector query, while the second encapsulates information about occupancy and explored area in the current environment. We also introduced a global read directly from the trainable weights of this representation. Our experiments showed that both implicit representations have a positive impact on the navigation performance of the agent. We also studied the scaling laws of the semantic representation and its behavior in the targeted lifelong learning problem.

The next chapter will also study the relationships between visual navigation and neural representations of 3D scenes. However, unlike in this chapter where we wondered how to use implicit representations as tools to use by navigating agents, we will next focus on a

different problem: how can neural agents navigate autonomously to collect training data for a high-quality neural representation of a scene to be computed in a second time?

Chapter

Training of Neural Implicit Representations through Autonomous Scene Exploration

Abstract

Implicit representations such as Neural Radiance Fields (NeRF) are very effective at novel view synthesis. However, these models typically require manual and careful human data collection for training. This chapter presents AutoNeRF, a method to collect data required to train NeRFs using autonomous embodied agents. The introduced method allows an agent to explore an unseen environment efficiently and use the experience to build an implicit map representation autonomously. We compare the impact of different exploration strategies including handcrafted frontier-based exploration, end-to-end and modular approaches composed of trained high-level planners and classical low-level path followers. We train these models with different reward functions tailored to this problem and evaluate the quality of the learned representations on four different downstream tasks: classical viewpoint rendering, map reconstruction, planning, and pose refinement. Empirical results show that NeRFs can be trained on actively collected data using just a single episode of experience in an unseen environment, and can be used for several downstream robotic tasks, and that modular trained exploration models outperform other classical and end-to-end baselines. Finally, we show that AutoNeRF can reconstruct large-scale scenes, and is thus a useful tool to perform scene-specific adaptation as the produced 3D environment models can be loaded into a simulator to fine-tune a policy of interest.

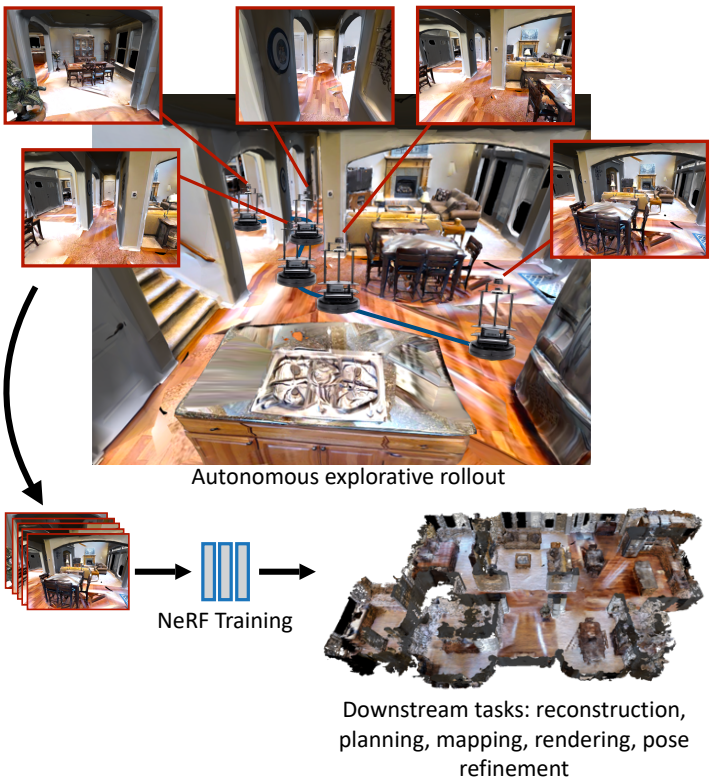


Figure 6.1: **AutoNeRF overview**: We propose a method for automatically generating 3D models of a scene by training NeRFs from data collected by autonomous agents. We compare classical and RL-trained exploration policies, with different reward definitions and evaluate the implicit representations on reconstruction, planning, mapping, rendering, and pose refinement.

6.1 Context

Exploration is a key challenge in building autonomous navigation agents that operate in unseen environments. As already presented in previous chapters, in the last few years, there has been a significant amount of work on training exploration policies to maximize coverage (Chaplot et al., 2020b; Chen et al., 2018; Savinov et al., 2018b), find goals specified by object categories (Gupta et al., 2017; Chaplot et al., 2020a; Ramakrishnan et al., 2022; Ramrakhya et al., 2022), images (Zhu et al., 2017b; Chaplot et al., 2020d; Hahn et al., 2021; Mezghani et al., 2022) or language (Anderson et al., 2018b; Krantz et al., 2020; Min et al., 2022) and for embodied active learning (Chaplot et al., 2020c, 2021). Among these methods, modular learning methods have shown to be very effective at various embodied tasks (Chaplot et al., 2020b,a; Deitke et al., 2022; Gervet et al., 2022). These methods learn an exploration policy that can build an explicit semantic map of the environment which is then used for planning and downstream embodied AI tasks such as *ObjNav* or *ImageNav*.

Concurrently, in the computer graphics and vision communities, there has been a recent but large body of work on learning implicit map representations, particularly based on Neural Radiance Fields (NeRF) (Mildenhall et al., 2020; Müller et al., 2022; Garbin et al., 2021; Yu

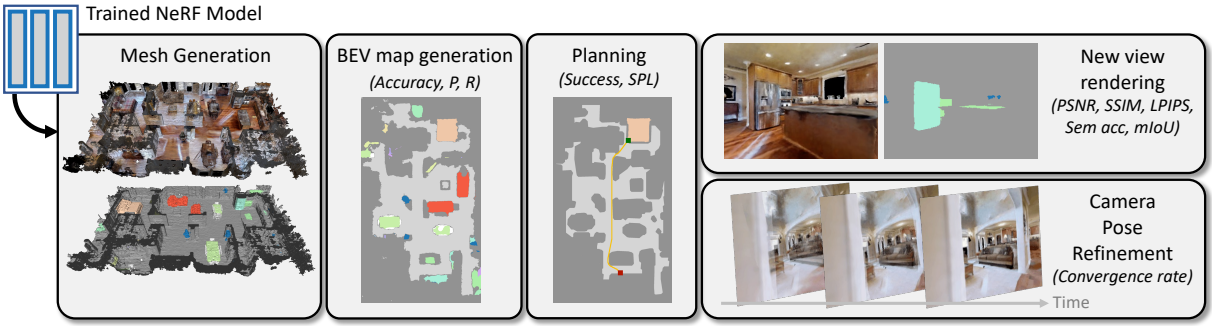


Figure 6.2: **Downstream tasks**: The model trained from autonomously collected data is used for several downstream tasks related to robotics: Mesh generation for the covered scene (color or semantic mesh); Birds-eye-view map generation and navigation/planning on this map; new view generation of RGB and semantic frames; camera pose refinement (visual servoing).

et al., 2021; Xie et al., 2022). Prior methods (Tancik et al., 2022; Vora et al., 2021; Zhi et al., 2021a,b) demonstrate strong performance in novel view synthesis and are appealing from a scene understanding point of view as a compact and continuous representation of appearance and semantics in a 3D scene. However, most approaches require training frames (RGB, depth, semantics) that should be collected manually. Can we train embodied agents to explore an unseen environment efficiently to collect data that can be used to create implicit map representations or NeRFs autonomously? In this chapter, our objective is to tackle this problem of active exploration for autonomous NeRF construction. If an embodied agent is able to build an implicit map representation autonomously, it can then use it for a variety of downstream tasks such as planning, pose estimation, and navigation. Just a single episode or a few minutes of exploration in an unseen environment can be sufficient to build an implicit representation that can be utilized for improving the performance of the agent in that environment for several tasks without any additional supervision.

This chapter introduces AutoNeRF, a modular policy trained with RL that can explore an unseen 3D scene to collect data for training a NeRF model autonomously (Figure 6.1). While most prior work evaluates NeRFs on rendering quality, we propose a range of downstream tasks to evaluate them (and indirectly, the exploration policies used to gather data for training these representations) for Embodied AI applications. Specifically, we use geometric and semantic map prediction accuracy, planning accuracy for *ObjNav* and *PointNav* and camera pose refinement (Figure 6.2). We show that AutoNeRF outperforms the well-known frontier-based exploration algorithm as well as state-of-the-art end-to-end learned policies, and also study the impact of different reward functions on the downstream performance of the NeRF model. We finally study how AutoNeRF can be used as a tool to autonomously adapt policies to a specific scene at deployment time by providing a high-quality reconstruction of large-scale environments that can be loaded into a simulator to improve the performance of any given agent safely.

The content of this chapter can thus be summarized as follows,

- AutoNeRF: a modular RL-based policy to autonomously collect NeRF training data in a new 3D environment.
- A study of different reward functions to favor the collection of relevant NeRF training data.
- A reflexion on how to evaluate the quality of learned neural fields in the context of Embodied AI, going further than novel view synthesis.
- Comparisons with frontier-based and end-to-end exploration baselines.
- A use-case showcasing the potential of the method to automatically scan a 3D scene and load it in a simulator in a second stage to finetune a policy of interest.

6.2 AutoNeRF

We present AutoNeRF, a method to collect data required to train NeRFs using autonomous embodied agents. In our task setup, the agent is initialized in an unseen environment and is tasked with gathering data in a single episode with a fixed time budget. The observations collected by the agent in this single trajectory are used to train a neural implicit representation of the scene, which will serve as a compact and continuous representation of the density, the RGB appearance, and the semantics of the considered environment. Finally, the trained scene model is evaluated on several downstream tasks in robotics: new view rendering, mapping, planning and pose refinement.

Task Specification – The agent is initialized at a random location in an unknown scene and at each timestep t can execute one of 3 discrete actions: *Move Forward 0.25m*, *Rotate Right* 30°, *Rotate Left* 30°. At each step, the agent receives an observation $\mathbf{o}_t$ composed of an egocentric RGB frame and a depth map. The field of view of the agent is 90°. It also has access to odometry information. The agent can navigate for a limited number of 1500 discrete steps.

AutoNeRF can be broken down into two phases: Exploration Policy Training and NeRF Training. In the first phase, we train an exploration policy to collect the observations. The policy is self-supervised, trained in a set of environments using intrinsic rewards. In the second phase, we use the trained exploration policy to collect data in unseen test scenes, one trajectory per scene, and train a NeRF model using this data. The trained NeRF model is then evaluated on the set of downstream tasks.

6.2.1 *Background*

We first briefly recall relevant background on modular exploration policies and neural radiance fields presented in chapters 2 and 3, but with more details and oriented toward our

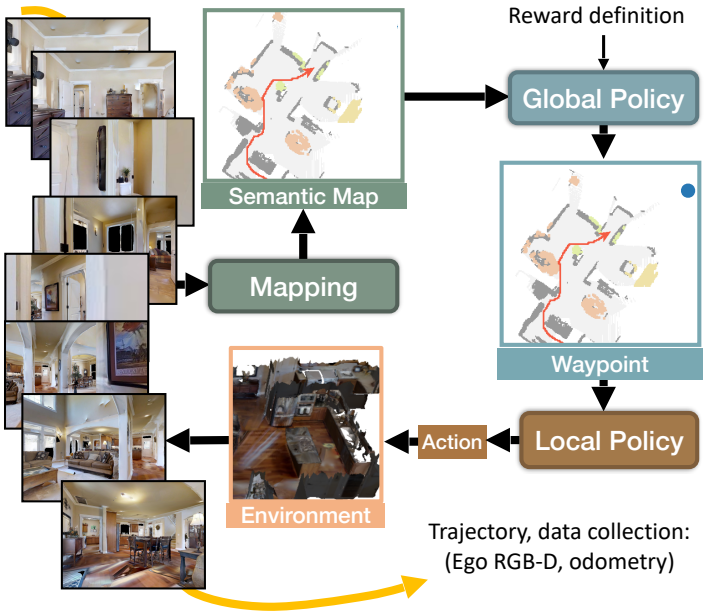


Figure 6.3: **Modular policy overview**: We adapted the modular policy from Chaplot et al. (2020a): a mapping module generates a semantic and occupancy top-down map from egocentric RGB-D observations and sensor pose. A high-level policy trained with RL predicts global waypoints, which are followed by a handcrafted low-level policy (Fast Marching Method). The sequence of observations comprises the data input to NeRF training.

proposed method.

Modular exploration policies

The trained policy aims to allow an agent to explore a 3D scene to collect a sequence of 2D RGB and semantic frames and camera poses, that will be used to train the continuous scene representation. Following Chaplot et al. (2020a,b), we adapt a modular policy composed of a *Mapping* process that builds a *Semantic Map*, a *Global Policy* that outputs a global waypoint from the semantic map as input, and finally, a *Local Policy* that navigates towards the global goal, see Figure 6.3.

Semantic Map – a 2D top-down map is maintained at each time step t, with several components: (i) an occupancy component $\mathbf{m}_t^{occ} \in \mathbb{R}^{M\times M}$ stores information on free navigable space; (ii) an exploration component $\mathbf{m}_t^{exp} \in \mathbb{R}^{M\times M}$ sets to 1 all cells which have been within the agent's field of view since the beginning of the episode; (iii) a semantic component $\mathbf{m}_t^{sem} \in \mathbb{R}^{S\times M\times M}$, where $M\times M$ is the spatial size and S denotes the number of channels storing information about the scene. Additional maps store the current and previous agent locations. All maps are updated at each timestep from sensor information. Structural components are updated by inverse projection of the current depth observation and pooling to the ground plane, a similar computation is done for the exploration component. The semantic maps additionally use predictions obtained with Mask R-CNN (He et al., 2017). Egocentric

maps are integrated over time taking into account agent poses estimated from sensor information.

Policies – intermediate waypoints are predicted by the *Global Policy*, a convolutional neural network taking as input the stacked maps (we use the architecture from Chaplot et al. (2020a)) and is trained with RL/PPO (Schulman et al., 2017). A *Local Policy* navigates towards the waypoint taking discrete actions for 25 steps with the analytical Fast Marching Method (Sethian, 1996).

Neural radiance fields

Vanilla Semantic NeRF – Neural Radiance Fields (Mildenhall et al., 2020) are composed of MLPs predicting the density σ, color $\mathbf{c}$ and, eventually as in Zhi et al. (2021a), the semantic class s of a particular $3D$ position in space $\mathbf{x} \in \mathbb{R}^3$, given a camera viewing direction $\mathbf{d} \in \mathbb{R}^3$. NeRFs have been designed to render new views of a scene provided a camera position and viewing direction. The color of a pixel is computed by performing an approximation of volumetric rendering, sampling N quadrature points along the ray. Given multiple images of a scene along with associated camera poses, a NeRF is trained with Stochastic Gradient Descent minimizing the difference between rendered and ground-truth images.

Enhanced NeRF (Semantic Nerfacto) – we leverage recent advances to train NeRF models faster while maintaining high rendering quality and follow what is done in the Nerfacto model from Tancik et al. (2023), that we augment with a semantic head. The inputs $\mathbf{x}$ and $\mathbf{d}$ are augmented with a learned appearance embedding $\mathbf{e} \in \mathbb{R}^{32}$. Both $\mathbf{x}$ and $\mathbf{d}$ are first encoded using respectively a hash encoding function h as $\tilde{\mathbf{x}} = h(\mathbf{x})$ and a spherical harmonics encoding function sh as $\tilde{\mathbf{d}} = sh(\mathbf{d})$. $\tilde{\mathbf{x}}$ is fed to an MLP f_d predicting the density at the given 3D position, yielding $(\sigma, \mathbf{h_d}) = f_d(\tilde{\mathbf{x}}; \Theta_d)$, where $\mathbf{h_d}$ is a latent representation. $\mathbf{h_d}$ is fed to another MLP model f_s that outputs a softmax distribution over the S considered semantic classes as $\mathbf{s} = f_s(\mathbf{h_d}; \Theta_s)$ where $\mathbf{s} \in \mathbb{R}^S$. Finally, $\mathbf{h_d}$, $\tilde{\mathbf{d}}$ and $\mathbf{e}$ are the inputs to f_c that predicts the RGB value at the given 3D location, $\mathbf{c} = f_c(\mathbf{h_d}, \tilde{\mathbf{d}}, \mathbf{e}; \Theta_c)$ where $\mathbf{c} \in \mathbb{R}^3$.

At training time, points must be sampled along shot rays to reconstruct image pixels. First, piecewise sampling will choose half of the points uniformly up to a distance d_s from the camera, the other half of the points are distributed with an increasing step size. In the second step, this initial set of samples is improved by proposal sampling, which consolidates samples in regions that have the most impact on the final rendering. This requires a fast query and coarse density representation, different from the neural field itself, implemented as a small MLP with hash encoding.

6.2.2 *Exploration Policy Training*

As described previously, we use a modular exploration policy architecture with the *Global Policy* primarily responsible for exploration. We consider different reward signals for training the *Global Policy* tailored to our task of scene reconstruction, and which differ in the importance they give to different aspects of the scene. All these signals are computed in a self-supervised fashion using the metric map representations built by the exploration policy.

Explored area — (*Ours (cov.)*) optimizes the coverage of the scene, i.e. the size of the explored area, and has been proposed in the literature, e.g. in Chaplot et al. (2020a,b). It accumulates differences in the exploration component $\mathbf{m}_t^{exp}$,

$$r_t^{cov} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \mathbf{m}_t^{exp}[i,j] - \mathbf{m}_{t-1}^{exp}[i,j]$$

Obstacle coverage — (*Ours (obs.)*) optimizes the coverage of obstacles in the scene, and accumulates differences in the corresponding component $\mathbf{m}_{t-1}^{occ}[i,j]$. It targets tasks where obstacles are considered more important than navigable floor space, which is arguably the case when viewing is less important than navigating.

$$r_t^{obs} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \mathbf{m}_t^{occ}[i,j] - \mathbf{m}_{t-1}^{occ}[i,j]$$

Semantic object coverage — (*Ours (sem.)*) optimizes the coverage of the S semantic classes detected and segmented in the semantic metric map $\mathbf{m}_t^{sem}$. This reward removes obstacles that are not explicitly identified as a notable semantic class — see section 6.3 for their definition.

$$r_t^{sem} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \sum_{k=0}^{S-1} \mathbf{m}_t^{sem}[i,j,k] - \mathbf{m}_{t-1}^{sem}[i,j,k]$$

Viewpoints coverage — (*Ours (view.)*) optimizes for the usage of the trained implicit representation as a dense and continuous representation of the scene usable to render arbitrary new viewpoints, either for later visualization as its own downstream task or for training new agents in simulation. To this end, we propose to maximize coverage not only in terms of agent positions but also in terms of agent viewpoints. Compared to Chaplot et al. (2020a), we introduce an additional 3D map $\mathbf{m}^{view}[i,j,k]$, where the first two dimensions correspond to spatial 2D positions in the scene and the third dimension corresponds to a floor plane angle of the given cell discretized into V=12 bins. A value of $\mathbf{m}_t^{view}[i,j,k] = 1$ indicates that cell (i,j) has been seen by the agent from a (discretized) angle k. The reward maximizes its changes,

$$r_t^{view} = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} \sum_{k=0}^{V-1} \mathbf{m}_t^{view}[i,j,k] - \mathbf{m}_{t-1}^{view}[i,j,k]$$

6.2.3 *NeRF training*

The sequence of observations collected by the agent comprises egocentric RGB frames $\{\mathbf{o}_t\}_{t=1...T}$, first-person semantic segmentations $\{\mathbf{s}_t\}_{t=1...T}$ and associated poses $\{\mathbf{p}_t\}_{t=1...T}$ in

a reference frame, which we define as the starting position t=0 of each episode. In our experiments, we leverage privileged pose and semantics information from simulation. We also conduct an experiment showcasing the difference between using ground-truth semantics from a simulator and a Mask R-CNN (He et al., 2017) model. As in Zhi et al. (2021a), we add a semantic head to the implicit representation, which predicts the semantic classes defined in the segmentation maps $\{\mathbf{s}_t\}$.

An important property of this procedure is that no depth information is required for reconstruction. The implicit representation is trained by mapping pixel coordinates $\mathbf{x}_i$ for each pixel i to RGB values and semantic values with the volume rendering loss described in Section 6.2.1. The input coordinates $\mathbf{x}_i$ are obtained using the global poses $\mathbf{p}_t$ and intrinsics from calibrated cameras.

6.2.4 *Downstream tasks*

Prior work on implicit representations generally focused on two different settings: (i) evaluating the quality of a neural field based on its new view rendering abilities given a dataset of (carefully selected) training views, and (ii) evaluating the quality of a scene representation in robotics conditioned on given (constant) trajectories, evaluated as reconstruction accuracy. We cast this task in a more holistic way and more aligned with our scene understanding objective. We evaluate the impact of trajectory generation (through exploration policies) directly on the quality of the representation, which we evaluate in a goal-oriented way through multiple tasks related to robotics (cf. Figure 6.2).

Task 1: Rendering – This task is the closest to the evaluation methodology prevalent in the neural field literature. We evaluate the rendering of RGB frames and semantic segmentation as proposed by Zhi et al. (2021a). Unlike the common method of evaluating an implicit representation on a subset of frames within a collected trajectory, we evaluate it on a set of uniformly sampled camera poses within the scene, independently of the trajectory taken by the policy used to collect training data. This allows us to evaluate the representation of the complete scene and not just the ability of the NeRF model to interpolate between close poses within the training trajectory.

We render ground-truth images and semantic masks associated with sampled camera poses using the Habitat simulator (Savva et al., 2019; Szot et al., 2021) and compare them against the NeRF rendering. RGB rendering metrics are PSNR (*Peak Signal-to-Noise Ratio*), SSIM (*Structural Similarity Index Measure*) and LPIPS (Zhang et al., 2018). PSNR estimates absolute errors while SSIM evaluates the amount of retrieved structural information in the image by incorporating priors such as pixel inter-dependencies, and LPIPS attempts to reflect human perception by computing distance between ground-truth and rendered frames in the feature space of a VGG network (Simonyan and Zisserman, 2014). Rendering of semantics is evaluated in terms of average per-class accuracy and mean intersection over union (mIoU).

Task 2: Metric Map Estimation – While rendering quality is linked to the perception of the scene, it is not necessarily a good indicator of its structural content, which is crucial for robotic downstream tasks. We evaluate the quality of the estimated structure by translating the continuous representation into a format, which is very widely used in map-and-plan baselines for navigation, a binary top-down (bird's-eye-view=BEV) map storing occupancy and semantic category information and compare it with the ground-truth from the simulator. We evaluate obstacle and semantic maps using accuracy, precision, and recall.

Cells in the occupancy and semantic top-down maps generated from NeRF models are of size $1cm \times 1cm$. In order to create the occupancy map, we first compute a 3D voxel grid by regularly querying the NeRF density head between scene bounds along x and z axes, and between 0 and the agent's height along the vertical y axis. We then transform the 3D grid into a 2D top-down map by applying a sum operation along the vertical axis. We found that, when using a *Semantic Nerfacto* model, generating a point cloud where each point is associated with a semantic class from the train camera poses works best when generating the semantic top-down map. The point cloud is converted into a 3D voxel grid, where each cell is associated with a channel for each semantic class. A per-class sum operation can finally transform the 3D grid into a 2D map with one channel per class. The same cell resolution is used when generating ground-truth maps from the Habitat simulator.

Task 3: Planning – Using maps for navigation, it is difficult to pinpoint the exact precision required for successful planning, as certain artifacts and noises might not have a strong impact on reconstruction metrics, but could lead to navigation problems, and vice-versa. We perform goal-oriented evaluation and measure to what extent path planning can be done on the obtained top-down maps.

We sample N_p ($= 100$) points on each scene and plan from those starting points to two different types of goals: to selected end positions, *PointNav* planning, and to objects categories, *ObjNav* planning. The latter, *ObjNav*, requires planning the shortest path from the given starting point to the closest object of each semantic class available on the given scene. For a given episode i, *Success* S_i is 1 if the last cell in the planned path is closer than 1m to the goal and if less than 10 planned cells are obstacles, otherwise it is 0. We chose to allow up to 10 obstacle cells on the planned path to keep the task from being overly complex and thus uninformative (as each cell is rather small, i.e. $1cm \times 1cm$). For both *PointNav planning* and *ObjNav planning*, we plan with the Fast Marching Method and report mean *Success* and *SPL*.

Resolution of the top-down maps used to plan a path are the same as for the *Metric Map Estimation* task. Once generated, the path is evaluated on the ground-truth top-down map from the Habitat simulator. In order to account for the size of a potential robot of radius $18cm$, obstacles on the ground-truth map are dilated. We also apply a dilation with a $20cm$ radius to obstacles on our top-down map before planning the path.

Task 4: Pose Refinement – This task (Yen-Chen et al., 2021) involves correcting an initial noisy camera position and associated rendered view and optimizing the position until a given ground-truth position is reached, which is given through its associated rendered view only. The optimization process therefore leads to a trajectory in camera pose space. This task is closely linked to visual servoing with a "eye-in-hand" configuration, a standard problem in robotics, in particular in its "direct" variant (Marchand, 2020), where the optimization is directly performed over losses on the observed pixel space.

We address this problem by taking the trained NeRF model f and freezing its weights θ. In what follows, we will denote the function rendering a full image $\mathbf{o}$ given camera pose c and viewing direction d as $\mathbf{o} = \mathcal{R}(c, d)$. Then, given a ground truth view $\mathbf{o}^*$, the camera position and direction can be directly optimized from a starting position $(c, d)^{[0]}$ with gradient descent as

$$(c, d)^{[t+1]} = (c, d)^{[t]} + \nu \left[\frac{\partial \mathcal{L}(\mathbf{o}^*, \mathcal{R}(c, d))}{\partial c, d} \right],$$

where $\mathcal{L}$ is the Mean Squared Error loss and ν is a learning rate.

To generate episodes of starting and end positions, we take N_c (= 100) sampled camera poses in each scene and apply a random transformation to generate noisy poses. The model is evaluated in terms of rotation and translation convergence rate, i.e. percentage of samples where the final difference with ground truth is less than 3° in rotation and 2cm in translation. We also report the mean translation and rotation errors for the converged samples.

At each pose refinement optimization step, we would ideally want to render the full image associated with the estimated camera pose to compare with the RGB frame from the camera. As already noticed in previous work (Yen-Chen et al., 2021), doing this is expensive, and we thus instead randomly sample pixels to be rendered within the image at each optimization step.

Task 5: Mesh generation – In order to create a mesh from a trained NeRF model, we first build a 3D voxel grid by querying the implicit representation regularly on a grid between the scene bounds. Each voxel will be associated with a density, and either a color or a semantic class depending on the nature of the mesh (color or semantic mesh) to generate. The voxel grid is converted into a mesh by applying the Marching Cubes algorithm (Lorensen and Cline, 1987).

6.3 Experimental results

Modular Policy training – is performed on one V100 GPU for 7 days. All modular policies are trained on the 25 scenes of the Gibson (Xia et al., 2018)-tiny training set. The used Mask R-CNN model is pre-trained on the MS-COCO dataset (Lin et al., 2014) and finetuned on Gibson train scenes. We consider S=15 semantic categories: {*chair*, *couch*, *potted plant*, *bed*,

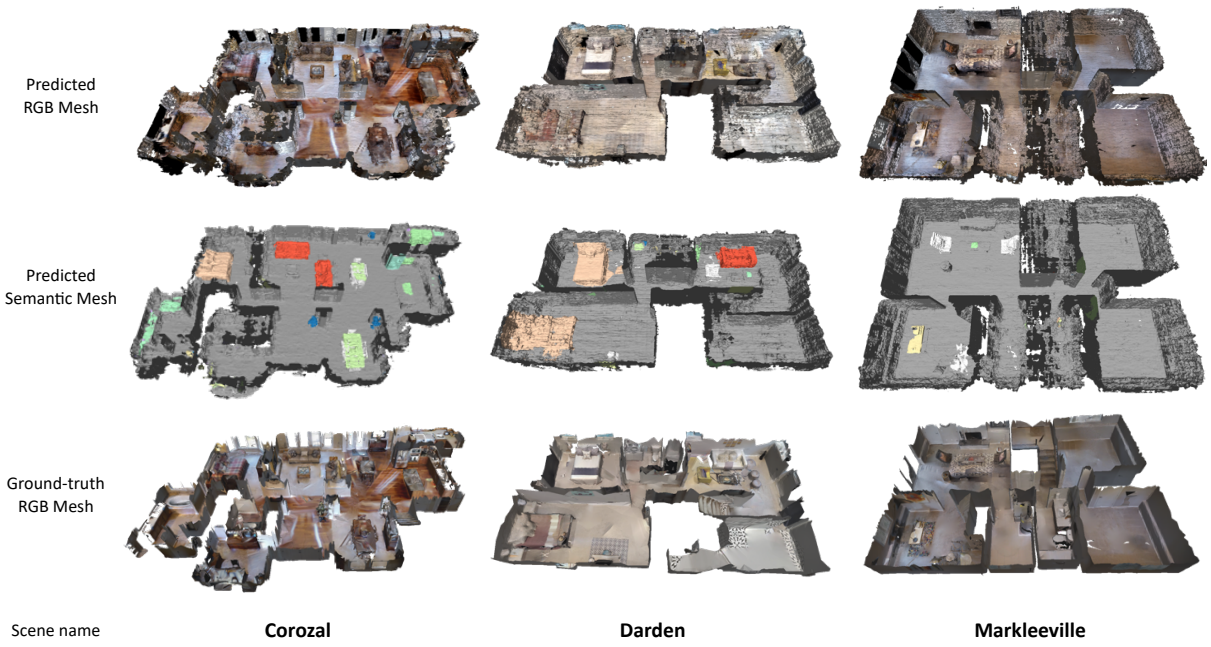


Figure 6.4: **Mesh reconstruction**: Reconstruction of 3 Gibson val scenes extracted from a NeRF model trained on data gathered by our modular policy. Both geometry, semantics, and appearance are satisfying. Exploration with the modular policy, *Ours (obs)*.

toilet, *tv*, *dining table*, *oven*, *sink*, *refrigerator*, *book*, *clock*, *vase*, *cup*, *bottle*}.

External baselines – We first compare our trained modular policies against a *Frontier* baseline that is similar in terms of mapping and local navigation, but only replaces the *Global Policy* with a classical Frontier-Based Exploration algorithm (Yamauchi, 1997) that maintains a 2D frontier computed from the exploration channel of the map (same map used by modular policies: the only difference is the selection of global goals from the frontier instead of predicting them with a neural-based *Global Policy*). Then, we also compare against end-to-end policies trained with RL. More specifically, we consider 4 end-to-end policies from Ramakrishnan et al. (2021b), that all share the same architecture but were trained with different exploration-related reward functions: coverage (*E2E (cov.)*), curiosity (*E2E (cur.)*), novelty (*E2E (nov.)*), reconstruction (*E2E (rec.)*). Reward functions are presented by Ramakrishnan et al. (2021b).

Evaluation – consists in running 5 rollouts with different start positions in each of the 5 Gibson-tiny val scenes for each policy, always on the first house floor. A NeRF model is then trained on each trajectory data.

NeRF models – In our experiments, we consider two different NeRF variants presented in Section 6.2.1. Most experiments are conducted with *Semantic Nerfacto*, as it provides a great trade-off between training speed and quality of representation. *Semantic Nerfacto* is built on top of the *Nerfacto* model from the nerfstudio library (Tancik et al., 2023). We augment the model with a semantic head and implement evaluation on test camera poses independently of the collected trajectory. Only the next two subsections (6.3.1, 6.3.3) will involve training a

Figure 6.5: **Navigating in the Habitat simulator**: The underlying mesh was extracted from the trained NeRF, *Ours (cov)*. Rendering quality and the generated BEV map are correct, as are free navigable space and collision handling. Temporal order indicated by →.

Mesh	Success	SPL
Original Gibson mesh	88.1	73.3
Rollout → NeRF training → Mesh generation	78.4	67.7

Table 6.1: **Navigating inside a NeRF-generated mesh**: Average *PointNav* performance of a policy planning a path to the goal with the Fast Marching Method and taking discrete actions on 4 Gibson val scenes in the Habitat simulator, from either the original mesh (**Gibson**) or the mesh extracted from the NeRF model (**Rollout + NeRF train. + Mesh gen.**) trained from autonomously collected data by *Ours (obs.)*. Our reconstruction does not require depth data.

vanilla Semantic NeRF model, more precisely the one introduced by Zhi et al. (2021a) that also contains a semantic head. We chose this variant for these specific experiments to illustrate the possibility of providing high-fidelity representations of complex scenes as a *vanilla Semantic NeRF* model trained for a longer time (12h) leads to better-estimated geometry. Results from *Semantic Nerfacto* are still very good (see Figures 6.7 and 6.8) but we found meshes to be of higher quality with a vanilla NeRF model.

6.3.1 *Reconstructing house-scale scenes*

We illustrate the possibility of autonomously reconstructing complex large-scale environments such as apartments or houses from the continuous representations trained on data collected by agents exploring the scene using the modular policy. Figure 6.4 shows RGB and semantic meshes for 3 Gibson val scenes. Geometry, appearance, and semantics are satisfying. In Figure 6.5 we show that such meshes can be loaded into the Habitat simulator and allow proper navigation and collision computations. Both occupancy top-down map generation and RGB renderings are performed by the Habitat simulator from the generated mesh.

Policy	Success	SPL
Finetuned on Gibson meshes (not comparable)†	99.7	97.9
Pre-trained (no finetuning)	90.2	82.9
Finetuned on AutoNeRF meshes	**92.9**	**86.7**

Table 6.2: **PointNav Finetuning**: Finetuning a PointNav agent on a mesh automatically collected from a rollout and a NeRF with AutoNeRF improves mean performance over a pre-trained policy on a specific scene. † an upper bound which finetunes on the original mesh. In a real use case involving a robot automatically collecting data, this mesh would not be available (not comparable).

6.3.2 *Navigating inside a NeRF-generated mesh*

To further evaluate the quality of the geometry learned by an autonomously generated NeRF and assess to what extent it can be used inside a simulator, we perform *PointNav* navigation on an original mesh and a NeRF-generated one for 4 Gibson val scenes. The considered agent is based on the introduced modular policy, restricted to the planning part: the output of the *Global Policy* is replaced with the input *PointNav* vector. Planning is performed using the Fast Marching Method, relying on the map channels storing information about obstacles and explored area. Table 6.1 shows that there is a performance drop between the navigation on the original Gibson meshes and the reconstructed ones, but *Success* and *SPL* are close. The performance on the original mesh is a "soft upper bound", as data was collected from navigating in this original mesh, before training a NeRF and finally generating a new mesh representation. These results show that our NeRF-generated mesh features a satisfying geometry, allowing to navigate properly when loaded within the Habitat simulator. Some further mesh post-processing could be needed, along with additional work on improving lighting within the simulator.

6.3.3 *Autonomous adaptation to a new scene*

A long-term goal of Embodied AI is to train general policies that can be deployed on any new scene and perform a task of interest. Even if some policies already showcase strong generalization abilities, they will likely struggle with some specificities of a given environment. When considering the deployment of trained agents on real robots in a house or apartment, such failure modes can be problematic. A scene-specific adaptation of a pre-trained policy thus appears as a relevant alternative to learn about a new environment. However, such adaptation should happen in simulation to ensure safety. We here explore the usage of AutoNeRF to first explore the environment to build a 3D representation, which is then loaded into a simulator to safely finetune a policy of interest. More specifically, we consider a policy for an agent taking a depth frame as input at each timestep t and pre-train it on *PointNav* navigation on the Gibson train scenes. It is then fine-tuned on 4 Gibson val scenes, using meshes generated with AutoNeRF, before being evaluated on the original Gibson meshes. Sampling of training

	RGB			Semantics	
Policy	PSNR	SSIM	LPIPS (↓)	Per-class acc.	mIoU
Frontier	19.75	0.743	0.343	81.4	65.7
E2E (cov.)	20.94	0.750	0.332	80.1	63.9
E2E (cur.)	20.60	0.747	0.338	78.7	61.9
E2E (nov.)	23.36	0.801	0.268	84.6	71.4
E2E (rec.)	23.17	0.797	0.270	84.1	70.5
Ours (cov.)	24.89	0.837	0.218	90.2	81.2
Ours (sem.)	25.34	0.843	0.207	**91.9**	81.8
Ours (obs.)	**25.56**	**0.846**	**0.203**	**91.8**	**83.2**
Ours (view.)	25.17	0.842	0.211	91.3	82.0

Table 6.3: **Rendering performance** on uniformed sampled viewpoints of the full scene after training on a single trajectory.

and validation episodes, as well as training and validation, are performed in the Habitat simulator. For each environment, we sample $50k$ episodes from our mesh to finetune the policy for $10M$ training frames using PPO with a learning rate of 2.5e$-$6. For evaluation, we sample $1k$ episodes per scene on the original Gibson meshes and report mean Success and SPL. Table 6.2 shows that the pre-trained policy already achieves great performance. However, some validation episodes fail because of certain specific scene configurations. Scene-specific finetuning on autonomously reconstructed 3D meshes allows to improve both *Success* and *SPL* compared with the pre-trained policy.

We also compare with finetuning directly on the Gibson mesh (for $10M$ frames with a learning rate of 2.5e$-$5), which provides a non-comparable soft upper bound — in a real robotic scenario, these meshes would not be accessible. We can see that performance could still be improved further by increasing mesh reconstruction quality. However, it is important to note that the mistakes from the pre-trained policy occur in very specific places in the environment, and thus reaching the performance of the upper bound might be about reconstructing fine details.

6.3.4 *Quantitative results*

Frontier-Based Exploration vs Modular Policy – as can be seen from the quantitative comparisons on the different downstream tasks (Tables 6.3, 6.4, 6.5, 6.6), RL-trained modular policies outperform Frontier-based Exploration on all metrics and should thus be considered as the preferred means of collecting NeRF data. This is a somewhat surprising result, since Frontier-Based Exploration generally performs satisfying visual coverage of the scene, even though it can sometimes get stuck because of map inaccuracies. This shows that vanilla visual coverage, the optimized metrics in many exploration-oriented tasks, is not a sufficient criterion to

	Occupancy			Semantics		
Policy	Acc.	Prec.	Rec.	Acc.	Prec.	Rec.
Frontier	81.2	86.9	49.9	99.7	26.6	21.0
E2E (cov.)	77.1	86.2	50.4	99.7	22.1	16.1
E2E (cur.)	81.8	90.3	50.7	99.7	19.2	12.5
E2E (nov.)	83.1	88.7	61.3	99.7	25.5	18.3
E2E (rec.)	81.6	87.6	60.0	99.7	26.2	18.0
Ours (cov.)	86.8	89.1	74.7	99.8	35.1	27.1
Ours (sem.)	86.6	88.3	76.5	99.8	35.7	29.8
Ours (obs.)	86.4	89.4	76.5	99.8	36.2	29.8
Ours (view.)	**88.1**	**90.9**	**77.0**	99.8	**37.4**	**30.2**

Table 6.4: **Map estimation performance**: Comparison of BEV maps estimated from the NeRF.

	PointNav		ObjNav	
Policy	Success	SPL	Success	SPL
Frontier	22.4	21.4	9.6	9.1
E2E (cov.)	30.0	29.3	8.9	8.3
E2E (cur.)	29.8	29.2	8.5	8.0
E2E (nov.)	32.3	31.9	11.4	10.8
E2E (rec.)	32.8	32.6	10.5	10.0
Ours (cov.)	**39.5**	**39.0**	14.8	14.3
Ours (sem.)	37.7	37.4	**16.0**	**15.4**
Ours (obs.)	38.2	37.8	**15.8**	**15.3**
Ours (view.)	39.0	38.6	**15.9**	**15.3**

Table 6.5: **Planning performance** on the BEV maps estimated from the NeRF obtained with the *Fast Marching* method.

collect data for NeRF training. Figure 6.6 illustrates this point with rollouts from FBE and a modular policy trained to maximize obstacle coverage. FBE properly covers the scene but does not necessarily cover a large diversity of viewpoints, while the modular policy provides richer training data to the NeRF.

End-to-end Policy vs Modular Policy – Tables 6.3, 6.4, 6.5, 6.6 also show that the modular policies outperform end-to-end RL policies on all considered metrics. Interestingly, *novelty* and *reconstruction* seem to be the best reward functions from Ramakrishnan et al. (2021b) when training end-to-end policies if the final goal is to autonomously collect data to build a NeRF model.

Comparing trained policies – Rewarding modular policies with obstacles (*Ours (obs.)*) and viewpoints (*Ours (view.)*) coverage appears to lead to the best overall performance when we

Policy	Conv. rate	Rot. Error (°) (↓)	Trans. Error (m) (↓)
Frontier	7.2	0.383	0.00955
E2E (cov.)	15.4	0.319	0.00775
E2E (cur.)	12.5	0.325	0.00799
E2E (nov.)	19.4	0.315	0.00774
E2E (rec.)	19.3	0.292	0.00734
Ours (cov.)	20.2	**0.283**	**0.00734**
Ours (sem.)	**23.0**	0.319	0.00784
Ours (obs.)	22.5	0.305	0.00765
Ours (view.)	21.1	0.316	0.00769

Table 6.6: **Pose refinement performance**: Optimizing camera viewpoints given a rendered target viewpoint.

Figure 6.6: **Rollouts by Frontier-Based Exploration vs. Modular policy (obs cov)**: FBE properly covers the scene, but does not collect a large diversity of viewpoints, while the modular policy provides richer training data for the neural field.

consider the different metrics. Explored area coverage (*Ours (cov.)*) leads to highest *PointNav* performance, corroborating its importance for geometric tasks, whereas other semantic reward functions lead to higher *ObjectNav* performance, again corroborating its importance for semantic understanding of the scene.

Semantics from Mask R-CNN – Table 6.7 shows the impact of using Mask R-CNN to compute the semantics training data of the NeRF model compared with semantics from simulation. As expected, performance drops because Mask R-CNN provides a much noisier training signal, which could partly be explained by the visual domain gap between the real world and simulators. However, performance on the different downstream tasks is still reasonable, showing that one could autonomously collect data and generate semantics training signal without requiring additional annotation.

Task	Metrics	Sim.	Mask R-CNN
Rendering	Per-class acc.	91.8	65.4
	mIoU	83.2	61.1
Map comparison	Sem acc.	99.8	99.7
	Sem prec.	36.2	14.1
	Sem rec.	29.8	8.5
Planning	ObjNav Success	15.8	6.8
	ObjNav SPL	15.3	6.5

Table 6.7: **NeRF semantic maps**: Impact of the choice of ground-truth semantics vs. semantics estimated by Mask R-CNN when data is collected by *Ours (obs.)*.

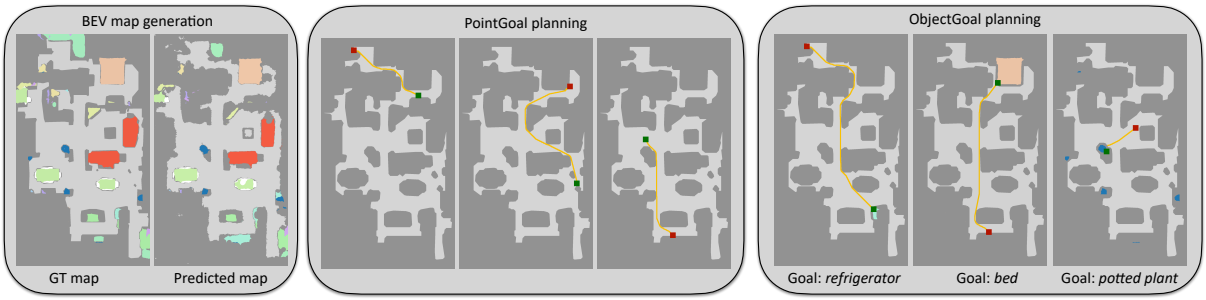


Figure 6.7: **BEV map tasks**: Generation of semantic BEV maps (Left), *PointNav* (Middle) and *ObjNav* planning (Right).

6.3.5 *Qualitative results*

BEV maps – Figure 6.7 gives examples of the BEV maps generated from the continuous representation: structural details and dense semantic information are nicely recovered (Left). Planned trajectories are close to the shortest paths, for both *PointNav* tasks (Middle) and *ObjNav* (Right).

Semantic rendering – Figure 6.8 compares the segmentation maps and RGB frames rendered with the continuous representation (trained with semantic masks from simulation) to the ground-truth maps from the simulator. Again, the structure of the objects and even fine details are well recovered, and only local noise is visible in certain areas. The semantic reconstruction is satisfying.

6.3.6 *Navigating with sensor and actuation noise*

All presented experiments were conducted following task specifications from Chaplot et al. (2020c), among which are perfect odometry information and actuation. We thus conduct an additional experiment to evaluate the impact of sensor and actuation noise on AutoNeRF. We add noise using realistic models from Chaplot et al. (2020b) and correct odometry information using the pose estimation module trained by Chaplot et al. (2020b). This allows our modular

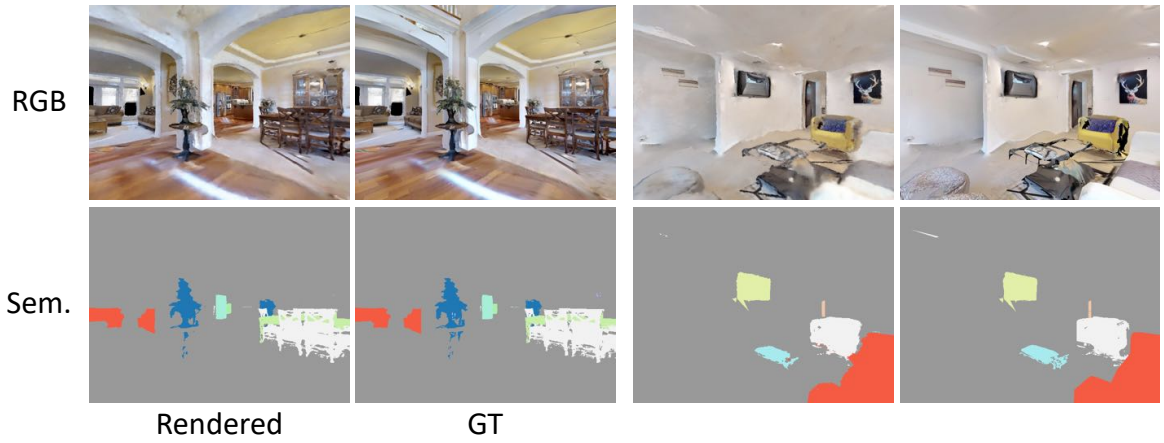


Figure 6.8: **Quality of rendering (RGB and semantic)**: RGB and semantic renderings on different scenes, compared with ground-truth (GT) from simulation. Data collected by *Ours (obs)*.

policy (*Ours (obs.)*) to explore properly environments and collect NeRF training data. We then refine camera poses with bundle adjustment before using them to train NeRF models. Sub-tables from Table 6.8 show that performance obviously decreases when adding noise, but we still reach a satisfying performance in all metrics. It is also important to note that, even after using the pose estimation module and post-processing bundle adjustment, camera poses spanning such large scenes are still noisy, and training NeRF models on noisy poses is considered a challenging problem in the literature Truong et al. (2023b). Better performance might thus come from new techniques to make NeRF models more robust to camera pose noise, which is orthogonal to the focus of this chapter.

6.4 Conclusion

This chapter introduced a task involving navigating in a 3D environment to collect NeRF training data. We showed that RL-trained modular policies outperform classic frontier-based exploration as well as other end-to-end RL baselines on this task, and compared different training reward functions. We also suggested evaluating NeRFs from a scene-understanding point of view and with robotics-oriented tasks: BEV map generation, planning, rendering, and camera pose refinement. Finally, we showed that it is possible with the considered method to reconstruct house-scale scenes, allowing to automatically scan a new environment to safely finetune another policy of interest safely in simulation.

The policy considered in this chapter tackles a single task, i.e. scene exploration, and is modular. In the next chapter, we will study the adaptation of visual features to allow a single end-to-end neural policy to target different tasks of interests. Unlike in this chapter and the two previous ones, we will not consider generalizing a task-specific policy to different environments, but instead tackle the problem of generalization to different visuomotor control tasks, even new ones unseen at training time.

		RGB			Semantics	
Policy	**Noise**	PSNR	SSIM	LPIPS (↓)	Per-class acc	mIoU
Ours (obs.)	–	25.56	0.846	0.203	91.8	83.2
Ours (obs.)	✓	20.87	0.761	0.264	85.4	73.6

(a) Rendering performance

		Occupancy			Semantics		
Policy	**Noise**	Acc.	Prec.	Rec.	Acc	Prec.	Rec.
Ours (obs.)	–	86.4	89.4	76.5	99.8	36.2	29.8
Ours (obs.)	✓	86.8	89.8	69.6	99.7	29.9	24.1

(b) Map estimation performance

		PointNav		ObjNav	
Policy	**Noise**	Success	SPL	Success	SPL
Ours (obs.)	–	38.2	37.8	15.8	15.3
Ours (obs.)	✓	34.5	33.8	12.9	12.4

(c) Planning performance

Policy	**Noise**	Conv. rate	Rot. Error (°) (↓)	Trans. Error (m)(↓)
Ours (obs.)	–	22.5	0.305	0.00765
Ours (obs.)	✓	7.9	0.405	0.01125

(d) Pose refinement performance

Table 6.8: **Navigating with sensor and actuation noise** – NeRF training data is collected by *Ours (obs)*.

Chapter

Efficiently adapting visual features to multiple tasks

Abstract

Successfully addressing a wide variety of tasks is a core ability of autonomous agents, requiring flexibly adapting the underlying decision-making strategies and, as will be presented in this chapter, also adapting the perception modules. An analogical argument would be the human visual system, which uses top-down signals to focus attention determined by the current task. Similarly, we adapt pre-trained large vision models conditioned on specific downstream tasks in the context of multi-task policy learning. We introduce task-conditioned adapters that do not require finetuning any pre-trained weights, combined with a single policy trained with BC and capable of addressing multiple tasks. We condition the visual adapters on task embeddings, which can be selected at inference if the task is known, or alternatively inferred from a set of example demonstrations. To this end, we propose an optimization-based estimator. We evaluate the method on a wide variety of tasks from the *CortexBench* benchmark and show that, compared to existing work, it can be addressed with a single policy. In particular, we demonstrate that adapting visual features is a key design choice and that the method generalizes to unseen tasks given a few demonstrations.

7.1 Context

Vision is one of the most important modalities for agents interacting with the world and is almost indispensable for dexterous manipulation or locomotion, as no other sensor can provide information as rich and versatile. The inherent flexibility of the sensor comes with a price, the high dimensionality of the information, and the complexity of the processes necessary to extract useful information. Humans and other biological agents are capable of adapting their perception systems to the task at hand. There is indeed evidence for bottom-up and top-down processes in human vision, with the latter guiding attention to regions determined by the requirements of the task (Corbetta and Shulman, 2002; Buschman and Miller, 2007).

There is a growing need for a similar versatility in artificial systems, and general neural networks have been trained from large-scale data in different domains such as NLP, CV, and, more recently, robotics. A single general vision model coupled with a neural policy would be an appealing choice if it could allow an easy generalization to new domains or tasks. The wide adoption of attention mechanisms in several domains has made it easier for trained models to adapt their behavior to the requirements of different tasks without changing parameters, and it has been shown that attention plays a crucial role in specialization on a specific instance in language models (Voita et al., 2019) and vision and language models (Kervadec et al., 2021). However, even powerful generally pre-trained models can benefit from parameter adaptations to specific tasks, either through fine-tuning (Hu et al., 2022) or by adding additional trained adapter layers to a frozen model (Chen et al., 2022a).

In robotics, prior work on the generalization capabilities of agents has focused on large-scale end-to-end training (Reed et al., 2022) or, targeting vision specifically, on pre-trained visual models required to generalize to various *different* policies (Majumdar et al., 2023; Ma et al., 2022; Nair et al., 2023). In this chapter, we show that a single policy can be trained for different tasks including manipulation, locomotion, and that the adaptation of visual features is highly beneficial, beyond the inherent adaptation capabilities of attention-based models. In particular, different tasks require diverse types of invariance and symmetries. While in principle it should be possible to learn to disentangle a sufficiently wide set of factors of variation in a captured representation such that it optimally performs on a wide variety of tasks, we will show that this is not the case for an arguably dominant pre-training method, masked auto-encoding (MAE – He et al. (2022)).

We propose task-conditioned adaptation, which allows leveraging the high-quality representations of generally pre-trained large vision models, while keeping the required flexibility to address a wide variety of tasks, and also new (unseen) tasks. We introduce a set of task-conditioned visual adapters that can be inserted inside a pre-trained visual Transformer-based backbone. The task is characterized by an embedding space, which is learned from supervision during training. We show that this embedding space captures regularities of

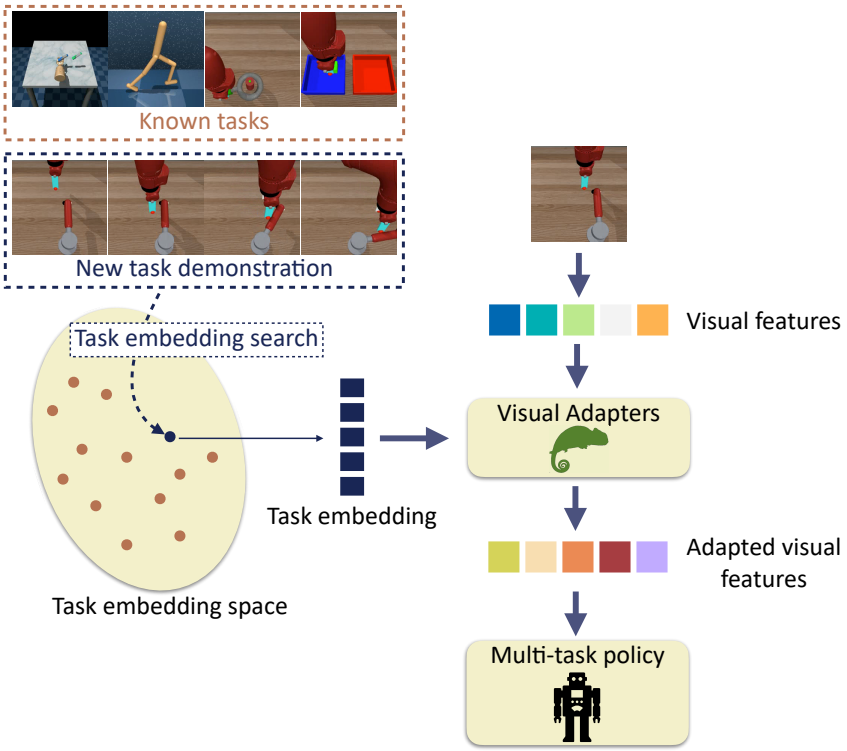


Figure 7.1: **Task-conditioned adaptation**: A single policy can be trained to address multiple heterogeneous tasks including manipulation, legged motion etc., and few-shot learning is possible to address tasks given as demonstrations but unseen during training. A key element is the task-conditioned adaptation of visual features.

tasks and demonstrate this with *few-shot capabilities*: the single policy and (adapted) visual representations can address new unseen tasks, whose embedding is estimated from a few demonstrations (cf. Figure 7.1).

We can summarize what will be presented in this chapter as:

- Task-conditioned visual adapters to flexibly modulate visual features to a specific task.
- A single multi-task policy solving tasks with different embodiments and environments.
- A task embedding optimization procedure based on a few demonstrations of a new task (unseen at training time) to adapt the model in a few-shot manner without any weight fine-tuning.
- Quantitative and qualitative results assessing the gain brought by the different novelties.

7.2 Task-conditioned adaptation

All tasks considered in this work are sequential decision-making problems, where at each discrete timestep t an agent receives the last 3 visual frames as an observation $\mathbf{v}_t \in \mathbb{R}^{3\times h\times w\times 3}$, where h and w are the height and width of images, and a proprioception input $\mathbf{p}_t \in \mathbb{R}^{d_a}$, and

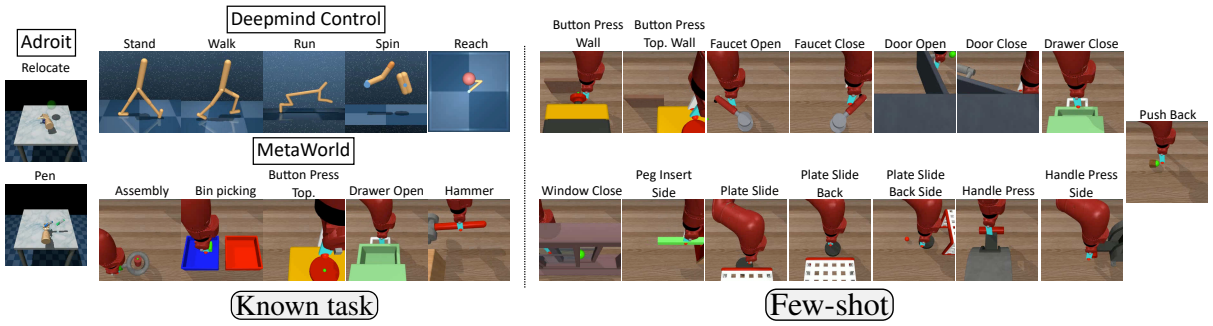


Figure 7.2: **Considered tasks**: We train the method on a set T^k of known tasks and evaluate it either on the same set, with the task known (Known task setting), or in a Few-shot setting, where a new unseen task from a set T^u is inferred from a few demonstrations.

predicts a continuous action $\hat{\mathbf{a}}_t \in \mathbb{R}^{d_a}$, where d_a is the dimension of the action space, which depends on the task at hand. We are provided with a training dataset of expert demonstrations to train a single policy, and for inference we study two different setups: Known task, where we *a priori* know the task to be executed, and Few-shot, where the trained policy must be adapted to a new unseen task without fine-tuning only given a small set of demonstrations.

Known tasks – Following Majumdar et al. (2023), we consider K=12 robotics tasks from 3 benchmarks, Adroit (Rajeswaran et al., 2018), Deepmind control suite (Tassa et al., 2018) and MetaWorld (Yu et al., 2020). The set of all known tasks is denoted as $T^k = \{t_i^k\}_{[i=1..K]}$, where t_i^k is a 1-in-K vector encoding a known task, and is illustrated in Figure 7.2.

Unknown tasks – The ability of our method to adapt to new skills is evaluated on a set of U=15 tasks from MetaWorld (Yu et al., 2020), for which we artificially generate demonstrations with a process described in section 7.3. The set of all unknown tasks is denoted as $T^u = \{t_i^u\}_{[i=1..U]}$, where t_i^u is a 1-in-U vector encoding an unknown task, and is also presented in Figure 7.2. Most importantly, $T^k \cap T^u = \emptyset$.

7.2.1 *Base agent architecture*

Following a large body of work in end-to-end training for robotics, the agent directly maps pixels to actions and decomposes into a visual encoder and a policy.

Visual encoder without adapters – following Majumdar et al. (2023), the visual encoder, denoted as ϕ, is a ViT model (Dosovitskiy et al., 2020) pre-trained with masked auto-encoding (MAE). We keep pre-trained weights from VC-1 by Majumdar et al. (2023), which are publicly available. However, we change the way the representation is collected from the pre-trained model. Unlike Majumdar et al. (2023), the representation is not taken as the embedding of the 'CLS' token, which we consider to be undertrained by the MAE pretext task. Instead, we train a fully-connected layer ψ to aggregate all the token representations of the last layer of the ViT except the 'CLS' token. The visual observation $\mathbf{v}_t$ associated with timestep t is thus encoded

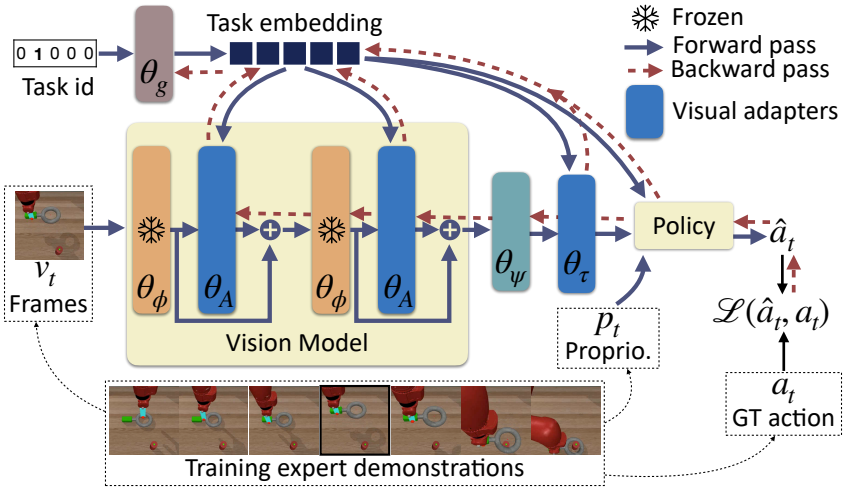


Figure 7.3: **Method overview – Training the policy and adapters**: the adapted policy is trained with BC from expert demonstrations and given a visual encoder pre-trained with MAE. The model is conditioned on a task embedding learned from ground-truth 1-in-K task identifiers.

as,

$$\mathbf{r}_t = \psi\big[\phi(\mathbf{v}_t;\theta_\phi);\theta_\psi\big], \tag{7-1}$$

where θ_ϕ and θ_ψ are weights parametrizing ϕ and ψ respectively. $\mathbf{r}_t \in \mathbb{R}^{3\times d_r}$ as it contains the d_r-dim encoding of each of the 3 last visual frames processed as a data batch, where d_r is the output dimension of ψ.

As this will be relevant later, we recall here that a ViT ϕ is composed of a sequence of N_l self-attention blocks, where ϕ_l is the layer at index l. If we denote the internal hidden representation predicted at layer l as $\mathbf{s}_t^l$, and omit the weights of ϕ_l for simplicity, we have,

$$\mathbf{s}_t^l = \phi_l(\mathbf{s}_t^{l-1}), \tag{7-2}$$

where $\mathbf{s}_t^0{=}\mathbf{v}_t$.

Single-task policy – following Majumdar et al. (2023), the policy π is implemented as an MLP predicting actions from the input which is a concatenation of the current frame, two frame differences and the proprioception input $\mathbf{p_t}$,

$$\hat{\mathbf{a}}_t = \pi\left(\Big[\mathbf{r}_{t,1}{-}\mathbf{r}_{t,0}, \mathbf{r}_{t,2}{-}\mathbf{r}_{t,1}, \mathbf{r}_{t,2}, \mathbf{p_t}\Big]; \theta_\pi\right), \tag{7-3}$$

where $[\,]$ is the concatenation operator and θ_π are weights parametrizing π.

7.2.2 *Adaptation*

Our key contributions are visual adapter modules along with a multi-task policy, which are all conditioned on the task at hand. This is done with a specific task embedding for each task, taken from an embedding space of dimension d_e, which is aimed to have sufficient regularities to enable few-show generalization to unseen tasks. Importantly, the different

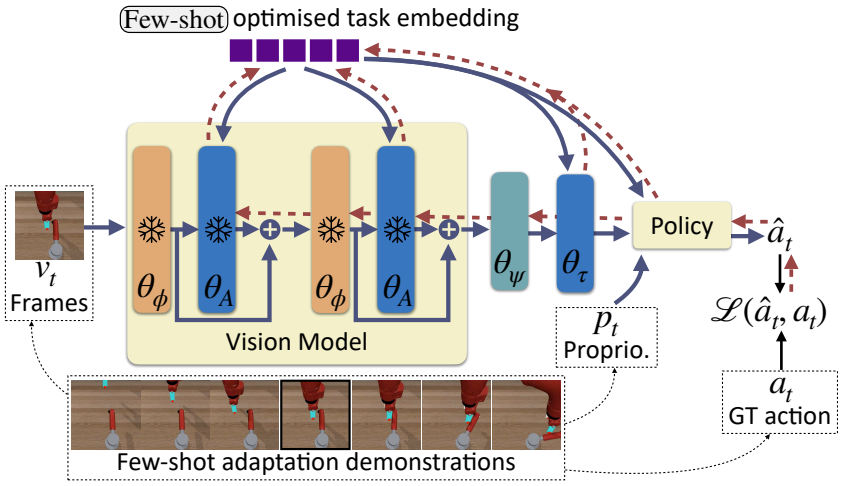


Figure 7.4: **Method overview – Optimization of the task embedding in the Few-shot setting**: In the Few-shot case, a task embedding is estimated by optimization, maximizing the likelihood of a given demonstration of an unknown task.

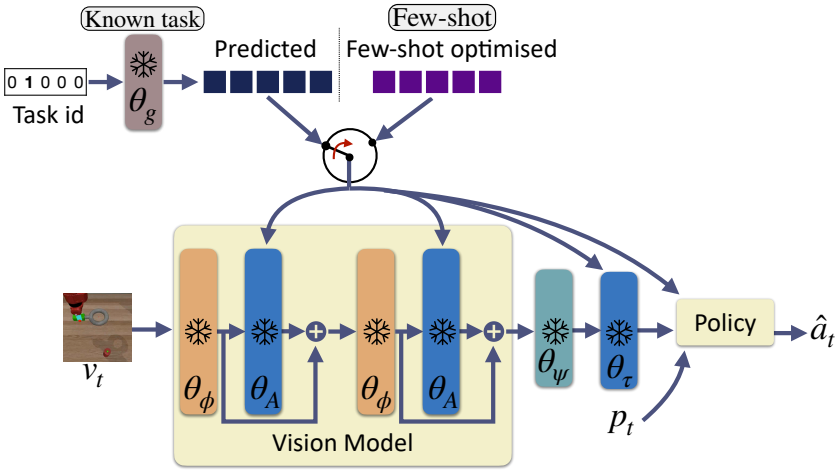


Figure 7.5: **Method overview – Inference for the Known task and Few-shot settings**: Inference uses a task embedding given in the Known task case, or estimated in the Few-shot case.

adapters and the multi-task policy are conditioned on the same task embedding, leading to a common and shared embedding space. For the *known task* setting, where the ground-truth label of the task is available, the task embedding is projected from a 1-in-K vector with a linear function g trained jointly with the adapters and the policy with the downstream loss (imitation learning). In the *Few-shot* setting, at test time a new unknown task is described with a few demonstrations, and a task embedding is estimated through optimization, as will be detailed in subsection 7.2.4. Figures 7.3, 7.4, 7.5 outline the architecture.

Conditioned on a task embedding we denote as **e**, the proposed adaptations are based on "middle" and "top" adapters following Houlsby et al. (2019); Sharma et al. (2022).

Middle adapters – we add one trainable adapter after each ViT block to modulate its output. We introduce a set of middle adapters $A = \{\alpha_l\}_{[l=1..N_l]}$, where α_l is a 2-layer MLP. In the

modified visual encoder ϕ^m, each adapter modulates the output of the corresponding self-attention block and is conditioned on the task embedding $\mathbf{e}$. Its output is combined with the one of the self-attention layer through a residual connection. If we denote the internal hidden representation predicted at layer l as $\mathbf{s}_t^{m,l}$, and omit references to the weights of ϕ_l and α_l as in equation 7-2 for simplicity, the associated forward pass of a given layer becomes,

$$\mathbf{s}_t^{m,l} = \phi_l^m(\mathbf{s}_t^{m,(l-1)}) \tag{7-4}$$

$$= \phi_l(\mathbf{s}_t^{m,(l-1)}) + \alpha_l(\phi_l(\mathbf{s}_t^{m,(l-1)}), \mathbf{e}). \tag{7-5}$$

Top adapter – A top adapter τ, also conditioned on the task at hand, is added after the ViT model, to transform the output of the aggregation layer ψ to be fed to the multi-task policy (presented below). τ has the same architecture as a single middle adapter α_l. The prediction of $\mathbf{r}_t^m$, equivalent to $\mathbf{r}_t$ in the non-adapted case, can be written as,

$$\mathbf{r}_t^m = \tau\Big[\psi\big[\phi^m(\mathbf{v}_t, \mathbf{e}; \theta_\phi, \theta_A); \theta_\psi\big], \mathbf{e}; \theta_\tau\Big], \tag{7-6}$$

where θ_A and θ_τ are the weights parametrizing the middle adapters (each middle adapter has a different set of weights) and the top adapter respectively.

Multi-task policy – We keep the architecture of the single-task policy in equation 7-3, as in Majumdar et al. (2023). However, instead of re-training a policy for each downstream task of interest, we train a single multi-task policy π^m, whose action space is the union of the action spaces of the different tasks. During training we apply a masking procedure on the output, considering only the actions possible for the task at hand.

Let's denote $\tilde{\mathbf{r}}_t^m$ as the input to the policy derived from the adapted representation $\mathbf{r}_t^m$ and the proprioception input $\mathbf{p_t}$ as done in equation 7-3. The conditioning on the task is done by concatenating $\tilde{\mathbf{r}}_t^m$ with the task embedding $\mathbf{e}$, giving

$$\hat{\mathbf{a}}_t = \pi^m\left([\tilde{\mathbf{r}}_t^m, \mathbf{e}], \theta_{\pi^m}\right), \tag{7-7}$$

where θ_{π^m} are weights parametrizing π^m.

7.2.3 *Training*

We train the model by keeping the weights of the pre-trained vision-encoder model θ_ϕ frozen, only the weights of the adapter modules (θ_A, θ_τ), aggregation layer (θ_ψ), embedding layer (θ_g) and multi-task policy (θ_{π^m}) are trained, cf. Figure 7.3. Lets' denote by $\Theta = \{\theta_A, \theta_\tau, \theta_\psi, \theta_g, \theta_{\pi^m}\}$ the set of optimized weights. We train with imitation learning, more specifically BC: for each known task t_i^k, we have access to a set of N_i expert trajectories that are composed of T_i discrete steps, including expert actions. The optimization problem is given as,

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{i=1}^{K} \sum_{n=1}^{N_i} \sum_{t=1}^{T_i} \mathcal{L}(\hat{\mathbf{a}}_{n,t}^i, \mathbf{a}_{n,t}^{i*}), \tag{7-8}$$

where $\hat{\mathbf{a}}^{i}_{n,t}$ and $\mathbf{a}^{i*}_{n,t}$ are the predicted and ground-truth actions for a given step in a trajectory, and $\mathcal{L}(.)$ is the Mean Squared Error loss.

7.2.4 *Few-shot adaption to new tasks*

For the Few-shot setting, the task embedding $\mathbf{e}$ is unknown at inference and needs to be estimated from a set of N_d example demonstrations $\mathcal{D}=\{d_n\}_{[n=1..N_d]}$ where $d_n=\{(\mathbf{v}^*_{n,t}, \mathbf{p}^*_{n,t}, \mathbf{a}^*_{n,t})\}_{[t=1..T_d]}$ is composed of observations and actions, with T_d being the length of each demonstration. We exploit the conditioning property of the policy itself to estimate the embedding $\hat{\mathbf{e}}$ as the one which obtains the highest probability of the demonstration actions, when the policy is applied to the demonstration inputs, i.e.

$$\hat{\mathbf{e}} = \arg\min_{\mathbf{e}} \sum_{n=1}^{N_d} \sum_{t=1}^{T_d} \mathcal{L}(\pi^m\left(\left[\tilde{\mathbf{r}}^{m*}_{n,t}, \mathbf{e}\right], \theta_{\pi^m}\right), \mathbf{a}^*_{n,t}), \tag{7-9}$$

where $\tilde{\mathbf{r}}^{m*}_{n,t}$ is the representation extracted from the demonstration input $(\mathbf{v}^*_{n,t}, \mathbf{p}^*_{n,t})$, and which itself depends on $\mathbf{e}$ (not made explicit in the notation). The minimization is carried out with SGD from an embedding initialized to zero.

7.3 Experimental results

Training – all variants involving adapters and/or a multi-task policy (rows (b)-(f) in Table 7.1) were trained for 50 epochs with BC, cf. §7.2.3, following training hyperparameters in Majumdar et al. (2023). Between 20 and 95 expert trajectories are available depending on the task. We used the datasets of trajectories from Majumdar et al. (2023).

Evaluation – to better handle possible overfit on hyperparameter selection, our evaluation setup is slightly different from Majumdar et al. (2023) as we perform 100 validation rollouts to select the best checkpoint of each model, and then test the chosen model on 100 test rollouts. For our multi-task policy, the best checkpoint is the one with the highest average validation performance across all tasks. Single-task policies are validated only on the task they were trained on, giving them an advantage, and for this reason, they are reported as "soft upper bounds". We report the average performance and standard deviation among 3 trained models (3 random seeds) for each variant as *mean* $\pm$ *std* (Table 7.1).

In total, we conduct evaluations of our method on 27 different tasks, 12 known and 15 unknown, with varying environments, embodiments, and required sub-skills. This allows to evaluate the adaptation and generalization abilities of the multi-task policy.

Evaluation metrics – Following Majumdar et al. (2023), we consider a rollout success (1 if the task was completed properly, 0 otherwise) for tasks in the Adroit and MetaWorld benchmarks, and report the normalized return for DMC. Episodes have a maximum length of 1000 steps and each step reward is comprised between 0 and 100 in DMC, normalization is therefore done by dividing the agent's return by 10. For all tasks, performance is averaged across

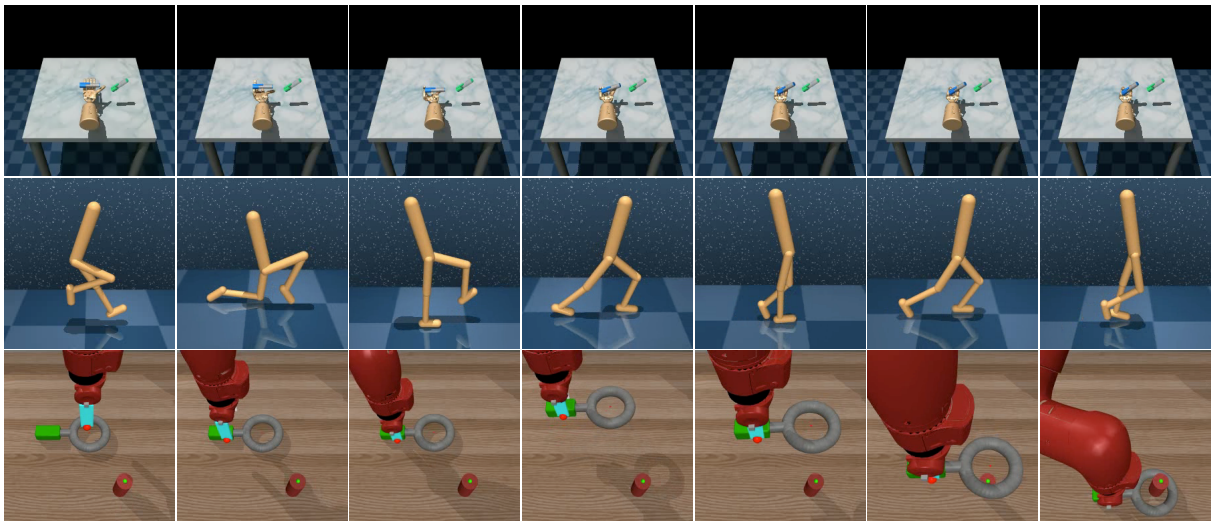


Figure 7.6: Known task — **Qualitative results**: Three successful policy rollouts on known tasks from the test set. The multi-task approach performs well on a variety of diverse tasks while being trained on a limited set of demonstrations.

rollouts.

Architecture details – are presented below.

Vision encoder: we use a ViT-B backbone, initialized from VC-1 weights, as the base vision encoder ϕ. It is made of 12 self-attention layers, each composed of 12 attention heads, with a hidden size of 768. The input image of size 224×224×3 is divided into a grid of 14×14 patches, where each patch has thus a size of 16×16 pixels. An additional 'CLS' token is appended to the sequence of image tokens to follow the setup used to pre-train the model.

Task embedding: the task embedding is a 1024-dim vector. For *known tasks*, it is predicted by a linear embedding layer from a 1-in-K vector where K=12.

Middle adapters: one adaptation module α_l is inserted after each self-attention layer inside ϕ. It is composed of 2 fully-connected layers with respectively 384 and 768 neurons. A *GELU* activation function is applied to the output of the first layer. The input to a middle adapter is the concatenation of the task embedding and a token representation from the previous self-attention layer. It thus processes all tokens as a batch.

Aggregation fully-connected layer: the input to the aggregation fully-connected layer ψ is a concatenation of the 768-dim representation of all 14×14=196 tokens. It is implemented as a simple fully-connected layer predicting a 768-dim vector representation.

Top adapter: the top adapter τ is fed with the output of ψ, again concatenated to the task embedding. It is composed of 2 fully-connected layers that both have 768 neurons. A *ReLU* activation function is applied to the output of the first layer.

	MT	Adapters		Task	Multi-task performance									
	π	Mid.	Top	emb.	Adroit		DMC		MetaWorld		Benchmarks avg		Tasks avg	
					Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
(a)	–	–	–	N/A	44.0 ± 1.1	38.3 ± 2.5	49.6 ± 0.5	48.0 ± 0.3	53.5 ± 2.1	47.8 ± 2.0	49.1 ± 0.5	44.7 ± 0.3	50.3 ± 0.9	46.3 ± 0.4
(b)	✓	–	–	L	36.3 ± 1.7	33.0 ± 4.0	55.3 ± 1.4	54.1 ± 0.3	41.7 ± 1.6	34.7 ± 0.9	44.4 ± 1.0	40.6 ± 1.8	46.5 ± 1.1	42.5 ± 1.2
(c)	✓	NC	–	L	40.2 ± 1.2	37.3 ± 2.8	54.0 ± 1.5	54.8 ± 1.9	45.8 ± 4.5	36.3 ± 2.5	46.7 ± 1.6	42.8 ± 1.9	48.3 ± 1.9	44.2 ± 1.8
(d)	✓	C	–	L	42.0 ± 2.5	**43.8** ± 2.2	59.1 ± 1.3	58.8 ± 0.3	48.6 ± 4.8	40.8 ± 3.0	49.9 ± 2.1	47.8 ± 1.4	51.9 ± 2.1	48.8 ± 1.3
(e)	✓	C	NC	L	**44.3** ± 1.2	43.2 ± 1.5	**60.5** ± 0.5	**60.3** ± 2.5	58.6 ± 1.6	48.4 ± 1.9	54.5 ± 0.7	50.6 ± 0.8	57.0 ± 0.7	52.5 ± 1.1
(f)	✓	C	C	L	42.0 ± 0.8	42.3 ± 1.0	59.9 ± 0.9	60.0 ± 0.5	**65.3** ± 1.0	**54.5** ± 3.3	**55.8** ± 0.1	**52.3** ± 1.0	**59.2** ± 0.1	**54.8** ± 1.2
(g)	✓	C	C	Rd	4.2 ± 4.0	1.3 ± 0.9	10.3 ± 0.7	8.5 ± 1.1	1.3 ± 0.9	0.1 ± 0.1	5.3 ± 0.8	3.3 ± 0.2	5.5 ± 0.1	3.8 ± 0.4
(h)	✓	C	C	RdP	0.7 ± 0.9	3.2 ± 2.5	5.7 ± 1.2	9.5 ± 6.1	0.9 ± 0.6	0.3 ± 0.4	2.4 ± 0.5	4.3 ± 2.2	2.9 ± 0.4	4.6 ± 2.5

Table 7.1: Known task — **Impact of visual adapters**: Validation and test performance on known tasks of different baselines highlighting the gain brought by adapters. Both middle and top adapters bring a boost in performance, and conditioning them on the learned task embedding increases performance. Our multi-task policy outperforms single-task policies with VC-1 non-adapted features. **MT** π: multi-task policy – NC: Non-conditioned – C: Conditioned – **Task emb.**: whether to input at evaluation time, either the learned task embedding and chosen from ground-truth (L), a random vector as the task embedding (Rd), or a randomly picked task embedding among the set of K=12 embeddings (RdP) – Benchmarks avg: average performance across the 3 considered benchmarks (Adroit, DMC, MetaWorld) – Tasks avg: average performance across all 12 known tasks. Performance is reported as *mean* ± *std* over 3 training runs (seeds).

Multi-task policy: the policy π^m is a 3-layer MLP, with 256 neurons for all layers and *ReLU* activation functions. A batch normalization operation is applied to the input to the policy. π^m outputs a 30-dim action vector, as 30 is the number of components in the action space with the most components among the 12 known tasks. When solving a task with a smaller action space, we mask out the additional dimensions.

Known task — Impact of visual adapters – Table 7.1 presents a detailed comparison of different methods on the known task setting. The baseline in row (a) follows the setup in Majumdar et al. (2023) to train single-task policies (one per task) from non-conditioned VC-1 features. For this variant, we use the representation of the 'CLS' token as the vector fed to the policy as done by Majumdar et al. (2023), while all other baselines use our proposed token aggregation layer.

Row (b) is our multi-task policy without any adapter. As expected there is a performance drop compared to the specialized policies in row (a), as the problem to solve has become more difficult. Adding adapters and conditioning them on the task embedding, shown in rows (c)-(f), brings a boost in performance, both for middle and top adapters. In particular, conditioning adds a further boost compared to non-conditioned adapters, with all choices enabled, row (f), obtaining the best average performance.

Rows (g) and (h) are ablation experiments evaluating the impact of choosing random task

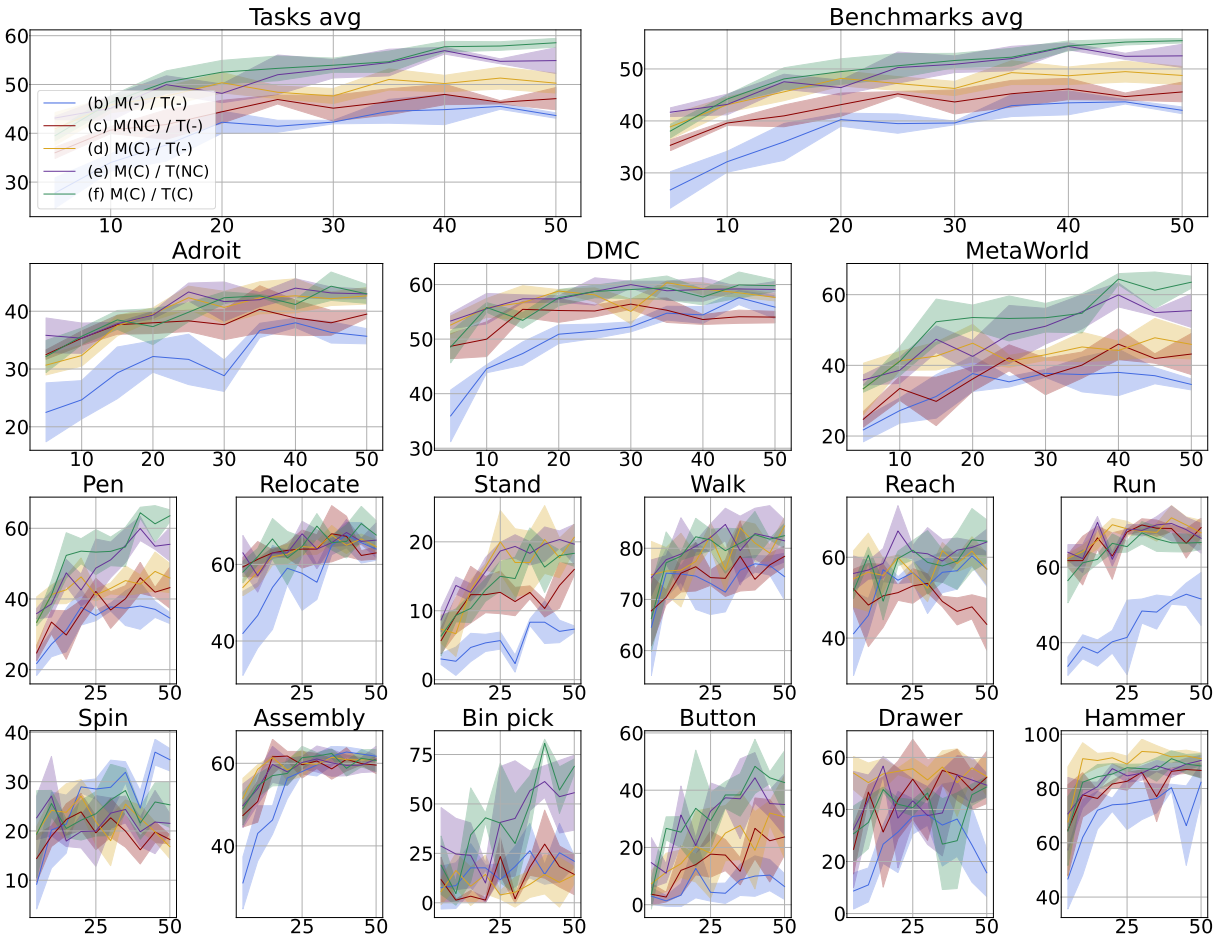


Figure 7.7: Known task — **Impact of visual adapters**: Evolution of the validation performance during training for rows (b)-(f) in Table 1 of the main paper. In the legend, M and T refer respectively to the state of middle and top adapters, and -, NC or C mean they are absent, not conditioned or conditioned on the task embedding. On all plots, the y-axis represents the performance score and the x-axis corresponds to the training epoch. Colored lines represent the evolution of mean performance over 3 training runs (3 random seeds) and shaded areas represent standard deviation.

embeddings, row (g), or of taking a random choice between the 12 learned embeddings, row (h). In both cases, the performance collapses.

A particularly important conclusion that can be drawn from the experiments outlined in Table 7.1 is that the proposed multi-task approach (row (f)) outperforms the single-task policies without adapters (row (a)). This shows that a multi-task policy can perform well on a series of tasks while being trained on a limited set of demonstrations. Figure 7.6 presents 3 successful test rollouts of our multi-task approach on diverse *known tasks*.

Figure 7.7 shows the evolution of the validation score as a function of training epochs for rows (b)-(f) in Table 7.1. These curves confirm the gain brought by both middle and top adapters, and the positive impact of conditioning them on the task at hand.

Finally, Figure 7.8 visualizes the per-task test performance on *known tasks* of single-task

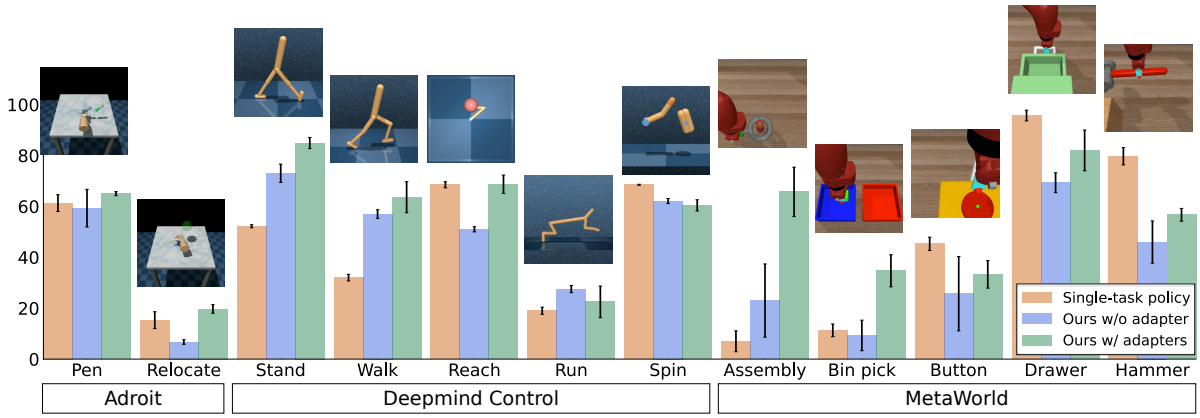


Figure 7.8: Known task — **Per-task performance** of policies in Table 7.1: Single-task policies (row (a)), our approach without any adapter (row (b)) and with conditioned middle and top adapters (row (f)). The adapters lead to a performance gain on most tasks, and our multi-task solution is competitive with single-task policies. Colored bars and error bars respectively show mean and std over 3 training runs (seeds).

policies (row (a) in Table 7.1), our approach without any adapter (row (b) in Table 7.1) and with conditioned middle and top adapters (row (f) in Table 7.1). The proposed adapters lead to a performance gain on most tasks compared with the solution without adapters, and the multi-task solution is competitive with single-task policies, even outperforming them on half the tasks.

Known task — Additional ablation studies – After studying the impact of introduced adapters, we conduct other studies presented below to learn more about specific design choices. Results are shown in Table 7.2.

Impact of conditioning the policy on the task at hand: Row (b) in Table 7.2 reaches the same performance, even slightly better, as row (a), showing that when adapters are conditioned on the task at hand, conditioning the policy itself is not necessary. This seems to indicate that conditioned adapters already insert task-related information into visual embeddings fed to the policy.

Using the 'CLS' token representation as input to the policy: Row (c) in Table 7.2 performs worse than row (a), indicating that our introduced tokens aggregation layer ψ improves over the strategy used in previous work consisting in feeding the output 'CLS' token to the policy. This confirms our assumption that the 'CLS' token is undertrained under the MAE pre-training task.

Impact of the middle adapters when using top adapters: Row (d) in Table 7.2 also performs worse compared with row (a), showing that when training a conditioned top adapter, middle adapters are still very important, bringing a significant boost in performance.

Known task — Impact on other visual backbones – Table 7.3 shows the impact of our task-

	Cond. π	Adapters Mid.	Adapters Top	'CLS' token	Multi-task performance									
					Adroit		DMC		MetaWorld		Benchmarks avg		Tasks avg	
					Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
(a)	✓	C	C	–	42.0 ± 0.8	42.3 ± 1.0	59.9 ± 0.9	60.0 ± 0.5	65.3 ± 1.0	54.5 ± 3.3	55.8 ± 0.1	52.3 ± 1.0	59.2 ± 0.1	54.8 ± 1.2
(b)	–	C	C	–	42.3 ± 2.0	40.8 ± 3.0	59.2 ± 1.0	59.2 ± 2.3	68.7 ± 2.2	57.6 ± 3.4	56.8 ± 0.8	52.5 ± 0.9	60.4 ± 0.8	55.5 ± 0.5
(c)	✓	C	C	✓	38.8 ± 5.9	36.2 ± 2.3	57.8 ± 1.9	58.1 ± 2.6	57.5 ± 8.0	50.9 ± 4.3	51.4 ± 2.8	48.4 ± 0.6	54.5 ± 2.9	51.4 ± 1.3
(d)	✓	–	C	–	34.7 ± 1.5	34.7 ± 3.3	51.4 ± 0.4	52.1 ± 0.5	53.6 ± 4.4	44.5 ± 4.7	46.6 ± 1.6	43.8 ± 1.8	49.5 ± 2.0	46.0 ± 1.9

Table 7.2: Known task — **Additional ablation studies**: Validation and test performance on known tasks of different neural variants. Row (a) is equivalent to row (f) in Table 1 of the main paper. When using conditioned adapters, giving the task embedding as input to the policy is not necessary. Our introduced tokens aggregation layer is better than using the representation of the 'CLS' token. Finally, when training a conditioned top adapter, middle adapters are still important and bring a boost in performance. **Cond** π: policy conditioned on the task embedding – C: Conditioned – **'CLS' token**: using the 'CLS' token representation as the frame embedding fed to the policy. Performance is reported as *mean* ± *std* over 3 training runs (seeds).

ViT	Ours	Multi-task performance									
		Adroit		DMC		MetaWorld		Benchmarks avg		Tasks avg	
		Val	Test	Val	Test	Val	Test	Val	Test	Val	Test
PVR (Parisi et al.)	–	34.7±2.8	30.2± 1.0	58.1±3.0	55.3± 7.2	41.8± 1.7	33.5± 1.4	44.8± 1.3	39.6± 3.0	47.4± 1.3	42.0± 3.5
	✓	**43.3**± 2.5	**41.0**± 5.1	**61.9**± 1.3	**61.3**± 1.2	**66.8**± 2.3	**55.7**± 3.4	**57.3**± 1.0	**52.7**± 3.1	**60.9**± 0.8	**55.6**± 2.7
MVP (Radosavovic et al.)	–	38.0±1.3	34.3±2.4	56.5±2.1	56.5±1.7	42.8±6.9	35.9±5.6	45.8±1.5	42.2±2.2	47.7±1.9	44.2±2.2
	✓	**47.7**±5.9	**46.2**±2.4	57.3±2.9	56.9±2.5	**64.9**±12.1	**55.4**±12.2	**56.6**±4.0	**52.8**±2.6	**58.9**±4.4	**54.5**±3.8

Table 7.3: Known task — **Impact on other visual backbones**: Validation and test performance on known tasks for two additional visual backbones (PVR Parisi et al. (2022) and MVP Radosavovic et al. (2023)). Our task-conditioned adapters improve the extracted visual features in both cases, leading to higher multi-task policy performance. Performance is reported as *mean* ± *std* over 3 training runs (seeds).

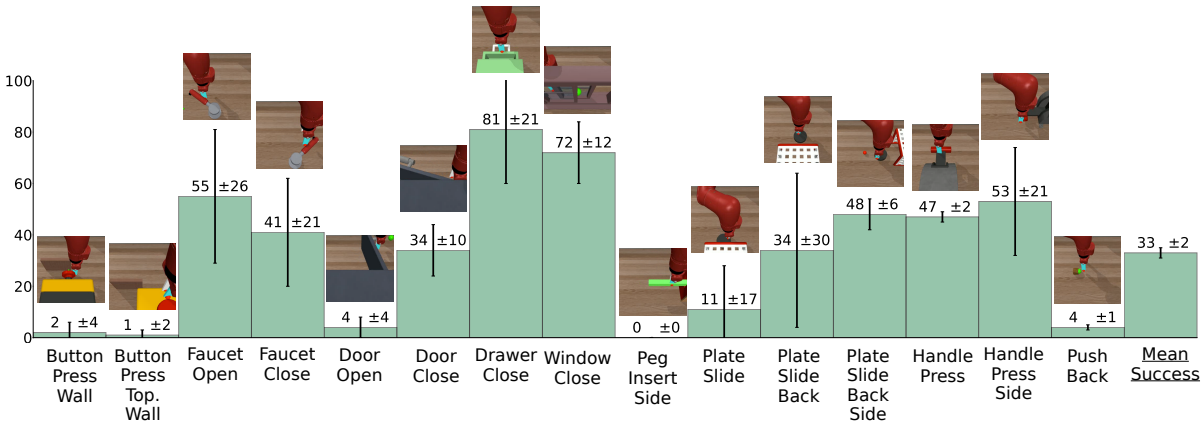


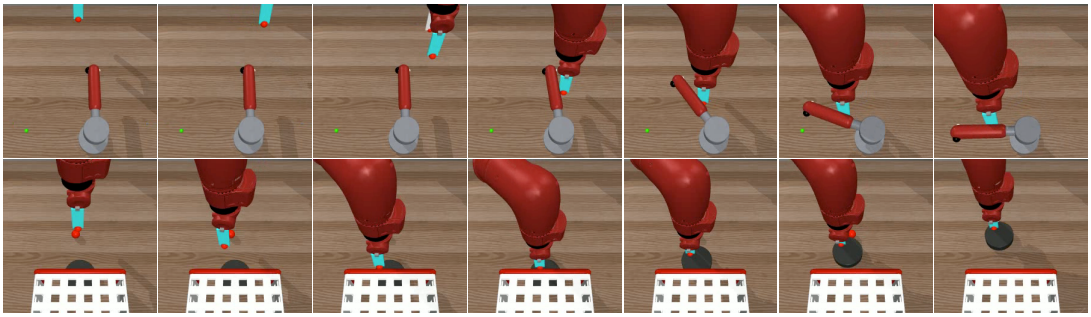
Figure 7.9: Few-shot — **Per-task performance**: After optimizing the task embedding for each task from 5 demonstrations, our method can adapt to many of them (without finetuning). Colored bars and error bars respectively show mean and std over 3 training runs (seeds).

conditioned adapters on the visual features extracted by two other SOTA ViT-B backbones, i.e. PVR (Parisi et al., 2022) and MVP (Radosavovic et al., 2023). As can be seen, our adapters bring a boost in policy performance for both pre-trained backbones, generalizing the conclusions obtained with VC-1.

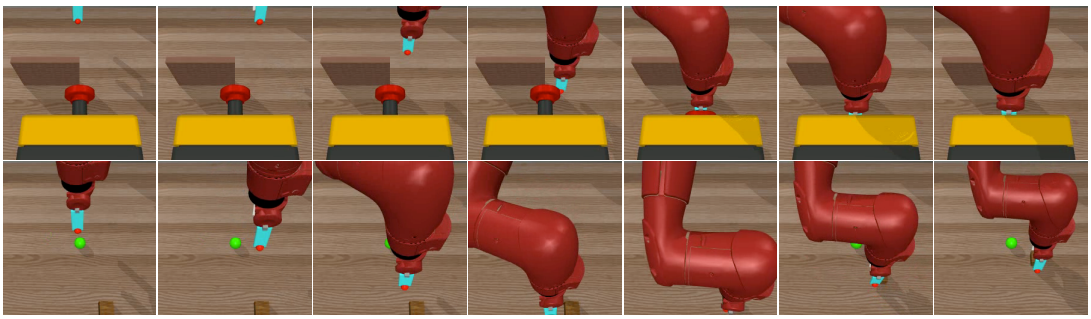
Few-shot – adaptation to new tasks without finetuning model weights is an important ability of any general policy, which we evaluate with the following experiments: for each task within a set of unknown tasks (cf. Figure 7.2), we collect only 5 demonstrations used to optimize a task embedding specific to this task with the method detailed in section §7.2.4. We then evaluate the method conditioned on the optimized embedding on 100 test rollouts.

To generate the set T^u of unknown tasks, we select tasks from the MetaWorld dataset that do not belong to *CortexBench*, and are thus not part of the set of training known tasks T^k. We collect demonstrations using single-task policies from TD-MPC2 (Hansen et al., 2024) that were specifically trained on each task of MetaWorld independently. To ensure high-quality demonstrations, we only consider tasks where TD-MPC2 policies reach a success rate higher than 95%. Furthermore, to be compatible with the setup from *CortexBench* authors, in particular, to keep the same camera locations, we filter out the tasks where the goal is not always visible in the camera FOV. This leads to a set of 15 unknown tasks that are quite different from the tasks in the training set T^k as they involve different objects (*handle press, faucet, plate, door, window*, etc.) and types of manipulation (*sliding an object, lowering a press, opening a window*, etc.). Each collected demonstration is a sequence of visual frames, proprioception inputs, and expert actions. The optimization of the task embedding is performed independently for each task (cf. §7.2.4). We use the AdamW optimizer and a learning rate of $1e{-}1$ during the task embedding search.

Figure 7.9 presents the per-task performance in this setting. Despite the large variations between the new tasks in T^u and the ones in the training set (T^k), the multi-task policy can



(a) Successful rollouts



(b) Failed rollouts

Figure 7.10: Few-shot — **Qualitative results**: (a) The policy tackles new tasks involving objects and/or manipulation requirements unseen during training. (b) In the first row (*button-press-wall* task), it performs the task correctly until the end where it fails to properly push the button fully. In the second row (*push-back* task), it properly moves the cube but fails to bring it to the goal position (green dot).

adapt to many of them, without requiring any weight finetuning. Interestingly, the method performs particularly well on the *Drawer Close* task, which could be related to the presence of the time-inversed *Drawer Open* task in the training set. This provides some evidence that the method can exploit regularities between tasks, which seem to be captured by the task embedding space, making it possible to generalize to unseen variations. Figure 7.10 (a) shows qualitative examples of successful rollouts on the new tasks. The policy manipulates new objects (*faucet*, *plate*, *window*) and performs new moves (rotating the faucet or sliding the plate) not seen during training.

Finally, Figure 7.10 (b) shows failure cases on unknown tasks. As seen on the first row, the policy avoids the wall, reaches the button, and starts pushing it, but fails to push it fully. This particular behavior was also observed on other rollouts, explaining the low success rate on this *button-press-wall* task while mastering a part of the required sub-skills. On the second row, the policy is able to move the cube but fails to bring it to the goal location (green dot). This gives some indication of the difficulty of the few-shot generalization case: exploiting regularities in the task space requires that tasks be performed more than just approximately, as often the success metric is sparse, and rollouts only count to the metric when they are executed fully and correctly.

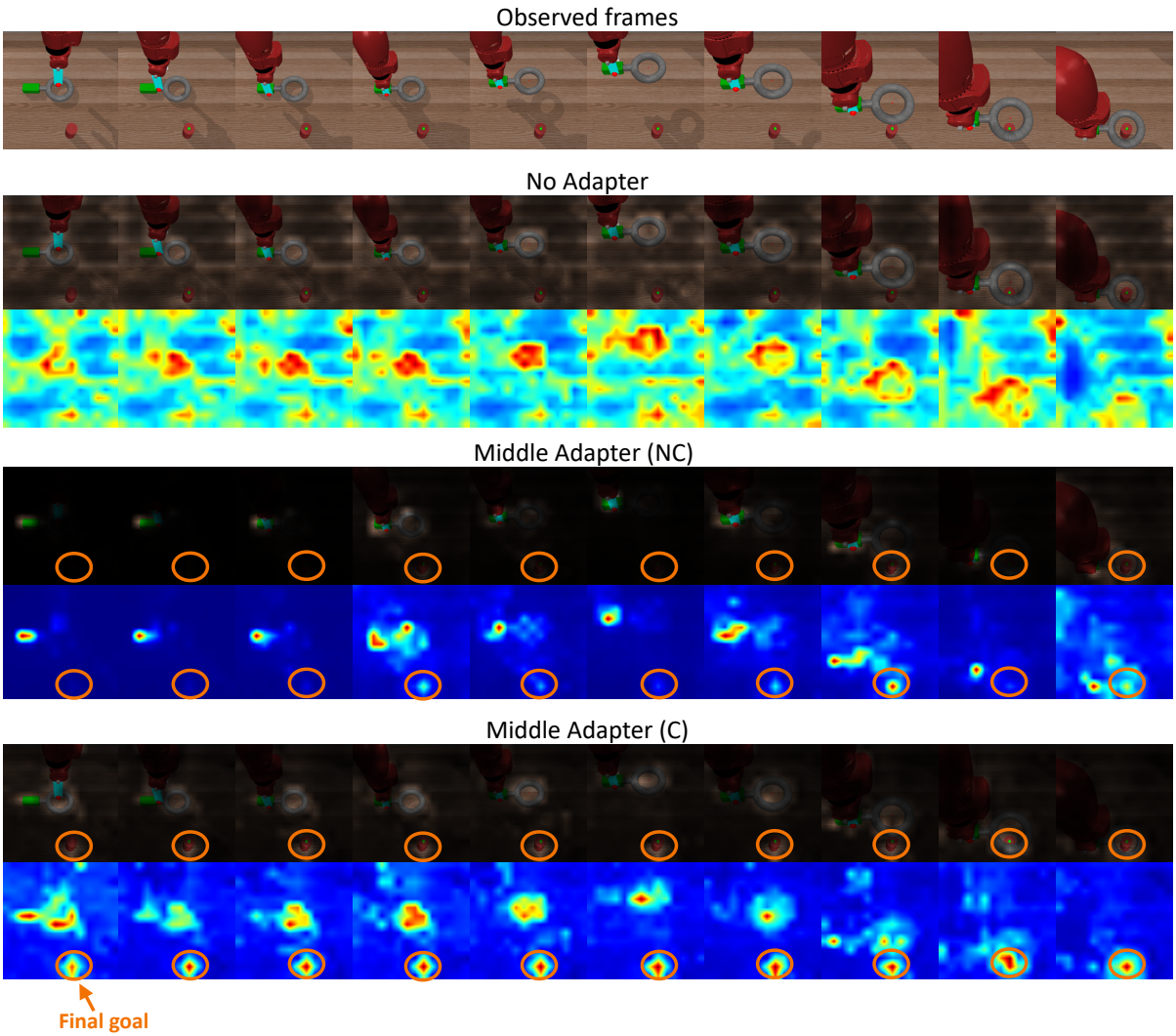


Figure 7.11: **Visualization of attention maps (Assembly task)** – First row: observed input frames. Following blocks: for each model type, we show the attention map of the last ViT layer, first overlaid on top of the visual frame and below as a colored heatmap. In this example, middle adapters allow to focus the attention on important regions, and task conditioning leads to a better covering of entire objects and agent parts, along with greater attention towards the final goal for all frames.

Known task **— Visualizing the influence of task-conditioned adaptation on attention** – Sequences of visual frames used in this experiment are taken from a held-out set of expert trajectories not used at training time. We visualize here the attention map of the last layer of the vision encoder. To this end, we sum attention maps for all tokens and all heads, and normalize them between 0 and 1. These visualizations are shown in Figures 7.11 and 7.12. In both figures, the first row presents a sequence of visual frames and below, for each model variant (*No Adapter*, *Middle Adapter (NC)*, *Middle Adapter (C)*), one can see the attention map overlaid on top of the visual frame and displayed below as a colored heatmap.

As can be seen in Figure 7.11, the middle adapters help focus the attention on the most important parts of the image compared with vanilla VC-1 attention without adapters. When

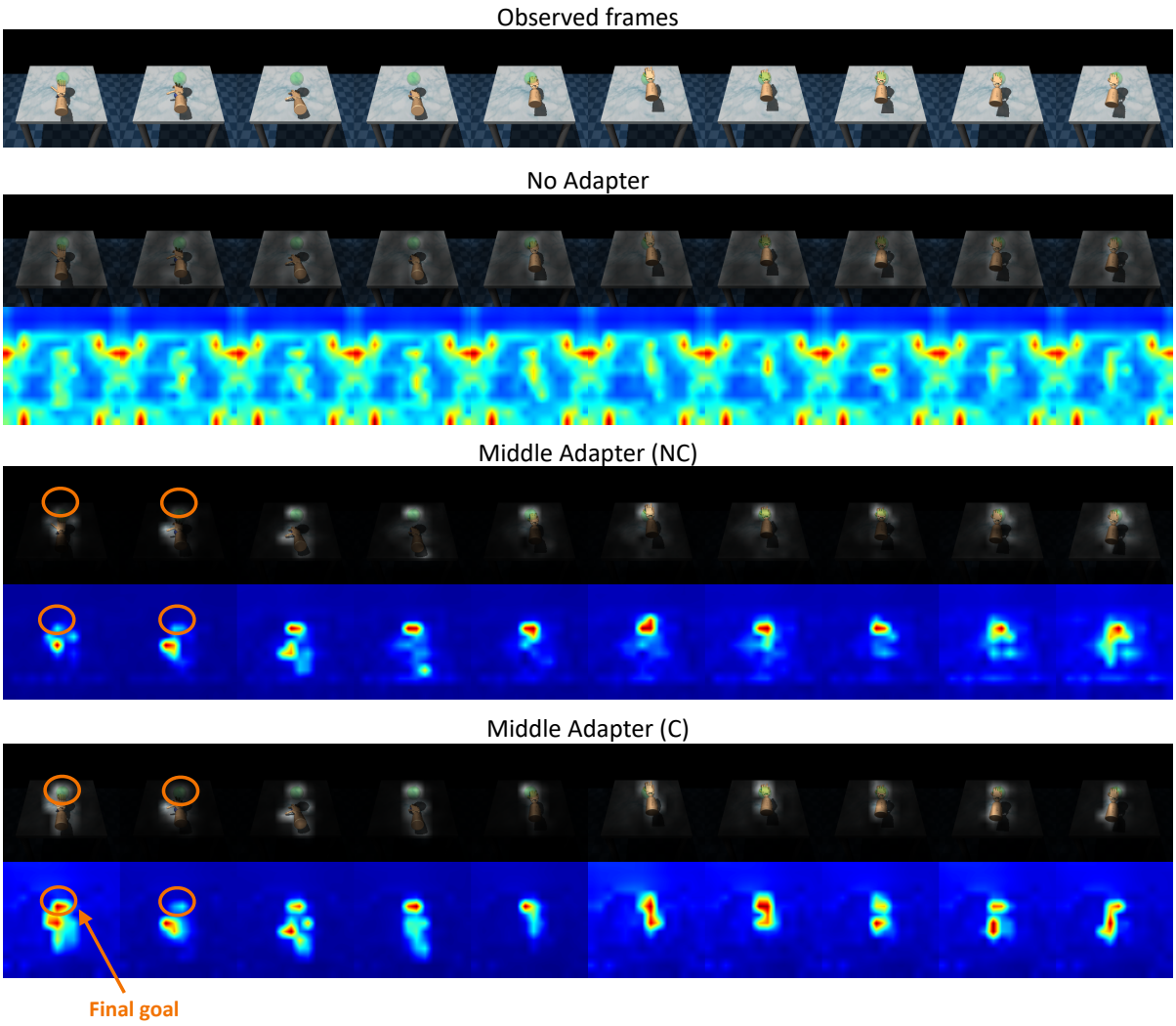


Figure 7.12: **Visualization of attention maps (Relocate task)** – First row: observed input frames. Following blocks: for each model type, we show the attention map of the last ViT layer, first overlaid on top of the visual frame and below as a colored heatmap. In this example, middle adapters allow to focus the attention on important regions, and task conditioning leads to a better covering of the robotic hand and the sphere goal in all frames.

adapters are not conditioned (NC), they tend to produce very narrow attention. Conditioning on the task at hand keeps the focus on important regions and leads to covering the entire objects of interest and important agent parts. Most importantly, when adapters are conditioned on the task embedding, more attention is put on the final goal in all frames, while this is not the case for unconditioned adapters.

Figure 7.12 presents another visualization of the attention map of the last layer of the vision encoder, confirming that middle adapters lead to better-focused attention around important objects related to the task, compared with vanilla VC-1. Unconditioned adapters (NC) tend to either produce very narrow (first frames in Figure 7.12) or quite broad (last frames in Figure 7.12) attention. Task conditioning leads to a better coverage of entire objects and

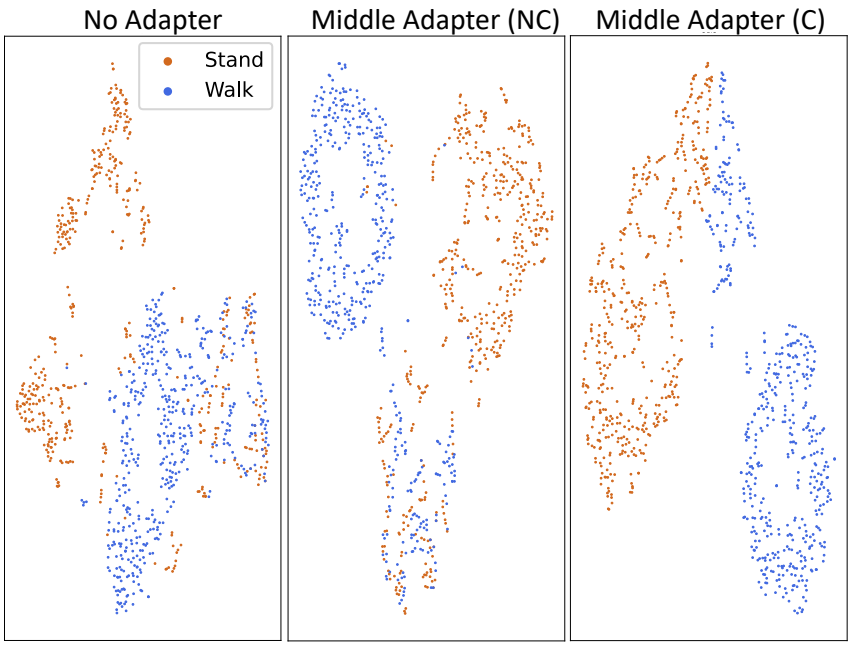


Figure 7.13: **Task-related information inside visual embeddings**: t-SNE plots of visual embeddings for a set of frames for DMC *Stand* and *Walk* tasks. We chose these tasks for their visual similarity, making it hard to distinguish between them from vision only. Conditioning of the middle adapters leads to two properly separated clusters, showing the insertion of task-related information into the visual embeddings.

important agent parts, again improving the attention towards the final goal to reach.

Known task **— Conditioning middle adapters helps insert task-related information into visual embeddings** – In order to study the underlying mechanisms of adapter modules, we examine the content of produced visual embeddings. Figure 7.13 shows t-SNE plots of visual embeddings for a set of frames for both the DMC *Stand* and *Walk* tasks. Visual observations are identical for both tasks at the beginning of rollouts, and very similar in the rest of the sequences, making it very hard to distinguish between these 2 tasks from a visual observation only. Embeddings from the conditioned middle adapters form two well-separated clusters, showcasing the task-related information brought by conditioning adapters on the task at hand.

Known task **— Non-linear probing of actions** – Table 7.4 shows the performance of a probing MLP network trained to regress the expert action to take from the visual embedding of a single frame only. As can be seen, its performance improves drastically when trained on embeddings predicted by a vision encoder composed of conditioned middle and top adapters. A conditioned top adapter thus inserts action-related information within visual embeddings.

Known task **— Diversity of known tasks** – Table 7.5 (b) and Table 7.6 (c) show a model trained on MetaWorld only, which performs better on MetaWorld than models trained on all 3 benchmarks (the domain gap between them is large). The lower performance on MetaWorld when training on all 3 benchmarks is largely outweighed by the ability to address Adroit and

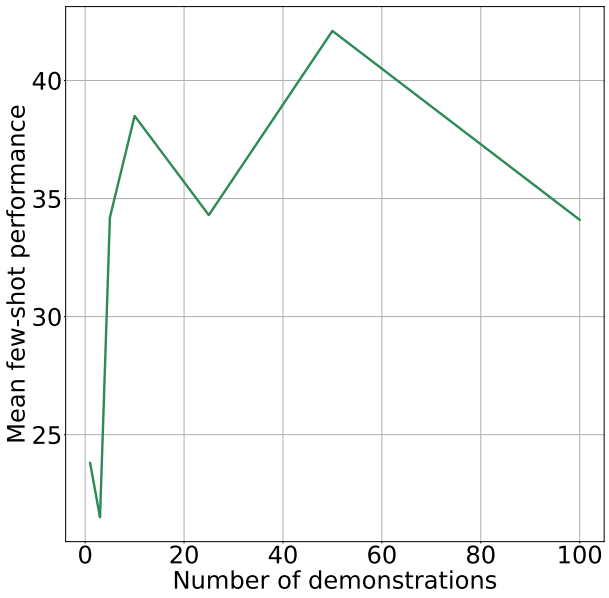


Figure 7.14: Few-shot — **Impact of the number of demonstrations on few-shot adaptation to unseen tasks**: Few-shot performance as a function of the number of demonstrations used to optimize the task embedding. We can achieve 23.8% from a single demonstration, and more demonstrations can lead to higher performance. However, the trend is not as simple as 100 demonstrations lead to the same performance as 5 demonstrations.

DMC.

Few-shot — Few-shot adaptation baseline – Table 7.6 compares model finetuning on new tasks (b) with our task embedding search (a). As expected, (b) performs better but task embedding search (a) solves a harder problem, as we keep a single policy. Our adapters can thus be used in 2 settings: (i) task embedding search, keeping a single policy addressing all tasks (low memory footprint), (ii) task-specific fine-tuning to reach the best performance possible if memory is not an issue (one specific set of 130M parameters for each task).

Few-shot — Impact of the number of demonstrations on few-shot adaptation to unseen tasks – Figure 7.14 presents the evolution of the average few-shot performance of our method across the 15 unknown tasks depending on the number of available demonstrations when optimizing the task embedding. From only a single demonstration per task, we can already reach a satisfying 23.8% mean performance. Adding more demonstrations can allow reaching higher performance, but the scaling law does not appear to be as simple as using 100 demonstrations leads to the same final performance as 5 demonstrations.

	Middle adapters	Top adapter	MSE	R^2
(a)	–	–	0.067	0.69
(b)	NC	–	0.069	0.57
(c)	C	–	0.069	0.59
(d)	C	NC	0.037	0.90
(e)	C	C	**0.034**	**0.92**

Table 7.4: Known task — **Non-linear probing of actions**: We explore the performance of action regression from the visual embedding of a single frame. Considered metrics are the Mean Squared Error (MSE) and coefficient of determination (R^2). The top adapter seems to insert action-related information into the visual embedding, as the probing MLP achieves the best performance.

	Training	**MetaWorld**	
		Val	Test
(a)	All 3 benchmarks	65.3±1.0	54.5 ±3.3
(b)	MetaWorld only	75.6±1.6	67.8±2.6

Table 7.5: Known task — **Diversity of known tasks**: Validation and test performance on MetaWorld known tasks when our approach with task-conditioned adapters is either trained on the three considered benchmarks (Adroit, DMC, MetaWorld), or on tasks from MetaWorld only. As expected, the model trained on known tasks from MetaWorld only reaches higher performance. Performance is reported as *mean* ± *std* over 3 training runs (seeds).

	Opt.	**Train.**	**Setting**	t^u_0	t^u_1	t^u_2	t^u_3	t^u_4	t^u_5	t^u_6	t^u_7	t^u_8	t^u_9	t^u_{10}	t^u_{11}	t^u_{12}	t^u_{13}	t^u_{14}	**Mean**
(a)	TE opt.	All 3	Single policy	2±4	1±2	55 ±26	41±21	4±4	34±10	81±21	72±12	0±0	11±17	34±30	48±6	47±2	53±21	4±1	33±2
(b)	Ft.	All 3	15 policies	1±2	0±0	49±44	69±22	5±2	62±8	96±4	91±7	2±3	54±8	50±4	22±19	66±7	89±1	3±2	44±3
(c)	TE opt.	MW	Single policy	3±6	0±0	56±44	64±20	1±2	69±19	100±0	77±20	0±1	6±6	22±38	19±3	55±13	57±17	5±3	36±5

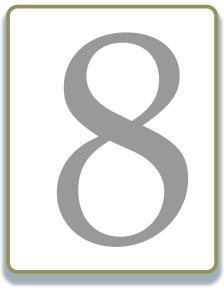
Table 7.6: Few-shot — **Performance of a finetuned baseline** (*Ft.*) and task embedding search (*TE opt.*) for a policy either trained on MetaWorld only (*MV*) or all 3 benchmarks (*All 3*). t^u_i refers to the i-th unknown task. Performance is reported as *mean* ± *std* over 3 training runs (seeds).

7.4 Conclusion

Perception and action are closely tied together, and studies of human cognition have shown that *a priori* knowledge about a downstream task guides the visual system. We followed this direction in the context of artificial agents by introducing task-conditioned adapters that modulate the visual features of a pre-trained neural backbone. Such adapters, conditioned on a learned task embedding, improve the performance of a multi-task policy across benchmarks and embodiments. Even more interesting is the use of task embeddings to adapt in a few-shot manner, i.e. from a small set of demonstrations, to new tasks unseen at training time. We proposed an optimization procedure to estimate a new task embedding and achieve generalization to unseen tasks, involving new objects and manipulation sub-skills, providing evidence for regularities in the learned embedding space.

The next chapter will be the occasion to summarize what we covered in this manuscript, and to take a step back: what are current promising trends in Embodied AI and, more importantly, what are future perspectives?

Chapter



Conclusion

Do you remember the *coffee dilemma* from the introduction of this manuscript? We have covered a few different subjects related in one way or another to it. Needless to say we are still far from autonomous agents able to reliably make a coffee themselves. We will first review what we have covered in this manuscript, before taking a step back to consider prominent directions in Embodied AI, either touched in the previous chapters or further in terms of considered concepts.

8.1 Summary of the presented contributions and directly related perspectives

First, chapter 4 explored the use of mapping-related auxiliary supervision to help the emergence of spatial reasoning abilities in neural agents. Interestingly, we were able to show that teaching agents to remember the location of previously seen landmark objects improves the navigation abilities of different variants, either encoding past experience with a grid map representation but also with a simple recurrent memory. This provides a relevant alternative to augmenting neural architectures with inductive biases: instead, one can include prior information about the task to solve directly in the supervision signal. Reducing inductive biases might be a way to keep more representation flexibility while augmenting the supervision signal could lead to the same performance with a simpler model and could be more straightforward to implement.

However, equipping neural agents with specific modules to help with important subskills such as scene mapping is still a promising direction as shown in chapter 5 where we studied the integration of neural implicit representations of space in RL-trained agents. As

presented in chapter 3, neural fields have been leveraged in robotics, in particular, to perform real-time visual SLAM, but without experimenting with how to incorporate them into an agent navigating autonomously to solve a task of interest. We thus targeted the real-time optimization of semantic and geometric neural representations and the RL-based training of neural agents to use these representations, showing good navigation performance. More specifically, we introduced continuous representations of semantic landmark locations and explored area, the latter requiring a special global reader to extract a global summary from representation weights. Our studies showed that implicit representations are an interesting direction but come with efficiency challenges, mainly if included in an RL training loop, for which we proposed solutions.

Unlike in chapter 5 where we questioned how to use neural fields optimized in real-time to support semantic navigation, chapter 6 dealt with a related yet different question: *how to autonomously navigate a new environment to collect NeRF training data?* This time, the neural field is not used at navigation time, but the agent tries to explore the environment as thoroughly as possible to build a permanent 3D reconstruction of the scene in a second stage to be used later for downstream tasks. We also reflected on how to best evaluate the quality of a trained neural field in the context of robotics, going further than the standard novel view synthesis evaluation task. We showed that a modular policy could explore a house-scale unknown scene to collect data in a single episode allowing to reconstruct the environment. Further than this, we also motivated the interest of such an approach to perform an automatic scanning of a new scene, allowing to then use the obtained 3D representation to perform safe fine-tuning of other policies of interest in simulation.

While chapters 4, 5 and 6 targeted the evaluation of the generalization abilities of neural agents to new scenes for a specific task, chapter 7 studied how to best adapt visual features to solve different tasks with a single policy, even considering generalizing to new ones. Visual perception has been an important topic of this whole manuscript, but this last chapter was about explicitly adapting a pre-trained visual model to flexibly extract relevant features depending on the task to solve. We introduced task-conditioned adapters, allowing to modulate the features of a pre-trained vision backbone. Adapters have been extensively used in NLP or CV and our study showed that task-conditioned adaptation could be a relevant solution to the problem of on-the-fly adaptation we would like autonomous agents to be able to solve. Indeed, the strategy to first pre-train a general vision model on a large diverse dataset that will then be adapted from a few demonstrations of a task of interest is appealing because very efficient.

8.2 Current and future trends in Embodied AI touched in this manuscript

This manuscript studied the improvement of mapping abilities in neural agents, both by augmenting the underlying supervision signal and proposing neural-based mapping methods.

Figure 8.1: PRISM-1 reconstruction sample – reproduced from[1]

We also covered the active collection of training data for such neural-based representations using autonomous agents. Finally, we considered the adaptation of visual features extracted by pre-trained vision models to the task at hand.

The common point between these different topics is the generalization to new 3D environments and/or tasks for neural-based agents. Indeed, the ability to robustly adapt to novelty is essential when targeting autonomy.

8.2.1 *Generalization to new environments*

As presented in chapter 3, generalization to new environments can be improved thanks to the increase in the quality and scale of 3D datasets, but also the neural models themselves to make them more robust.

More realistic simulation – Current efforts in the community are directed toward simulation realism to narrow the gap between simulation and reality. A recent example is in the autonomous driving domain, which shares many similarities with Embodied AI, with the *PRISM-1* model introduced by *Wayve*[1]. A reconstruction sample is shown in Figure 8.1, highlighting the high-quality rendering of dynamic outdoor scenes. *PRISM-1* is a 4D scene reconstruction model where in addition to space, time is also taken into account in the reconstruction of a scene. It is thus possible to train an agent to interact in a photo-realistic dynamic scene with pedestrians, bikes, cars moving. Methods like NeRF or the more recent Gaussian Splatting might lead to higher-quality simulations if their rendering speed can still be increased to match the requirements of learning algorithms such as ones based on Reinforcement Learning.

Increasing data diversity – The scale of current 3D datasets could also benefit the recent im-

[1] https://wayve.ai/thinking/prism-1/

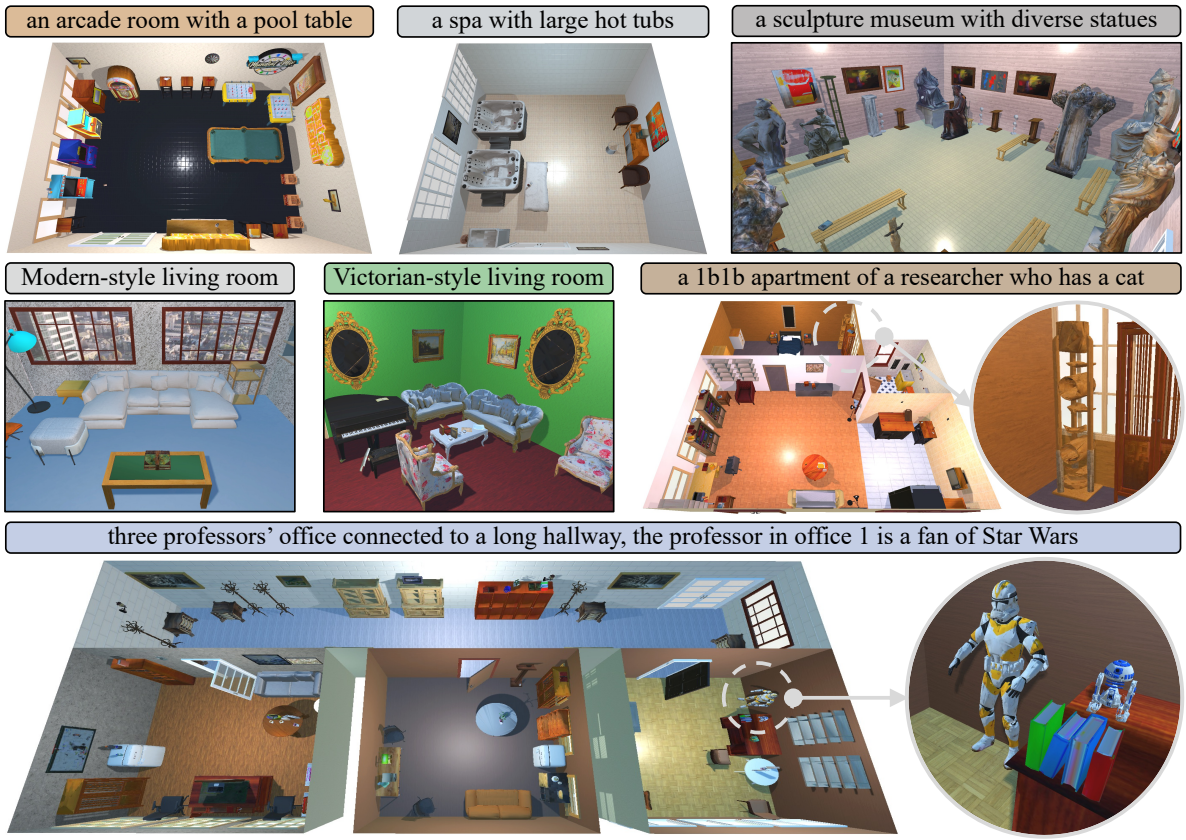


Figure 8.2: Holodeck generation samples – reproduced from Yang et al. (2024)

provements in generative models. Indeed, 2D language-prompted generative models have been successfully introduced during the past years (Ramesh et al., 2021; Rombach et al., 2022; Saharia et al., 2022; Podell et al., 2024), leading to a significant improvement in image generation from language queries. This trend is now entering the space of 3D generative modeling with approaches like *Holodeck* (Yang et al., 2024) generating 3D scenes from a text prompt. Figure 8.2 provides examples of generated 3D environments. All appear to be realistic, detailed, and well aligned with specified prompts. Automatic generation of 3D assets thus becomes a promising direction to follow in the future to increase the size and diversity of Embodied AI datasets. Moreover, this might provide more control over the characteristics of used scenes with for instance the ability to control the semantics of objects inside generated environments, with potentially additional semantic maps available without the need to rely on human annotators.

Improving the robustness of neural representations – In addition to improving the quality and quantity of data, important work should also be conducted on neural models themselves to make them more robust to domain shifts and generalize better to long-tail scenarios. Visual perception is indeed an important part of Embodied AI systems and current methods still suffer from a lack of robustness. Some works like the recent *COLOSSEUM* benchmark (Pumacay et al., 2024) evaluate the generalization abilities of current manipulation models to environ-

ment perturbations. More specifically, they study 20 manipulation tasks and implement 14 perturbation factors such as changes in object mass, friction, table color, texture, or different camera poses. They show that many vision-based models such as R3M (Nair et al., 2023) or MVP (Radosavovic et al., 2023) already considered in chapter 7 in this manuscript suffer from perturbations, leading to low performance of policies trained with BC from their visual observations as input. Benchmarks such as *COLOSSEUM* will be helpful to better evaluate the robustness of vision models and policies to environment perturbations.

8.2.2 *Generalization to new tasks*

Autonomous agents should be able to robustly operate in any environment, but also to adapt to new tasks that might be variants of already known ones, or to learn new skills on the fly from limited data. Generalizing to new tasks is thus another crucial ability to consider, as it is currently very hard for our neural-based methods. Indeed, most progress has been achieved with a combination of scientific novelties (e.g. architectures or training signals) and data scaling. A regime where our approaches shine less is when encountering new tasks, in particular when specified through a limited set of data samples.

Benchmarking fluid intelligence – In chapter 7, we started to touch an important topic, referred to by Chollet (2019) as *fluid intelligence*, i.e. the ability to efficiently learn to master new skills from sparse supervision. We studied this from a robotic point of view with the generalization to new manipulation tasks from a few expert demonstrations, but other benchmarks in the broader AI landscape study it as well. A well-known example is the *ARC challenge* introduced by Chollet (2019), where the goal is to solve IQ test-like problems without knowing about them a priori and from only a few examples. Studying few-shot generalization in domains with simple observation spaces (e.g. 2D discrete grids in ARC) might allow designing new fundamental learning and adaptation methods that could later be applied to robotics.

General visual backbones in Embodied AI – Generalizing to different tasks might require general feature extractors, pre-trained visual models in the case of the vision modality. As presented in chapter 7, previous work has already studied the use of pre-trained frozen models to extract features from visual observations fed to a single or several task-specific policies. This direction appears very promising as a single model pre-trained on a large and diverse dataset might lead to rich features containing all required information for any task of interest. However, an interesting question thus arises: *Can a single feature-based representation provide the required information for any task?* In a very recent position paper, Huh et al. (2024) introduce the *Platonic Representation Hypothesis* stating that "*Neural networks, trained with different objectives on different data and modalities, are converging to a shared statistical model of reality in their representation spaces*". They experimentally verify this hypothesis, focusing on vector embedding representations only, showing signs of a general representation space of the world, to which different models trained on various modalities seem to converge. While these results

are very intriguing and worth further studying, such a general representation space might not be guaranteed to provide the required representational flexibility to extract relevant features for all tasks. An alternative direction is thus, as presented in chapter 7, to train visual models to adapt the final representation of an observation based on the task to solve, reminiscent of the top-down processes observed in visual regions of the human brain. While we will not present works in this area as already done earlier in this manuscript (chapters 3 and 7), this direction appears very appealing and is worth considering in the future.

8.2.3 *Inductive biases in neural architectures*

Another important question is about the structure of autonomous agents, with a recurrent opposition happening between end-to-end and modular agents. Another way to frame this question could be: *Should all sub-skills be implicitly learned by a full neural agent, or should we integrate prior knowledge through specific modules?* One should probably consider two different timelines when thinking about this question: short-term and long-term progress.

Modular agents – are currently very efficient as showcased in recent work (Gervet et al., 2022; Chang et al., 2023). The right combination between strong general neural modules and other non-neural blocks indeed allows reaching great performance, even in real-world environments. As of today, integrating knowledge priors about building blocks required to solve certain sub-tasks thus appears as a great short-term solution. However, one might wonder whether this might not constrain the flexibility and generality of neural agents in the long term.

End-to-end agents – on the other hand offer more flexibility as every required sub-skill will be learned to solve a downstream task. This can come with currently known caveats such as sample complexity, shortcut learning, or overfitting to specificities of observations (e.g. RGB, depth) in simulators. However, such end-to-end agents might be a great choice in the long term when the field progresses. Interesting findings showed the implicit emergence of map-like structures in the hidden memory of recurrent end-to-end agents (Wijmans et al., 2022), opening future perspectives. Such implicit phenomenons should be studied in more detail as they could help us better understand the way end-to-end agents learn to represent their environments. More than this, we could even want to guide agents in learning high-quality representations of environments, as shown in our chapter 4. Indeed, prior knowledge does not have to only support the design of the neural architecture but also the training signal. A richer training signal could allow to keep a simple end-to-end architecture but make it more sample efficient and help it better represent its past experience. Finally, very recent work (Bono et al., 2023a; Khanna et al., 2024) draws conclusions that seem to contradict the story about modular policies being generally superior to end-to-end agents in terms of navigation performance and generalization. Indeed, both works present end-to-end agents outperforming modular counterparts, opening new directions: in particular, Bono et al. (2023a) leverage a pre-trained

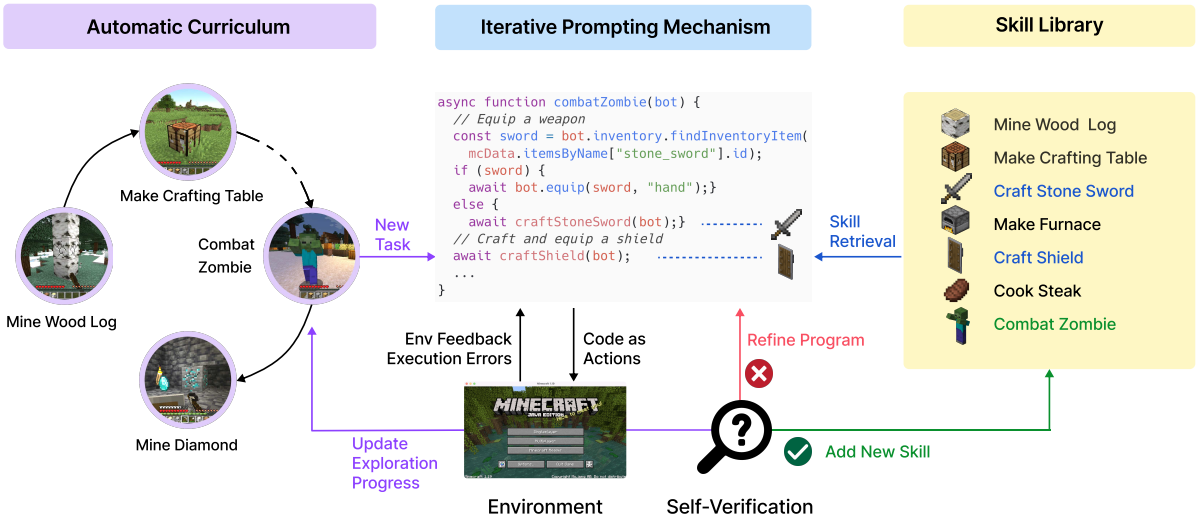


Figure 8.3: Voyager – reproduced from Wang et al. (2023)

vision backbone that is only adapted along with the policy being trained and showcase state-of-the-art performance on the *InstanceImageNav* task.

8.3 Other future perspectives in Embodied AI

The Embodied AI space is broad, and we have obviously not covered all of it in this manuscript. We will now review two exciting work directions that have received attention recently and will likely continue in the coming years.

8.3.1 *Intrinsic curiosity*

The ability to autonomously explore the world to improve its understanding is an exciting direction to pursue. It has already been shown (Pathak et al., 2017) that agents could learn to visit environments following a curiosity metric based on the inability to predict the future. Such an approach allows to also relegate data collection to the neural agents themselves, truly interacting with the world to self-improve. Self-supervised learning in 3D environments has already been studied (Chaplot et al., 2021), even following curiosity-based metrics (Chaplot et al., 2020c), but still has a lot to offer. A relevant recent work in this domain is *Voyager* (Wang et al., 2023), an agent based on a Large Language Model (LLM) that autonomously explores the world of *Minecraft* and acquires new sub-skills that are stored to be used again later (Figure 8.3). Here we see the tight connection between such lifelong learning, intrinsic curiosity-driven exploration, and the discovery of and adaptation to new tasks of interest.

8.3.2 *Experimenting in the real world*

Real-world experimentation is another important direction in the Embodied AI field. This is indeed a way to verify how well current agents can behave in real conditions, and more

importantly, study the potential differences between simulation and the real world (*sim-to-real gap*). There have been works trying to better understand and even quantify such gap (Kadian et al., 2020; Anderson et al., 2021; Sadek et al., 2022; Truong et al., 2023a), and more will likely come as it is an important problem. More generally, evaluating neural-based systems in the real world has been done in the recent past (Gervet et al., 2022; Chang et al., 2023; Krantz et al., 2023; Sadek et al., 2023; Bono et al., 2024) and this trend will probably continue further. An example is the introduction of new tasks such as *HomeRobot* (Yenamandra et al., 2023) that explicitly evaluates the ability of agents to perform open-vocabulary mobile manipulation on a real platform. Such effort allows comparing state-of-the-art methods based on their real-world performance under a common setup.

While it has not been presented in this manuscript, we have also performed experiments related to the deployment of modular exploration policies similar to the ones from chapter 6 in a house-scale scene. Figure 8.4 presents a sample rollout of a modular policy deployed on a *HelloRobot Stretch* robot (Kemp et al., 2022). The platform is composed of an RGB-D camera and a 2D LIDAR, the latter being used for localization and collision avoidance. RGB frames are used to perform semantic segmentation as shown in the first column of Figure 8.4 (RGB input + Mask R-CNN semantic mask output). Depth frames are converted to 3D point clouds that will allow building the obstacle and semantic map (second column of Figure 8.4). Finally, the third column in Figure 8.4 shows 3rd person views of the agent from external cameras. This information is not accessible to the policy but only serves visualization purposes. As can be seen, the agent properly covers the scene, visiting the different rooms, and builds a relatively complete semantic map of objects encountered along the way, highlighting the robustness of modular policies when deployed in the real world, echoing previous work (Gervet et al., 2022; Chang et al., 2023).

Deploying policies on real platforms is an important requirement to assess their abilities to interact with the real world we, humans, live in. Important engineering efforts should thus be spent to provide common robotic platforms allowing different research teams to reproduce experiments and compare algorithms rigorously. An example is the recent *LeRobot* project [2] from *Huggingface* aiming at providing a common framework to deploy neural policies on real robots.

8.4 Closing remarks

As already mentioned, robustly behaving in the 3D world to tackle useful tasks is a hard goal to achieve. However, the Embodied AI community has followed interesting directions in the past years and many perspectives are identifiable. An important challenge will be to make neural agents robust to different scenarios, environments and conditions, but more importantly able to adapt efficiently to new tasks and acquire the right sub-skills. Learn-

[2] https://github.com/huggingface/lerobot

ing by interacting is a great paradigm from a scientific point of view as neuroscience seems to indicate this is how we, humans, learn some of our skills. More than being an exciting research direction, action-based learning might even prove to be necessary to build proper world models.

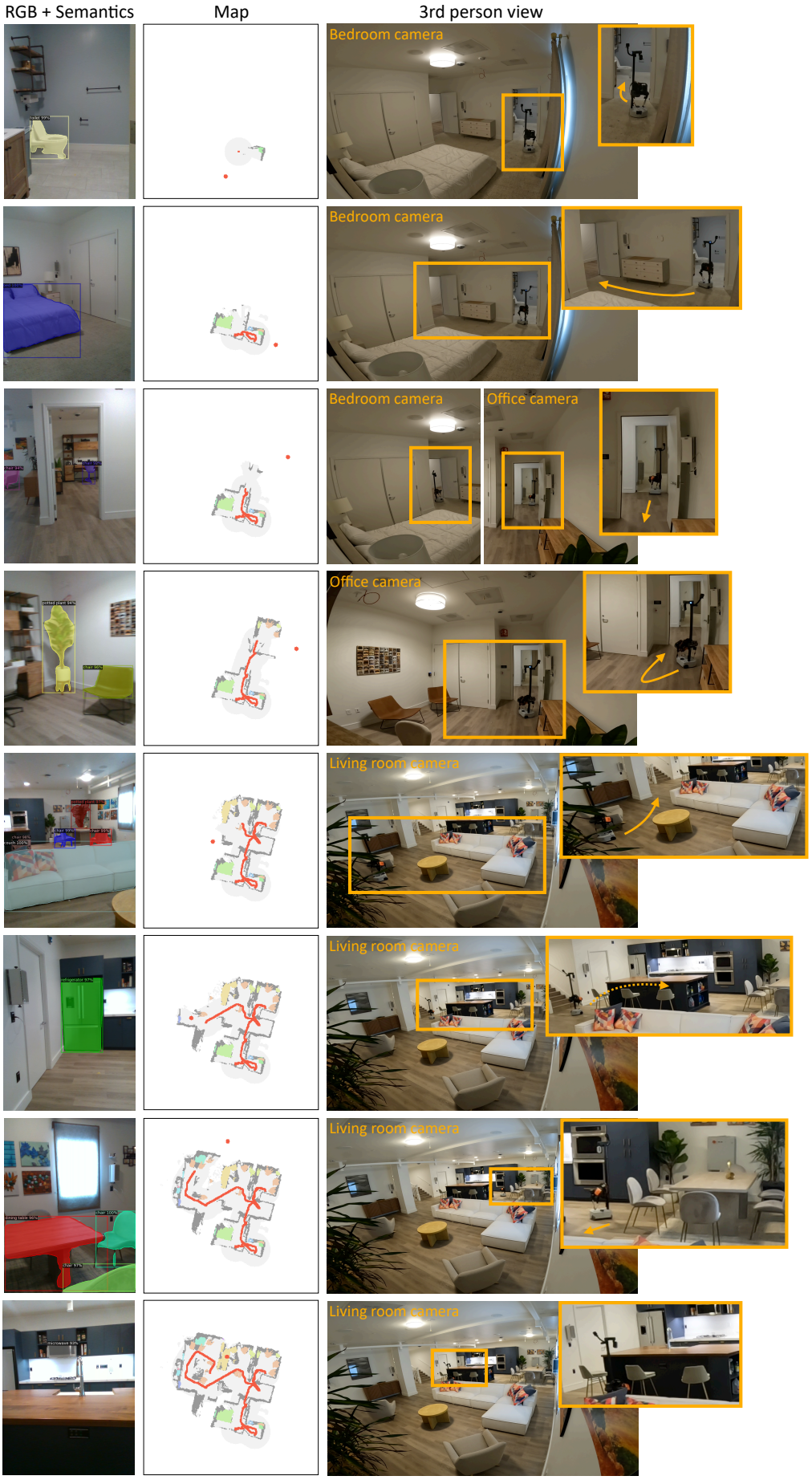


Figure 8.4: Deployment of a modular exploration policy on a *HelloRobot Stretch* robot (work done as a PhD intern at Meta AI in 2022).

Bibliography

Adamkiewicz, M., Chen, T., Caccavale, A., Gardner, R., Culbertson, P., Bohg, J., and Schwager, M. (2022). Vision-only robot navigation in a neural radiance world. *Robotics and Automation Letters*.

Anderson, P., Chang, A. X., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., and Zamir, A. R. (2018a). On evaluation of embodied navigation agents. *arXiv preprint*.

Anderson, P., Shrivastava, A., Truong, J., Majumdar, A., Parikh, D., Batra, D., and Lee, S. (2021). Sim-to-real transfer for vision-and-language navigation. In *Conference on Robot Learning*.

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and Van Den Hengel, A. (2018b). Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Conference on Computer Vision and Pattern Recognition*.

Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Zitnick, C. L., and Parikh, D. (2015). Vqa: Visual question answering. In *International Conference on Computer Vision*.

Asri, Z. E., Sigaud, O., and Thome, N. (2024). Physics-informed model and hybrid planning for efficient dyna-style reinforcement learning. *arXiv preprint*.

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., Pritzel, A., Chadwick, M., Degris, T., Modayil, J., Wayne, G., Soyer, H., Viola, F., Zhang, B., Goroshin, R., Rabinowitz, N., Pascanu, R., Beattie, C., Petersen, S., Sadik, A., Gaffney, S., King, H., Kavukcuoglu, K., Hassabis, D., Hadsell, R., and Kumaran, D. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature*.

Bardes, A., Ponce, J., and LeCun, Y. (2022). Vicreg: Variance-invariance-covariance regularization for self-supervised learning. In *International Conference on Learning Representations*.

Batra, D., Chang, A. X., Chernova, S., Davison, A. J., Deng, J., Koltun, V., Levine, S., Malik, J.,

Mordatch, I., Mottaghi, R., et al. (2020). Rearrangement: A challenge for embodied ai. *arXiv preprint*.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. (2016). Deepmind lab. *arXiv preprint*.

Beeching, E., Dibangoye, J., Simonin, O., and Wolf, C. (2020a). Deep reinforcement learning on a budget: 3d control and reasoning without a supercomputer. In *International Conference on Pattern Recognition*.

Beeching, E., Dibangoye, J., Simonin, O., and Wolf, C. (2020b). Egomap: Projective mapping and structured egocentric memory for deep RL. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*.

Beeching, E., Dibangoye, J., Simonin, O., and Wolf, C. (2020c). Learning to plan with uncertain topological maps. In *European Conference on Computer Vision*.

Bellman, R. (1957). A markovian decision process. *Journal of mathematics and mechanics*.

Ben-Younes, H., Cadene, R., Cord, M., and Thome, N. (2017). Mutan: Multimodal tucker fusion for visual question answering. In *International Conference on Computer Vision*.

Bhat, S. F., Alhashim, I., and Wonka, P. (2021). Adabins: Depth estimation using adaptive bins. In *Conference on Computer Vision and Pattern Recognition*.

Bonin-Font, F., Ortiz, A., and Oliver, G. (2008). Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*.

Bono, G., Antsfeld, L., Chidlovskii, B., Weinzaepfel, P., and Wolf, C. (2023a). End-to-end (instance)-image goal navigation through correspondence as an emergent phenomenon. In *International Conference on Learning Representations*.

Bono, G., Antsfeld, L., Sadek, A., Monaci, G., and Wolf, C. (2023b). Learning with a mole: Transferable latent spatial representations for navigation without reconstruction. In *International Conference on Learning Representations*.

Bono, G., Poirier, H., Antsfeld, L., Monaci, G., Chidlovskii, B., and Wolf, C. (2024). Learning to navigate efficiently and precisely in real environments. In *Conference on Computer Vision and Pattern Recognition*.

Borenstein, J., Koren, Y., et al. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *Transactions on robotics and automation*.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *International Conference on Computational Statistics*.

Bresson, G., Alsayed, Z., Yu, L., and Glaser, S. (2017). Simultaneous localization and mapping: A survey of current trends in autonomous driving. *Transactions on Intelligent Vehicles*.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., et al. (2022). Rt-1: Robotics transformer for real-world control at scale. *arXiv preprint*.

Buschman, T. J. and Miller, E. K. (2007). Top-down versus bottomup control of attention in the prefrontal and posterior parietal cortices. *Science*.

Cadene, R., Ben-Younes, H., Cord, M., and Thome, N. (2019). Murel: Multimodal relational reasoning for visual question answering. In *Conference on Computer Vision and Pattern Recognition*.

Campbell, M., Hoane Jr, A. J., and Hsu, F.-h. (2002). Deep blue. *Artificial intelligence*.

Campos, C., Elvira, R., Rodríguez, J. J. G., Montiel, J. M., and Tardós, J. D. (2021). Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *Transactions on robotics*.

Caron, M., Misra, I., Mairal, J., Goyal, P., Bojanowski, P., and Joulin, A. (2020). Unsupervised learning of visual features by contrasting cluster assignments. In *Neural Information Processing Systems*.

Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. (2021). Emerging properties in self-supervised vision transformers. In *International Conference on Computer Vision*.

Chan, E. R., Monteiro, M., Kellnhofer, P., Wu, J., and Wetzstein, G. (2021). pi-gan: Periodic implicit generative adversarial networks for 3d-aware image synthesis. In *Conference on Computer Vision and Pattern Recognition*.

Chane-Sane, E., Schmid, C., and Laptev, I. (2023). Learning video-conditioned policies for unseen manipulation tasks. In *International Conference on Robotics and Automation*.

Chang, A., Dai, A., Funkhouser, T., Halber, M., Niebner, M., Savva, M., Song, S., Zeng, A., and Zhang, Y. (2018). Matterport3d: Learning from rgb-d data in indoor environments. In *International Conference on 3D Vision*.

Chang, M., Gervet, T., Khanna, M., Yenamandra, S., Shah, D., Min, S. Y., Shah, K., Paxton, C., Gupta, S., Batra, D., et al. (2023). Goat: Go to any thing. *arXiv preprint*.

Chaplot, D. S., Dalal, M., Gupta, S., Malik, J., and Salakhutdinov, R. R. (2021). Seal: Self-supervised embodied active learning using exploration and 3d consistency. In *Neural Information Processing Systems*.

Chaplot, D. S., Gandhi, D., Gupta, A., and Salakhutdinov, R. (2020a). Object goal navigation using goal-oriented semantic exploration. In *Neural Information Processing Systems*.

Chaplot, D. S., Gandhi, D., Gupta, S., Gupta, A., and Salakhutdinov, R. (2020b). Learning to explore using active neural slam. In *International Conference on Learning Representations*.

Chaplot, D. S., Jiang, H., Gupta, S., and Gupta, A. (2020c). Semantic curiosity for active visual learning. In *European Conference on Computer Vision*.

Chaplot, D. S., Salakhutdinov, R., Gupta, A., and Gupta, S. (2020d). Neural topological slam for visual navigation. In *Conference on Computer Vision and Pattern Recognition*.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. (2021a). Decision transformer: Reinforcement learning via sequence modeling. In *Neural Information Processing Systems*.

Chen, S., Ge, C., Tong, Z., Wang, J., Song, Y., Wang, J., and Luo, P. (2022a). Adaptformer: Adapting vision transformers for scalable visual recognition. In *Neural Information Processing Systems*.

Chen, S., Guhur, P.-L., Schmid, C., and Laptev, I. (2021b). History aware multimodal transformer for vision-and-language navigation. In *Neural Information Processing Systems*.

Chen, S., Guhur, P.-L., Tapaswi, M., Schmid, C., and Laptev, I. (2022b). Think Global, Act Local: Dual-scale Graph Transformer for Vision-and-Language Navigation. In *Conference on Computer Vision and Pattern Recognition*.

Chen, T., Gupta, S., and Gupta, A. (2018). Learning exploration policies for navigation. In *International Conference on Learning Representations*.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning*.

Chen, X., Fang, H., Lin, T.-Y., Vedantam, R., Gupta, S., Dollár, P., and Zitnick, C. L. (2015). Microsoft coco captions: Data collection and evaluation server. *arXiv preprint*.

Chen, Z. and Zhang, H. (2019). Learning implicit fields for generative shape modeling. In *Conference on Computer Vision and Pattern Recognition*.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase repr. using RNN encoder–decoder for statistical machine translation. In *Empirical Methods in Natural Language Processing*.

Chollet, F. (2019). On the measure of intelligence. *arXiv preprint*.

Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. (2016). 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European Conference on Computer Vision*.

Chung, C.-M., Tseng, Y.-C., Hsu, Y.-C., Shi, X.-Q., Hua, Y.-H., Yeh, J.-F., Chen, W.-C., Chen, Y.-T., and Hsu, W. H. (2023). Orbeez-slam: A real-time monocular visual slam with orb features and nerf-realized mapping. In *International Conference on Robotics and Automation*.

Corbetta, M. and Shulman, G. L. (2002). Control of goal-directed and stimulus-driven attention in the brain. *Nature Reviews Neuroscience*.

Csáji, B. C. et al. (2001). Approximation with artificial neural networks. *Faculty of Sciences, Etvs Lornd University, Hungary*.

Cueva, C. and Wei, X.-X. (2018). Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. In *International Conference on Learning Representations*.

Dai, A., Chang, A. X., Savva, M., Halber, M., Funkhouser, T., and Nießner, M. (2017). Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Conference on Computer Vision and Pattern Recognition*.

Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *Conference on Computer Vision and Pattern Recognition*.

Dechter, R. and Pearl, J. (1985). Generalized best-first search strategies and the optimality of a. *Journal of the ACM*.

Deitke, M., Batra, D., Bisk, Y., Campari, T., Chang, A. X., Chaplot, D. S., Chen, C., D'Arpino, C. P., Ehsani, K., Farhadi, A., et al. (2022). Retrospectives on the embodied ai workshop. *arXiv preprint*.

Deitke, M., Han, W., Herrasti, A., Kembhavi, A., Kolve, E., Mottaghi, R., Salvador, J., Schwenk, D., VanderBilt, E., Wallingford, M., et al. (2020). Robothor: An open simulation-to-real embodied ai platform. In *Conference on Computer Vision and Pattern Recognition*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*.

D'Eramo, C. and Chalvatzaki, G. (2022). Prioritized sampling with intrinsic motivation in multi-task reinforcement learning. In *International Joint Conference on Neural Networks*.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2019). BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Annual Conference of the North American Chapter of the Association for Computational Linguistics*.

DeVries, T., Bautista, M. A., Srivastava, N., Taylor, G. W., and Susskind, J. M. (2021). Unconstrained scene generation with locally conditioned radiance fields. In *International Conference on Computer Vision*.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*.

Ding, Y., Florensa, C., Abbeel, P., and Phielipp, M. (2019). Goal-conditioned imitation learning. In *Neural Information Processing Systems*.

Donoho, D. (2017). 50 years of data science. *Journal of Computational and Graphical Statistics*.

Dornhege, C. and Kleiner, A. (2013). A frontier-void-based approach for autonomous exploration in 3d. *Advanced Robotics*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*.

Du, H., Yu, X., and Zheng, L. (2021). VTNet: Visual Transformer Network for Object Goal Navigation. In *International Conference on Learning Representations*.

Dupont, E., Kim, H., Eslami, S. A., Rezende, D. J., and Rosenbaum, D. (2022). From data to functa: Your data point is a function and you can treat it like one. In *International Conference on Machine Learning*.

Ehsani, K., Han, W., Herrasti, A., VanderBilt, E., Weihs, L., Kolve, E., Kembhavi, A., and Mottaghi, R. (2021). Manipulathor: A framework for visual object manipulation. In *Conference on Computer Vision and Pattern Recognition*.

Eigen, D. and Fergus, R. (2015). Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *International Conference on Computer Vision*.

Ekstrom, A. D., Spiers, H. J., Bohbot, V. D., and Rosenbaum, R. S. (2018). *Human spatial navigation*. Princeton University Press.

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*.

Fan, H., Su, H., and Guibas, L. J. (2017). A point set generation network for 3d object reconstruction from a single image. In *Conference on Computer Vision and Pattern Recognition*.

Fang, K., Toshev, A., Fei-Fei, L., and Savarese, S. (2019). Scene memory transformer for embodied agents in long-horizon tasks. In *Conference on Computer Vision and Pattern Recognition*.

Fridovich-Keil, S., Yu, A., Tancik, M., Chen, Q., Recht, B., and Kanazawa, A. (2022). Plenox-

els: Radiance fields without neural networks. In *Conference on Computer Vision and Pattern Recognition*.

Fu, Z., Zhao, T. Z., and Finn, C. (2024). Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation. *arXiv preprint*.

Gal, Y. and Ghahramani, Z. (2014). Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*.

Gan, C., Schwartz, J., Alter, S., Mrowca, D., Schrimpf, M., Traer, J., De Freitas, J., Kubilius, J., Bhandwaldar, A., Haber, N., et al. (2021). Threedworld: A platform for interactive multi-modal physical simulation. In *NeurIPS Datasets and Benchmarks Track*.

Garbin, S. J., Kowalski, M., Johnson, M., Shotton, J., and Valentin, J. (2021). Fastnerf: High-fidelity neural rendering at 200fps. In *International Conference on Computer Vision*.

Garrido, S., Moreno, L., Abderrahim, M., and Martin, F. (2006). Path planning for mobile robot navigation using voronoi diagram and fast marching. In *International Conference on Intelligent Robots and Systems*.

Gervet, T., Chintala, S., Batra, D., Malik, J., and Chaplot, D. S. (2022). Navigating to objects in the real world. *Science Robotics*.

Gibson, J. J. (1966). The senses considered as perceptual systems.

Gibson, J. J. (1979). The ecological approach to visual perception.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. In *International Conference on Machine Learning*.

Goodfellow, I. J., Mirza, M., Xiao, D., Courville, A., and Bengio, Y. (2014). An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *International Conference on Learning Representations*.

Gottlieb, J., Oudeyer, P.-Y., Lopes, M., and Baranes, A. (2013). Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in cognitive sciences*.

Grill, J.-B., Strub, F., Altché, F., Tallec, C., Richemond, P., Buchatskaya, E., Doersch, C., Avila Pires, B., Guo, Z., Gheshlaghi Azar, M., et al. (2020). Bootstrap your own latent-a new approach to self-supervised learning. In *Neural Information Processing Systems*.

Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J. (2017). Cognitive mapping and planning for visual navigation. In *Conference on Computer Vision and Pattern Recognition*.

Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks. *arXiv preprint*.

Ha, D. and Schmidhuber, J. (2018). World models. *arXiv preprint*.

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature*.

Hahn, M., Chaplot, D. S., Tulsiani, S., Mukadam, M., Rehg, J. M., and Gupta, A. (2021). No rl, no simulation: Learning to navigate without navigating. In *Neural Information Processing Systems*.

Hanocka, R., Hertz, A., Fish, N., Giryes, R., Fleishman, S., and Cohen-Or, D. (2019). Meshcnn: a network with an edge. *Transactions on Graphics*.

Hansen, N., Su, H., and Wang, X. (2024). Td-mpc2: Scalable, robust world models for continuous control. In *International Conference on Learning Representations*.

Hao, W., Li, C., Li, X., Carin, L., and Gao, J. (2020). Towards learning a generic agent for vision-and-language navigation via pre-training. In *Conference on Computer Vision and Pattern Recognition*.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *Transactions on Systems Science and Cybernetics*.

Hart, P. E., Nilsson, N. J., and Raphael, B. (1972). Correction to" a formal basis for the heuristic determination of minimum cost paths". *SIGART Bulletin*.

Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable mdps. In *AAAI Conference on Artificial Intelligence*.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Conference on Computer Vision and Pattern Recognition*.

He, K., Fan, H., Wu, Y., Xie, S., and Girshick, R. (2020). Momentum contrast for unsupervised visual representation learning. In *Conference on Computer Vision and Pattern Recognition*.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *International Conference on Computer Vision*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*.

Henriques, J. F. and Vedaldi, A. (2018). Mapnet: An allocentric spatial memory for mapping environments. In *Conference on Computer Vision and Pattern Recognition*.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint*.

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint*.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. (2022). Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Hu, E. J., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., Chen, W., et al. (2021). Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Hu, H., Zhang, Z., Xie, Z., and Lin, S. (2019). Local relation networks for image recognition. In *International Conference on Computer Vision*.

Huang, H., Li, L., Cheng, H., and Yeung, S.-K. (2024). Photo-slam: Real-time simultaneous localization and photorealistic mapping for monocular stereo and rgb-d cameras. In *Conference on Computer Vision and Pattern Recognition*.

Huang, Z., Wang, X., Huang, L., Huang, C., Wei, Y., and Liu, W. (2019). Ccnet: Criss-cross attention for semantic segmentation. In *International Conference on Computer Vision*.

Huh, M., Cheung, B., Wang, T., and Isola, P. (2024). Position: The platonic representation hypothesis. In *International Conference on Machine Learning*.

Iyer, A., Peng, Z., Dai, Y., Guzey, I., Haldar, S., Chintala, S., and Pinto, L. (2024). Open teach: A versatile teleoperation system for robotic manipulation. *arXiv preprint*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement learning with unsupervised auxiliary tasks. In *International Conference on Learning Representations*.

James, S., Ma, Z., Arrojo, D. R., and Davison, A. J. (2020). Rlbench: The robot learning benchmark & learning environment. *Robotics and Automation Letters*.

James, W., Burkhardt, F., Bowers, F., and Skrupskelis, I. K. (1890). *The principles of psychology*. Macmillan London.

Jang, E., Irpan, A., Khansari, M., Kappler, D., Ebert, F., Lynch, C., Levine, S., and Finn, C. (2022). Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*.

Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S. (2019). Predicting the generalization gap in deep networks with margin distributions. In *International Conference on Learning Representations*.

Kadian, A., Truong, J., Gokaslan, A., Clegg, A., Wijmans, E., Lee, S., Savva, M., Chernova, S., and Batra, D. (2020). Sim2real predictivity: Does evaluation in simulation predict real-world performance? *Robotics and Automation Letters*.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial intelligence*.

Kajiya, J. T. and Von Herzen, B. P. (1984). Ray tracing volume densities. *SIGGRAPH computer graphics*.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*.

Keetha, N., Karhade, J., Jatavallabhula, K. M., Yang, G., Scherer, S., Ramanan, D., and Luiten, J. (2024). Splatam: Splat track & map 3d gaussians for dense rgb-d slam. In *Conference on Computer Vision and Pattern Recognition*.

Kemp, C. C., Edsinger, A., Clever, H. M., and Matulevich, B. (2022). The design of stretch: A compact, lightweight mobile manipulator for indoor human environments. In *International Conference on Robotics and Automation*.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. (2016). Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *C. on Comp. Intelligence and Games*.

Kerbl, B., Kopanas, G., Leimkühler, T., and Drettakis, G. (2023). 3d gaussian splatting for real-time radiance field rendering. *Transactions on Graphics*.

Kervadec, C., Jaunet, T., Antipov, G., Baccouche, M., Vuillemot, R., and Wolf, C. (2021). How Transferrable are Reasoning Patterns in VQA? In *Conference on Computer Vision and Pattern Recognition*.

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., and Shah, M. (2022). Transformers in vision: A survey. *ACM computing surveys*.

Khandelwal, A., Weihs, L., Mottaghi, R., and Kembhavi, A. (2022). Simple but effective: CLIP embeddings for embodied AI. In *Conference on Computer Vision and Pattern Recognition*.

Khanna, M., Ramrakhya, R., Chhablani, G., Yenamandra, S., Gervet, T., Chang, M., Kira, Z., Chaplot, D. S., Batra, D., and Mottaghi, R. (2024). Goat-bench: A benchmark for multi-modal lifelong navigation. In *Conference on Computer Vision and Pattern Recognition*.

Kim, D., Ka, W., Ahn, P., Joo, D., Chun, S., and Kim, J. (2022). Global-local path networks for monocular depth estimation with vertical cutdepth. *arXiv preprint*.

Kim, N., Kwon, O., Yoo, H., Choi, Y., Park, J., and Oh, S. (2023). Topological semantic graph memory for image-goal navigation. In *Conference on Robot Learning*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint*.

Kobyzev, I., Prince, S. J., and Brubaker, M. A. (2021). Normalizing flows: An introduction and review of current methods. *Transactions on Pattern Analysis and Machine Intelligence*.

Kolve, E., Mottaghi, R., Han, W., VanderBilt, E., Weihs, L., Herrasti, A., Deitke, M., Ehsani, K., Gordon, D., Zhu, Y., et al. (2017). Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint*.

Konolige, K. and Chou, K. (1999). Markov localization using correlation. In *International Joint Conference on Artificial Intelligence*.

Kortenkamp, D., Bonasso, R., and Murphy, R. (1998). Ai-based mobile robots: Case studies of successful robot systems.

Kortenkamp, D. and Weymouth, T. (1994). Topological mapping for mobile robots using a combination of sonar and vision sensing. In *AAAI Conference on Artificial Intelligence*.

Krantz, J., Gervet, T., Yadav, K., Wang, A., Paxton, C., Mottaghi, R., Batra, D., Malik, J., Lee, S., and Chaplot, D. S. (2023). Navigating to objects specified by images. In *International Conference on Computer Vision*.

Krantz, J., Lee, S., Malik, J., Batra, D., and Chaplot, D. S. (2022). Instance-specific image goal navigation: Training embodied agents to find object instances. *arXiv preprint*.

Krantz, J., Wijmans, E., Majumdar, A., Batra, D., and Lee, S. (2020). Beyond the nav-graph: Vision-and-language navigation in continuous environments. In *European Conference on Computer Vision*.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2017). Imagenet classification with deep convolutional neural networks. *Communications of the ACM*.

Kroemer, O., Niekum, S., and Konidaris, G. (2021). A review of robot learning for manipulation: Challenges, representations, and algorithms. *Journal of Machine Learning Research*.

Kwon, O., Park, J., and Oh, S. (2023). Renderable neural radiance map for visual navigation. In *Conference on Computer Vision and Pattern Recognition*.

Lample, G. and Chaplot, D. S. (2017). Playing fps games with deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*.

Lassner, C. and Zollhofer, M. (2021). Pulsar: Efficient sphere-based neural rendering. In *Conference on Computer Vision and Pattern Recognition*.

LaValle, S. M. (2006). *Planning algorithms*. Cambridge university press.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *IEEE*.

Legg, S., Hutter, M., et al. (2007). A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and applications*.

Lenssen, J. E., Osendorfer, C., and Masci, J. (2020). Deep iterative surface normal estimation. In *Conference on Computer Vision and Pattern Recognition*.

Lesort, T., Díaz-Rodríguez, N., Goudou, J.-F., and Filliat, D. (2018). State representation learning for control: An overview. *Neural Networks*.

Levoy, M. (1990). Efficient ray tracing of volume data. *Transactions on Graphics*.

Li, A. C., Prabhudesai, M., Duggal, S., Brown, E., and Pathak, D. (2023). Your diffusion model is secretly a zero-shot classifier. In *International Conference on Computer Vision*.

Li, X., De Mello, S., Wang, X., Yang, M.-H., Kautz, J., and Liu, S. (2022a). Learning Continuous Environment Fields via Implicit Functions. In *International Conference on Learning Representations*.

Li, Y., Li, S., Sitzmann, V., Agrawal, P., and Torralba, A. (2022b). 3d neural scene representations for visuomotor control. In *Conference on Robot Learning*.

Li, Z., Liu, X., Drenkow, N., Ding, A., Creighton, F. X., Taylor, R. H., and Unberath, M. (2021). Revisiting stereo depth estimation from a sequence-to-sequence perspective with transformers. In *International Conference on Computer Vision*.

Liang, A., Singh, I., Pertsch, K., and Thomason, J. (2022). Transformer adapters for robot learning. In *CoRL Workshop on Pre-training Robot Learning*.

Lin, C.-H., Ma, W.-C., Torralba, A., and Lucey, S. (2021). Barf: Bundle-adjusting neural radiance fields. In *International Conference on Computer Vision*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European Conference on Computer Vision*.

Lindenberger, P., Sarlin, P.-E., and Pollefeys, M. (2023). Lightglue: Local feature matching at light speed. In *International Conference on Computer Vision*.

Lorensen, W. E. and Cline, H. E. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH computer graphics*.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*.

Ma, Y. J., Sodhani, S., Jayaraman, D., Bastani, O., Kumar, V., and Zhang, A. (2022). Vip: Towards universal visual reward and representation via value-implicit pre-training. In *International Conference on Learning Representations*.

Macario Barros, A., Michel, M., Moline, Y., Corre, G., and Carrel, F. (2022). A comprehensive survey of visual slam algorithms. *Robotics*.

Majumdar, A., Yadav, K., Arnaud, S., Ma, Y. J., Chen, C., Silwal, S., Jain, A., Berges, V.-P., Abbeel, P., Malik, J., et al. (2023). Where are we in the search for an artificial visual cortex for embodied intelligence? *arXiv preprint*.

Marchand, E. (2020). Direct visual servoing in the frequency domain. *Robotics and Automation Letters*.

Martin, C. H. and Mahoney, M. W. (2020). Heavy-tailed universality predicts trends in test accuracies for very large pre-trained deep neural networks. In *International Conference on Data Mining*.

Martin, C. H., Peng, T. S., and Mahoney, M. W. (2021). Predicting trends in the quality of state-of-the-art neural networks without access to training or testing data. *Nature Communications*.

Marza, P., Matignon, L., Simonin, O., Batra, D., Wolf, C., and Chaplot, D. S. (2024a). Autonerf: Training implicit scene representations with autonomous agents. In *International Conference on Intelligent Robots and Systems*.

Marza, P., Matignon, L., Simonin, O., and Wolf, C. (2022). Teaching agents how to map:

Spatial reasoning for multi-object navigation. In *International Conference on Intelligent Robots and Systems*.

Marza, P., Matignon, L., Simonin, O., and Wolf, C. (2023). Multi-object navigation with dynamically learned neural implicit representations. In *International Conference on Computer Vision*.

Marza, P., Matignon, L., Simonin, O., and Wolf, C. (2024b). Task-conditioned adaptation of visual features in multi-task policy learning. In *Conference on Computer Vision and Pattern Recognition*.

Matsuki, H., Murai, R., Kelly, P. H., and Davison, A. J. (2024). Gaussian splatting slam. In *Conference on Computer Vision and Pattern Recognition*.

McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*.

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy networks: Learning 3d reconstruction in function space. In *Conference on Computer Vision and Pattern Recognition*.

Mezghani, L., Bojanowski, P., Alahari, K., and Sukhbaatar, S. (2023a). Think before you act: Unified policy for interleaving language reasoning with actions. In *Workshop on Reincarnating Reinforcement Learning (ICLR)*.

Mezghani, L., Sukhbaatar, S., Bojanowski, P., Lazaric, A., and Alahari, K. (2023b). Learning goal-conditioned policies offline with self-supervised reward shaping. In *Conference on Robot Learning*.

Mezghani, L., Sukhbaatar, S., Lavril, T., Maksymets, O., Batra, D., Bojanowski, P., and Alahari, K. (2022). Memory-augmented reinforcement learning for image-goal navigation. In *International Conference on Intelligent Robots and Systems*.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*.

Min, S. Y., Chaplot, D. S., Ravikumar, P., Bisk, Y., and Salakhutdinov, R. (2022). Film: Following instructions in language with modular methods. In *International Conference on Learning Representations*.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., Kumaran, D., and Hadsell, R. (2017). Learning to navigate in complex environments. In *International Conference on Learning Representations*.

Mishkin, D., Dosovitskiy, A., and Koltun, V. (2019). Benchmarking classic and learned navigation in complex 3d environments. *arXiv preprint*.

Mitchell, T. M. (1980). The need for biases in learning generalizations. *Rutgers University*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*.

Moravec, H. (1988a). *Mind children: The future of robot and human intelligence*. Harvard University Press.

Moravec, H. (1988b). Sensor fusion in certainty grids for mobile robots. *AI magazine*.

Müller, T., Evans, A., Schied, C., and Keller, A. (2022). Instant neural graphics primitives with a multiresolution hash encoding. *Transactions on Graphics*.

Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015). Orb-slam: a versatile and accurate monocular slam system. *Transactions on robotics*.

Mur-Artal, R. and Tardós, J. D. (2017). Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *Transactions on robotics*.

Nair, S., Rajeswaran, A., Kumar, V., Finn, C., and Gupta, A. (2023). R3m: A universal visual representation for robot manipulation. In *Conference on Robot Learning*.

Newcombe, R. A., Lovegrove, S. J., and Davison, A. J. (2011). Dtam: Dense tracking and mapping in real-time. In *International Conference on Computer Vision*.

Oh, J., Chockalingam, V., Satinder, and Lee, H. (2016). Control of memory, active perception, and action in minecraft. In *International Conference on Machine Learning*.

Oquab, M., Darcet, T., Moutakanni, T., Vo, H., Szafraniec, M., Khalidov, V., Fernandez, P., Haziza, D., Massa, F., El-Nouby, A., et al. (2023). Dinov2: Learning robust visual features without supervision. *arXiv preprint*.

Ortiz, J., Clegg, A., Dong, J., Sucar, E., Novotny, D., Zollhoefer, M., and Mukadam, M. (2022). iSDF: Real-Time Neural Signed Distance Fields for Robot Perception. *arXiv preprint*.

Oudeyer, P.-Y. and Kaplan, F. (2007). What is intrinsic motivation? a typology of computational approaches. *Frontiers in neurorobotics*.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic motivation systems for autonomous mental development. *Transactions on evolutionary computation*.

Pan, S., Brunton, S. L., and Kutz, J. N. (2023). Neural implicit flow: a mesh-agnostic dimensionality reduction paradigm of spatio-temporal data. *Journal of Machine Learning Research*.

Pan, X., Lai, Z., Song, S., and Huang, G. (2022). Activenerf: Learning where to see with uncertainty estimation. In *European Conference on Computer Vision*.

Papert, S. A. (1966). The summer vision project.

Parisi, S., Rajeswaran, A., Purushwalkam, S., and Gupta, A. (2022). The unsurprising effectiveness of pre-trained vision models for control. In *International Conference on Machine Learning*.

Parisotto, E. and Salakhutdinov, R. (2018). Neural map: Structured memory for deep reinforcement learning. In *International Conference on Learning Representations*.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. In *Conference on Computer Vision and Pattern Recognition*.

Parker, A. (2003). In the blink of an eye: how vision sparked the big bang of evolution.

Pashevich, A., Schmid, C., and Sun, C. (2021). Episodic transformer for vision-and-language navigation. In *International Conference on Computer Vision*.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning*.

Peer, M., Brunec, I. K., Newcombe, N. S., and Epstein, R. A. (2020). Structuring knowledge with cognitive maps and cognitive graphs. *Trends in Cognitive Sciences*.

Peng, S., Niemeyer, M., Mescheder, L., Pollefeys, M., and Geiger, A. (2020). Convolutional occupancy networks. In *European Conference on Computer Vision*.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *AAAI Conference on Artificial Intelligence*.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. (2021). Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*.

Pfeiffer, J., Kamath, A., Rücklé, A., Cho, K., and Gurevych, I. (2021). Adapterfusion: Non-destructive task composition for transfer learning. In *Conference of the European Chapter of the Association for Computational Linguistics*.

Podell, D., English, Z., Lacey, K., Blattmann, A., Dockhorn, T., Müller, J., Penna, J., and Rombach, R. (2024). Sdxl: Improving latent diffusion models for high-resolution image synthesis. In *International Conference on Learning Representations*.

Pomerleau, D. A. (1988). Alvinn: An autonomous land vehicle in a neural network. In *Neural Information Processing Systems*.

Puig, X., Undersander, E., Szot, A., Cote, M. D., Yang, T.-Y., Partsey, R., Desai, R., Clegg, A., Hlavac, M., Min, S. Y., et al. (2023). Habitat 3.0: A co-habitat for humans, avatars, and robots. In *International Conference on Learning Representations*.

Pumacay, W., Singh, I., Duan, J., Krishna, R., Thomason, J., and Fox, D. (2024). The colosseum: A benchmark for evaluating generalization for robotic manipulation. *arXiv preprint*.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017a). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Conference on Computer Vision and Pattern Recognition*.

Qi, C. R., Yi, L., Su, H., and Guibas, L. J. (2017b). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Neural Information Processing Systems*.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., et al. (2021). Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*.

Radosavovic, I., Xiao, T., James, S., Abbeel, P., Malik, J., and Darrell, T. (2023). Real-world robot learning with masked visual pre-training. In *Conference on Robot Learning*.

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*.

Ramachandran, P., Parmar, N., Vaswani, A., Bello, I., Levskaya, A., and Shlens, J. (2019). Stand-alone self-attention in vision models. In *Neural Information Processing Systems*.

Ramakrishnan, S. K., Chaplot, D. S., Al-Halah, Z., Malik, J., and Grauman, K. (2022). Poni: Potential functions for objectgoal navigation with interaction-free learning. In *Conference on Computer Vision and Pattern Recognition*.

Ramakrishnan, S. K., Gokaslan, A., Wijmans, E., Maksymets, O., Clegg, A., Turner, J. M., Undersander, E., Galuba, W., Westbury, A., Chang, A. X., et al. (2021a). Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai. In *NeurIPS Datasets and Benchmarks Track*.

Ramakrishnan, S. K., Jayaraman, D., and Grauman, K. (2021b). An exploration of embodied visual exploration. *International Journal of Computer Vision*.

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. (2021). Zero-shot text-to-image generation. In *International Conference on Machine Learning*.

Ramrakhya, R., Batra, D., Wijmans, E., and Das, A. (2023). Pirlnav: Pretraining with imitation and rl finetuning for objectnav. In *Conference on Computer Vision and Pattern Recognition*.

Ramrakhya, R., Undersander, E., Batra, D., and Das, A. (2022). Habitat-web: Learning embodied object-search strategies from human demonstrations at scale. In *Conference on Computer Vision and Pattern Recognition*.

Ran, Y., Zeng, J., He, S., Chen, J., Li, L., Chen, Y., Lee, G., and Ye, Q. (2023). Neurar: Neural uncertainty for autonomous 3d reconstruction with implicit neural representations. *Robotics and Automation Letters*.

Ranftl, R., Bochkovskiy, A., and Koltun, V. (2021). Vision transformers for dense prediction. In *International Conference on Computer Vision*.

Ranftl, R., Lasinger, K., Hafner, D., Schindler, K., and Koltun, V. (2020). Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *Transactions on Pattern Analysis and Machine Intelligence*.

Reed, S., Zolna, K., Parisotto, E., Colmenarejo, S. G., Novikov, A., Barth-Maron, G., Gimenez, M., Sulsky, Y., Kay, J., Springenberg, J. T., et al. (2022). A generalist agent. *Transactions on Machine Learning Research*.

Ritter, S., Faulkner, R., Sartran, L., Santoro, A., Botvinick, M., and Raposo, D. (2021). Rapid task-solving in novel environments. In *International Conference on Learning Representations*.

Rombach, R., Blattmann, A., Lorenz, D., Esser, P., and Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In *Conference on Computer Vision and Pattern Recognition*.

Rosenblatt, F. (1957). *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory.

Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*.

Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. In *International Conference on Computer Vision*.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. *Biometrika*.

Rummelhard, L., Nègre, A., and Laugier, C. (2015). Conditional Monte Carlo Dense Occupancy Tracker. In *International Conference on Intelligent Transportation Systems*.

Ruoss, A., Delétang, G., Medapati, S., Grau-Moya, J., Wenliang, L. K., Catt, E., Reid, J., and Genewein, T. (2024). Grandmaster-level chess without search. *arXiv preprint*.

Sadek, A., Bono, G., Chidlovskii, B., Baskurt, A., and Wolf, C. (2023). Multi-Object Navigation in real environments using hybrid policies. In *International Conference on Robotics and Automation*.

Sadek, A., Bono, G., Chidlovskii, B., and Wolf, C. (2022). An in-depth experimental study of sensor usage and visual reasoning of robots navigating in real environments. In *International Conference on Robotics and Automation*.

Saharia, C., Chan, W., Saxena, S., Li, L., Whang, J., Denton, E. L., Ghasemipour, K., Gontijo Lopes, R., Karagol Ayan, B., Salimans, T., et al. (2022). Photorealistic text-to-image diffusion models with deep language understanding. In *Neural Information Processing Systems*.

Sarlin, P.-E., DeTone, D., Malisiewicz, T., and Rabinovich, A. (2020). Superglue: Learning feature matching with graph neural networks. In *Conference on Computer Vision and Pattern Recognition*.

Sarlin, P.-E., Unagar, A., Larsson, M., Germain, H., Toft, C., Larsson, V., Pollefeys, M., Lepetit, V., Hammarstrand, L., Kahl, F., et al. (2021). Back to the feature: Learning robust camera localization from pixels to pose. In *Conference on Computer Vision and Pattern Recognition*.

Savinov, N., Dosovitskiy, A., and Koltun, V. (2018a). Semi-parametric topological memory for navigation. In *International Conference on Learning Representations*.

Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., and Gelly, S. (2018b). Episodic curiosity through reachability. In *International Conference on Learning Representations*.

Savva, M., Kadian, A., Maksymets, O., Zhao, Y., Wijmans, E., Jain, B., Straub, J., Liu, J., Koltun, V., Malik, J., Parikh, D., and Batra, D. (2019). Habitat: A platform for embodied ai research. In *International Conference on Computer Vision*.

Schiele, B. and Crowley, J. L. (1994). A comparison of position estimation techniques using occupancy grids. *Robotics and autonomous systems*.

Schneider, T., Belousov, B., Chalvatzaki, G., Romeres, D., Jha, D. K., and Peters, J. (2022). Active exploration for robotic manipulation. In *International Conference on Intelligent Robots and Systems*.

Schonberger, J. L. and Frahm, J.-M. (2016). Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition*.

Schuhmann, C., Beaumont, R., Vencu, R., Gordon, C., Wightman, R., Cherti, M., Coombes, T., Katta, A., Mullis, C., Wortsman, M., et al. (2022). Laion-5b: An open large-scale dataset for training next generation image-text models. In *Neural Information Processing Systems*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint*.

Sethian, J. A. (1996). A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*.

Shafiullah, N. M. M., Rai, A., Etukuru, H., Liu, Y., Misra, I., Chintala, S., and Pinto, L. (2023). On bringing robots home. *arXiv preprint*.

Shah, D., Eysenbach, B., Rhinehart, N., and Levine, S. (2021). Rapid Exploration for Open-World Navigation with Latent Goal Models. In *Conference on Robot Learning*.

Sharma, M., Fantacci, C., Zhou, Y., Koppula, S., Heess, N., Scholz, J., and Aytar, Y. (2022). Lossless adaptation of pretrained vision models for robotic manipulation. In *International Conference on Learning Representations*.

Shatkay, H. and Kaelbling, L. P. (1997). Learning topological maps with weak local odometric information. In *International Joint Conference on Artificial Intelligence*.

Shatkay, H. and Kaelbling, L. P. (2002). Learning geometrically-constrained hidden markov models for robot navigation: Bridging the topological-geometrical gap. *Journal of Artificial Intelligence Research*.

Shen, B., Xia, F., Li, C., Martín-Martín, R., Fan, L., Wang, G., Pérez-D'Arpino, C., Buch, S., Srivastava, S., Tchapmi, L., et al. (2021). igibson 1.0: a simulation environment for interactive tasks in large realistic scenes. In *International Conference on Intelligent Robots and Systems*.

Shridhar, M., Thomason, J., Gordon, D., Bisk, Y., Han, W., Mottaghi, R., Zettlemoyer, L., and Fox, D. (2020). Alfred: A benchmark for interpreting grounded instructions for everyday tasks. In *Conference on Computer Vision and Pattern Recognition*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint*.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint*.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. In *Neural Information Processing Systems*.

Smith, L. and Gasser, M. (2005). The development of embodied cognition: Six lessons from babies. *Artificial life*.

Smith, R. C. and Cheeseman, P. (1986). On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*.

Smith, S. L., Brock, A., Berrada, L., and De, S. (2023). Convnets match vision transformers at scale. *arXiv preprint*.

Smolyanskiy, N., Kamenev, A., and Birchfield, S. (2018). On the importance of stereo for accurate depth estimation: An efficient semi-supervised deep neural network approach. In *CVPR workshops*.

Soucek, T., Alayrac, J.-B., Miech, A., Laptev, I., and Sivic, J. (2024). Multi-task learning of object states and state-modifying actions from web videos. *Transactions on Pattern Analysis and Machine Intelligence*.

Straub, J., Whelan, T., Ma, L., Chen, Y., Wijmans, E., Green, S., Engel, J. J., Mur-Artal, R., Ren, C., Verma, S., et al. (2019). The replica dataset: A digital replica of indoor spaces. *arXiv preprint*.

Sucar, E., Liu, S., Ortiz, J., and Davison, A. J. (2021). imap: Implicit mapping and positioning in real-time. In *International Conference on Computer Vision*.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems*.

Szot, A., Clegg, A., Undersander, E., Wijmans, E., Zhao, Y., Turner, J., Maestre, N., Mukadam, M., Chaplot, D., Maksymets, O., Gokaslan, A., Vondrus, V., Dharur, S., Meier, F., Galuba, W., Chang, A., Kira, Z., Koltun, V., Malik, J., Savva, M., and Batra, D. (2021). Habitat 2.0: Training home assistants to rearrange their habitat. In *Neural Information Processing Systems*.

Tancik, M., Casser, V., Yan, X., Pradhan, S., Mildenhall, B., Srinivasan, P. P., Barron, J. T., and Kretzschmar, H. (2022). Block-nerf: Scalable large scene neural view synthesis. In *Conference on Computer Vision and Pattern Recognition*.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. In *Neural Information Processing Systems*.

Tancik, M., Weber, E., Ng, E., Li, R., Yi, B., Kerr, J., Wang, T., Kristoffersen, A., Austin, J., Salahi, K., et al. (2023). Nerfstudio: A modular framework for neural radiance field development. In *SIGGRAPH*.

Tankovich, V., Hane, C., Zhang, Y., Kowdle, A., Fanello, S., and Bouaziz, S. (2021). Hitnet: Hierarchical iterative tile refinement network for real-time stereo matching. In *Conference on Computer Vision and Pattern Recognition*.

Tassa, Y., Doron, Y., Muldal, A., Erez, T., Li, Y., Casas, D. d. L., Budden, D., Abdolmaleki, A., Merel, J., Lefrancq, A., et al. (2018). Deepmind control suite. *arXiv preprint*.

Tatarchenko, M., Dosovitskiy, A., and Brox, T. (2017). Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs. In *International Conference on Computer Vision*.

Thelen, E., Schöner, G., Scheier, C., and Smith, L. B. (2001). The dynamics of embodiment: A field theory of infant perseverative reaching. *Behavioral and brain sciences*.

Thelen, E. and Smith, L. B. (1994). *A dynamic systems approach to the development of cognition and action*. MIT press.

Thomason, J., Murray, M., Cakmak, M., and Zettlemoyer, L. (2020). Vision-and-dialog navigation. In *Conference on Robot Learning*.

Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*.

Thrun, S. (2003). Learning occupancy grid maps with forward sensor models. *Autonomous robots*.

Thrun, S., Burgard, W., Fox, D., et al. (2005). Probabilistic robotics.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*.

Torralba, A., Isola, P., and Freeman, W. T. (2024). *Foundations of Computer Vision*. MIT Press.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. (2021). Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*.

Truong, J., Rudolph, M., Yokoyama, N. H., Chernova, S., Batra, D., and Rai, A. (2023a). Re-

thinking sim2real: Lower fidelity simulation leads to higher sim2real transfer in navigation. In *Conference on Robot Learning*.

Truong, P., Rakotosaona, M.-J., Manhardt, F., and Tombari, F. (2023b). Sparf: Neural radiance fields from sparse and noisy poses. In *Conference on Computer Vision and Pattern Recognition*.

Unterthiner, T., Keysers, D., Gelly, S., Bousquet, O., and Tolstikhin, I. (2020). Predicting neural network accuracy from weights. *arXiv preprint*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Neural Information Processing Systems*.

Vincent, P., Larochelle, H., Bengio, Y., and Manzagol, P.-A. (2008). Extracting and composing robust features with denoising autoencoders. In *International Conference on Machine Learning*.

Voita, E., Talbot, D., Moiseev, F., Sennrich, R., and Titov, I. (2019). Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Annual Meeting of the Association for Computational Linguistics*.

Vora, S., Radwan, N., Greff, K., Meyer, H., Genova, K., Sajjadi, M. S., Pot, E., Tagliasacchi, A., and Duckworth, D. (2021). Nesf: Neural semantic fields for generalizable semantic segmentation of 3d scenes. *arXiv preprint*.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. (2023). Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*.

Wang, X., Fouhey, D., and Gupta, A. (2015). Designing deep networks for surface normal estimation. In *Conference on Computer Vision and Pattern Recognition*.

Wang, X., Girshick, R., Gupta, A., and He, K. (2018). Non-local neural networks. In *Conference on Computer Vision and Pattern Recognition*.

Wani, S., Patel, S., Jain, U., Chang, A. X., and Savva, M. (2020). Multion: Benchmarking semantic map memory using multi-object navigation. In *Neural Information Processing Systems*.

Warren, W. H., Rothman, D. B., Schnapp, B. H., and Ericson, J. D. (2017). Wormholes in virtual space: From cognitive maps to cognitive graphs. *Cognition*.

Weinzaepfel, P., Leroy, V., Lucas, T., Brégier, R., Cabon, Y., Arora, V., Antsfeld, L., Chidlovskii, B., Csurka, G., and Revaud, J. (2022). Croco: Self-supervised pre-training for 3d vision tasks by cross-view completion. In *Neural Information Processing Systems*.

Wijmans, E., Kadian, A., Morcos, A., Lee, S., Essa, I., Parikh, D., Savva, M., and Batra, D.

(2019). Dd-ppo: Learning near-perfect pointgoal navigators from 2.5 billion frames. In *International Conference on Learning Representations*.

Wijmans, E., Savva, M., Essa, I., Lee, S., Morcos, A. S., and Batra, D. (2022). Emergence of maps in the memories of blind navigation agents. In *International Conference on Learning Representations*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*.

Wiyatno, R. R., Xu, A., and Paull, L. (2022). Lifelong topological visual navigation. *Robotics and Automation Letters*.

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Conference on Computer Vision and Pattern Recognition*.

Wüthrich, M., Widmaier, F., Grimminger, F., Akpo, J., Joshi, S., Agrawal, V., Hammoud, B., Khadiv, M., Bogdanovic, M., Berenz, V., et al. (2020). Trifinger: An open-source robot for learning dexterity. *arXiv preprint*.

Xia, F., Zamir, A. R., He, Z., Sax, A., Malik, J., and Savarese, S. (2018). Gibson env: Real-world perception for embodied agents. In *Conference on Computer Vision and Pattern Recognition*.

Xiang, F., Qin, Y., Mo, K., Xia, Y., Zhu, H., Liu, F., Liu, M., Jiang, H., Yuan, Y., Wang, H., et al. (2020). Sapien: A simulated part-based interactive environment. In *Conference on Computer Vision and Pattern Recognition*.

Xie, Y., Takikawa, T., Saito, S., Litany, O., Yan, S., Khan, N., Tombari, F., Tompkin, J., Sitzmann, V., and Sridhar, S. (2022). Neural fields in visual computing and beyond. In *Computer Graphics Forum*.

Xu, K., Zheng, L., Yan, Z., Yan, G., Zhang, E., Niessner, M., Deussen, O., Cohen-Or, D., and Huang, H. (2017). Autonomous reconstruction of unknown indoor scenes guided by time-varying tensor fields. *Transactions on Graphics*.

Yadav, K., Majumdar, A., Ramrakhya, R., Yokoyama, N., Baevski, A., Kira, Z., Maksymets, O., and Batra, D. (2023a). Ovrl-v2: A simple state-of-art baseline for imagenav and objectnav. *arXiv preprint*.

Yadav, K., Ramrakhya, R., Majumdar, A., Berges, V.-P., Kuhar, S., Batra, D., Baevski, A., and Maksymets, O. (2022). Offline visual representation learning for embodied navigation. *arXiv preprint*.

Yadav, K., Ramrakhya, R., Ramakrishnan, S. K., Gervet, T., Turner, J., Gokaslan, A., Maestre,

N., Chang, A. X., Batra, D., Savva, M., et al. (2023b). Habitat-matterport 3d semantics dataset. In *Conference on Computer Vision and Pattern Recognition*.

Yak, S., Gonzalvo, J., and Mazzawi, H. (2019). Towards task and architecture-independent generalization gap predictors. *arXiv preprint*.

Yamauchi, B. (1997). A frontier-based approach for autonomous exploration. In *International Symposium on Computational Intelligence in Robotics and Automation*.

Yamauchi, B. and Beer, R. (1996). Spatial learning for navigation in dynamic environments. *Transactions on Systems, Man, and Cybernetics*.

Yan, C., Qu, D., Xu, D., Zhao, B., Wang, Z., Wang, D., and Li, X. (2024). Gs-slam: Dense visual slam with 3d gaussian splatting. In *Conference on Computer Vision and Pattern Recognition*.

Yan, Z., Yang, H., and Zha, H. (2023). Active neural mapping. In *International Conference on Computer Vision*.

Yang, Y., Sun, F.-Y., Weihs, L., VanderBilt, E., Herrasti, A., Han, W., Wu, J., Haber, N., Krishna, R., Liu, L., et al. (2024). Holodeck: Language guided generation of 3d embodied ai environments. In *Conference on Computer Vision and Pattern Recognition*.

Ye, J., Batra, D., Das, A., and Wijmans, E. (2021). Auxiliary tasks and exploration enable objectnav. *arXiv preprint*.

Ye, J., Batra, D., Wijmans, E., and Das, A. (2020). Auxiliary tasks speed up learning pointgoal navigation. In *Conference on Robot Learning*.

Yen-Chen, L., Florence, P., Barron, J. T., Rodriguez, A., Isola, P., and Lin, T.-Y. (2021). inerf: Inverting neural radiance fields for pose estimation. In *International Conference on Intelligent Robots and Systems*.

Yenamandra, S., Ramachandran, A., Yadav, K., Wang, A., Khanna, M., Gervet, T., Yang, T.-Y., Jain, V., Clegg, A. W., Turner, J., et al. (2023). Homerobot: Open-vocabulary mobile manipulation. In *Conference on Robot Learning*.

Yeshwanth, C., Liu, Y.-C., Nießner, M., and Dai, A. (2023). Scannet++: A high-fidelity dataset of 3d indoor scenes. In *International Conference on Computer Vision*.

Yin, W., Zhang, J., Wang, O., Niklaus, S., Mai, L., Chen, S., and Shen, C. (2021). Learning to recover 3d scene shape from a single image. In *Conference on Computer Vision and Pattern Recognition*.

Yu, A., Ye, V., Tancik, M., and Kanazawa, A. (2021). pixelnerf: Neural radiance fields from one or few images. In *Conference on Computer Vision and Pattern Recognition*.

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2020). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*.

Zbontar, J., Jing, L., Misra, I., LeCun, Y., and Deny, S. (2021). Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*.

Zeng, J., Li, Y., Ran, Y., Li, S., Gao, F., Li, L., He, S., Chen, J., and Ye, Q. (2023). Efficient view path planning for autonomous implicit reconstruction. In *International Conference on Robotics and Automation*.

Zhai, A. J. and Wang, S. (2023). Peanut: predicting and navigating to unseen targets. In *International Conference on Computer Vision*.

Zhan, H., Zheng, J., Xu, Y., Reid, I., and Rezatofighi, H. (2022). Activermap: Radiance field for active mapping and planning. *arXiv preprint*.

Zhang, L., Rao, A., and Agrawala, M. (2023). Adding conditional control to text-to-image diffusion models. In *International Conference on Computer Vision*.

Zhang, R., Isola, P., Efros, A. A., Shechtman, E., and Wang, O. (2018). The unreasonable effectiveness of deep features as a perceptual metric. In *Conference on Computer Vision and Pattern Recognition*.

Zhi, S., Laidlow, T., Leutenegger, S., and Davison, A. J. (2021a). In-place scene labelling and understanding with implicit scene representation. In *International Conference on Computer Vision*.

Zhi, S., Sucar, E., Mouton, A., Haughton, I., Laidlow, T., and Davison, A. J. (2021b). ilabel: Interactive neural scene labelling. *arXiv preprint*.

Zhmoginov, A., Sandler, M., and Vladymyrov, M. (2022). Hypertransformer: Model generation for supervised and semi-supervised few-shot learning. In *International Conference on Machine Learning*.

Zhou, A., Finn, C., and Harrison, J. (2024). Universal neural functionals. *arXiv preprint*.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017a). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *International Conference on Robotics and Automation*.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., and Farhadi, A. (2017b). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *International Conference on Robotics and Automation*.

Zhu, Z., Peng, S., Larsson, V., Cui, Z., Oswald, M. R., Geiger, A., and Pollefeys, M. (2024). Nicer-slam: Neural implicit scene encoding for rgb slam. In *International Conference on 3D Vision*.

Zhu, Z., Peng, S., Larsson, V., Xu, W., Bao, H., Cui, Z., Oswald, M. R., and Pollefeys, M. (2022). Nice-slam: Neural implicit scalable encoding for slam. In *Conference on Computer Vision and Pattern Recognition*.

Zitkovich, B., Yu, T., Xu, S., Xu, P., Xiao, T., Xia, F., Wu, J., Wohlhart, P., Welker, S., Wahid, A., et al. (2023). Rt-2: Vision-language-action models transfer web knowledge to robotic control. In *Conference on Robot Learning*.

FOLIO ADMINISTRATIF

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

Nom : Marza

Date de soutenance: 25 Novembre 2024

Prénom : Pierre

Titre : Learning spatial representations for single-task navigation and multi-task policies

Nature : Doctorat

Numéro d'ordre : 2024ISAL0105

Ecole doctorale : Infomaths

Spécialité : Informatique

Résumé : Agir de manière autonome dans notre monde 3D requiert un large éventail de compétences, parmi lesquelles se trouvent la perception du milieu environnant, sa représentation précise et suffisamment efficace pour garder une trace du passé, la prise de décisions et l'action en vue d'atteindre des objectifs. Les animaux, par exemple les humains, se distinguent par leur robustesse lorsqu'il s'agit d'agir dans le monde. En particulier, ils savent s'adapter efficacement à de nouveaux environnements, mais sont aussi capables de maîtriser rapidement de nombreuses tâches à partir de quelques exemples. Ce manuscrit étudiera comment les réseaux neuronaux artificiels peuvent être entrainés pour acquérir un sous-ensemble de ces capacités. Nous nous concentrerons tout d'abord sur l'entrainement d'agents neuronaux à la cartographie sémantique, à la fois à partir d'un signal de supervision augmenté et avec des représentations neuronales de scènes. Les agents neuronaux sont souvent entrainés par apprentissage par renforcement (RL) à partir d'un signal de récompense peu dense. Guider l'apprentissage des capacités de cartographie d'une scène en ajoutant au signal de supervision des tâches auxiliaires favorisant le raisonnement spatial aidera à naviguer plus efficacement. Au lieu de travailler sur le signal d'entrainement des agents neuronaux, nous verrons également comment l'incorporation de représentations neuronales spécifiques de la sémantique et de la géométrie à l'architecture de l'agent peut contribuer à améliorer les performances de navigation sémantique. Ensuite, nous étudierons la meilleure façon d'explorer un environnement 3D afin de construire des représentations neuronales de l'espace qui soient aussi satisfaisantes que possible sur la base de métriques pensées pour la robotique que nous proposerons. Enfin, nous passerons d'agents de navigation à des agents multi-tâches et nous verrons à quel point il est important d'adapter les caractéristiques visuelles extraites des observations de capteurs à chaque tâche à accomplir pour réaliser une variété de tâches, mais aussi pour s'adapter à de nouvelles tâches inconnues à partir de quelques démonstrations.

Mots-clés : Apprentissage profond, Vision par ordinateur, Apprentissage par Renforcement, Représentations spatiales, Navigation Visuelle, Politiques multi-tâches

Laboratoires de recherche : LIRIS, CITI

Directeur de thèse : Olivier Simonin

Président du jury : Nicolas Thome

Composition du jury : Ivan Laptev, Karteek Alahari, Georgia Chalvatzaki, Nicolas Thome, Laëtitia Matignon, Olivier Simonin, Christian Wolf